

Міністерство освіти і науки України
Національний університет
«Львівська політехніка»
Кафедра електронних обчислювальних машин

КУРСОВИЙ ПРОЄКТ

з дисципліни “Системне програмне забезпечення”

на тему: “Розробка утиліти для моніторингу системи. розробка програми, що дозволяє моніторити систему на комп'ютері з операційною системою windows та отримувати інформацію про поточний стан системи”

Студента 3-го курсу групи KI-306
123 «Комп'ютерна інженерія»
Ярмола Ю. Ю.
Керівник
Олексів М. В.

Національна шкала: _____
Кількість балів: _____
Оцінка ECTS: _____

Члени комісії:	_____	_____
	(підпис)	(прізвище та ініціали)
	_____	_____
	(підпис)	(прізвище та ініціали)
	_____	_____
	(підпис)	(прізвище та ініціали)

Львів 2024

Завдання на курсовий проект

Основною метою цього курсового проекту є розробка та впровадження програми "MoniSy" що розшифровується як Monitoring System, яка забезпечить користувачам детальну інформацію про стан їхньої системи в реальному часі. Програма буде оснащена функціями для моніторингу використання ресурсів, аналізу поточних процесів та системних подій, що дозволить вчасно виявляти проблеми з продуктивністю та ефективно керувати ресурсами операційної системи Windows.

Завдання проекту

1. Аналіз існуючих рішень
2. Розробка архітектури програми
3. Реалізація функціональних можливостей
 - a. Моніторинг використання ресурсів комп'ютера з частотою оновлення 1 секунда
 - b. Моніторинг температури основного та графічного процесорів з частотою оновлення 1 секунда
 - c. Перегляд активних запущених процесів на комп'ютері, та змога сортування їх по назві, PID та використанню процесора у %
 - d. Перегляд системних журналів зі змогою їх сортування
 - e. Перегляд використання мереж
 - f. Має бути зручний, зрозумілий та сучасний інтерфейс для користувача
4. Провести тестування та відлагодження, усунути знайдені проблеми
5. Створити чітку та детальну документацію про проект

Очікувані результати:

Результатом виконання курсового проекту має стати функціональна програма "MoniSy", яка буде здатна ефективно моніторити та аналізувати ресурси операційної системи Windows. Програма повинна бути зручною у використанні, надійною та надавати користувачам усю необхідну інформацію для оптимізації роботи їхніх систем.

Анотація

У даній курсовій роботі представлено розробку утиліти для моніторингу системи на комп'ютерах з операційною системою Windows. Основною метою цієї утиліти є надання користувачам детальної інформації про поточний стан системи, що включає використання центрального процесора (CPU), оперативної пам'яті (RAM), стан жорстких дисків, мережеву активність та інші ключові параметри.

У роботі розглянуто основні методи та інструменти системного програмування, що використовуються для отримання та обробки системної інформації на платформі Windows. Описано процес розробки утиліти, включаючи вибір мови програмування, структуру програми, реалізацію основних функцій та інтерфейс користувача.

Розроблена утиліта має зручний графічний інтерфейс, що дозволяє користувачам легко відстежувати та аналізувати стан системи в режимі реального часу. Особлива увага приділена ефективності роботи утиліти та мінімізації її впливу на продуктивність системи.

Результати тестування показують, що утиліта успішно виконує поставлені завдання та може бути корисним інструментом для адміністраторів систем, розробників програмного забезпечення та інших користувачів, які потребують надійної інформації про стан системи.

Зміст

Завдання на курсовий проект	2
Завдання проекту	2
Очікувані результати:	2
Анотація	3
Зміст	4
Вступ	6
1. АНАЛІТИЧНИЙ ОГЛЯД	7
1.1 Аналіз методів для моніторингу системи	7
1.2 Відомі розробки	8
1.3 Цільова аудиторія	9
2. Проектування програми	12
2.1 Обґрунтування засобів для розробки	12
2.2 Високорівневий огляд програми	12
2.3 Архітектура програми	13
2.4 Функціональні можливості	14
3. Реалізація	17
3.1 Реалізація графічного інтерфейсу користувача	17
3.2 Високорівневий опис реалізації програмного рішення згідно розроблених вимог	22
3.3 Реалізація діаграми класів та функцій моніторингу, їх призначення і взаємодія	23
3.4 Реалізація функцій інтерфейсу для взаємодії з користувачами	25
3.6 Опис алгоритму роботи графічного інтерфейсу	27
4. Тестування	29
5. Висновки	36
Список використаних джерел	37
Додатки	38
Додаток 1 – Оцінка виконання програми за допомогою User Story	38
Додаток 2 – Експорт програми в exe	43
Додаток 3 – Вихідний код програми	44
Main.py	44
functions.py	55
charts.py	60
WinTmp.py	65

Вступ

Сучасні комп'ютерні системи є складними комплексами апаратних і програмних компонентів, які працюють у тісній взаємодії. Забезпечення стабільної та ефективної роботи таких систем вимагає постійного моніторингу їхнього стану. Особливо це актуально для серверів, робочих станцій та інших критичних систем, де будь-яка несправність або зниження продуктивності може призвести до значних втрат даних або ресурсів.

Моніторинг системи дозволяє вчасно виявляти та усувати проблеми, аналізувати навантаження на апаратні ресурси, а також оптимізувати роботу програмного забезпечення. У зв'язку з цим існує потреба у надійних та зручних інструментах, які б надавали користувачам детальну інформацію про поточний стан системи.

Дана курсова робота присвячена розробці утиліти для моніторингу системи на комп'ютерах з операційною системою Windows. Утиліта надає користувачам можливість отримувати дані про використання центрального процесора, оперативної пам'яті, жорстких дисків, мережеву активність та інші ключові параметри системи в режимі реального часу.

Метою роботи є створення ефективного інструменту для моніторингу системи, який буде корисним як для звичайних користувачів, так і для системних адміністраторів та розробників програмного забезпечення. У процесі розробки утиліти будуть використані сучасні методи та технології системного програмування, що дозволить забезпечити високу продуктивність та надійність роботи програми.

У роботі розглядаються теоретичні аспекти системного моніторингу, методи збору та аналізу системної інформації, а також практичні питання реалізації програмного забезпечення для моніторингу системи. Результати даного дослідження можуть бути використані для подальшого розвитку інструментів моніторингу та оптимізації роботи комп'ютерних систем.

1. АНАЛІТИЧНИЙ ОГЛЯД

Моніторинг системи – це процес спостереження за станом та продуктивністю комп'ютерної системи з метою забезпечення її стабільної та ефективної роботи. Основні показники, які зазвичай моніторяться, включають використання центрального процесора (CPU), оперативної пам'яті (RAM), жорстких дисків, мережеву активність та інші ключові параметри. Розробка утиліти для моніторингу системи на платформі Windows потребує застосування специфічних методів і технологій, що дозволяють отримувати та обробляти відповідну інформацію в реальному часі.

1.1 Аналіз методів для моніторингу системи

Для отримання детальної інформації про стан операційної системи Windows використовуються різні системні API, що дозволяють доступ до низькорівневих даних:

1. **Системні API Windows:** Забезпечують доступ до основних функцій операційної системи, дозволяючи отримувати детальну інформацію про стан різних компонентів системи.
2. **Performance Counters API:** Використовується для збору даних про продуктивність різних компонентів системи, таких як CPU, пам'ять, дискові накопичувачі та мережеву активність. Це дозволяє отримувати поточні значення та статистичні дані для аналізу продуктивності.
3. **Windows Management Instrumentation (WMI):** Надає розширені можливості для управління та моніторингу системи. За допомогою WMI можна виконувати запити для отримання інформації про апаратне забезпечення, встановлене програмне забезпечення, мережеві налаштування та інші системні параметри.
4. **QueryPerformanceCounter:** Використовується для високоточних вимірювань часу. Ця функція дозволяє здійснювати точні таймери, що є корисним при детальному аналізі продуктивності додатків та системи загалом.

Застосування цих методів дозволяє отримувати всебічну та детальну інформацію про стан системи, що є необхідним для ефективного моніторингу та аналізу продуктивності.

Використання бібліотек та фреймворків

Існують готові бібліотеки та фреймворки, які значно спрощують процес розробки утиліт для моніторингу системи. Вони надають високорівневі інтерфейси для доступу до системної інформації, що знижує складність розробки та дозволяє зосередитися на функціональних аспектах програми.

1. **.NET System.Diagnostics:**

- **Опис:** Набір класів у середовищі .NET, які дозволяють отримувати інформацію про процеси, продуктивність системи та інші аспекти.
- **Переваги:** Простота використання, інтеграція з .NET, можливість збирання детальної інформації про процеси та продуктивність.

2. **Open Hardware Monitor:**

- **Опис:** Безкоштовне програмне забезпечення з відкритим вихідним кодом, яке дозволяє моніторити температури, напруги, швидкість вентиляторів та завантаження процесора.
- **Переваги:** Відкритий код, підтримка широкого спектру апаратного забезпечення, можливість розширення функціональності.

3. **Performance Data Helper (PDH):**

- **Опис:** API для доступу до даних про продуктивність Windows, що дозволяє легко збирати та інтерпретувати дані.
- **Переваги:** Доступ до детальних даних про продуктивність, можливість створення власних інтерпретацій та звітів.

Використання цих бібліотек та фреймворків забезпечує розробникам зручні інструменти для швидкого та ефективного створення утиліт для моніторингу системи, дозволяючи зосередитися на розробці специфічних функцій та покращенні користувацького досвіду.

1.2 Відомі розробки

Системні утиліти Windows

1. **Task Manager:**

- **Опис:** Вбудований інструмент Windows, який надає базову інформацію про використання CPU, пам'яті, диска та мережі. Task Manager також дозволяє управляти запущеними процесами та службами.
- **Переваги:** Легкий доступ, інтуїтивно зрозумілий інтерфейс, базові можливості управління процесами.

2. **Resource Monitor:**

- **Опис:** Більш розширений інструмент, який дозволяє детально аналізувати використання ресурсів системи, включаючи активність дисків, мережі та пам'яті.
- **Переваги:** Детальний аналіз ресурсів, можливість спостереження за активністю окремих компонентів системи.

Комерційні та відкриті утиліти

1. HWMonitor:

- **Опис:** Популярна утиліта для моніторингу температури та напруги компонентів комп'ютера. Вона підтримує широкий спектр апаратного забезпечення та надає детальну інформацію про його стан.
- **Переваги:** Підтримка великої кількості апаратних компонентів, детальна інформація про температуру та напругу.

2. AIDA64:

- **Опис:** Потужний інструмент для діагностики та моніторингу системи, який пропонує широкі можливості для тестування та отримання інформації про всі аспекти комп'ютера.
- **Переваги:** Широкі діагностичні можливості, детальна інформація про апаратні та програмні компоненти, можливість проведення тестів продуктивності.

3. Open Hardware Monitor:

- **Опис:** Безкоштовне програмне забезпечення з відкритим вихідним кодом, яке дозволяє моніторити широкий спектр показників апаратного забезпечення.
- **Переваги:** Відкритий вихідний код, можливість розширення функціональності, підтримка широкого спектру апаратного забезпечення.

1.3 Цільова аудиторія

Програма "MoniSy" адресована широкому колу користувачів операційної системи Windows. Вона розроблена для задоволення потреб різних категорій користувачів, включаючи досвідчених ентузіастів, системних адміністраторів та звичайних користувачів, які стикаються з проблемами продуктивності. Нижче подано детальний опис кожної цільової аудиторії:

Досвідчені користувачі

- **Опис:** Це категорія користувачів, які добре розуміються на технічних аспектах роботи операційної системи Windows та прагнуть отримати детальну інформацію про роботу своєї системи.

- **Потреби:** Оптимізація продуктивності системи, моніторинг використання ресурсів у реальному часі, виявлення вузьких місць у роботі комп'ютера.
- **Функції, що задовольняють потреби:** Детальний моніторинг процесора (CPU), пам'яті, дискового простору та інших ключових компонентів системи. Можливість налаштовувати та отримувати сповіщення про перевищення певних порогових значень використання ресурсів.

Системні адміністратори

- **Опис:** Професіонали, які відповідають за моніторинг та підтримку великої кількості комп'ютерів або серверів під управлінням Windows. Вони забезпечують безперебійну роботу IT-інфраструктури організацій.
- **Потреби:** Централізоване управління та моніторинг численних систем, швидке виявлення та усунення проблем з продуктивністю, забезпечення стабільності та безпеки мережі.
- **Функції, що задовольняють потреби:** Можливість моніторингу в реальному часі з централізованим контролем та віддаленим доступом до даних, деталізовані звіти та аналітика, функціонал для виявлення і усунення збоїв і проблем.

Користувачі з проблемами продуктивності

- **Опис:** Користувачі, які часто стикаються з проблемами продуктивності або виявляють неефективне використання ресурсів системи. Вони можуть не мати глибоких технічних знань, але бажають покращити роботу свого комп'ютера.
- **Потреби:** Ідентифікація та усунення проблем, які негативно впливають на продуктивність, простий у використанні інтерфейс для моніторингу стану системи, допомога у розумінні використання ресурсів.
- **Функції, що задовольняють потреби:** Зрозумілий та інтуїтивно зрозумілий інтерфейс, функції автоматичного виявлення та виправлення проблем, рекомендації щодо оптимізації використання ресурсів.

Користувачі, які бажають отримати дані про систему охолодження та навантаження ресурсів

- **Опис:** Користувачі, які бажають отримати детальну інформацію про систему охолодження, напруги, температури та навантаження різних компонентів, щоб забезпечити ефективну роботу свого комп'ютера.
- **Потреби:** Моніторинг температури компонентів, ефективність системи охолодження, виявлення проблем з перегрівом, контроль за напругою та іншими параметрами.
- **Функції, що задовольняють потреби:** Відображення температури, напруги та швидкості вентиляторів у реальному часі, сповіщення про перегрів чи інші критичні стани, детальні графіки та історичні дані для аналізу ефективності охолодження.

Програма "MoniSy" розроблена з урахуванням потреб кожної з цих груп користувачів, що дозволяє їм ефективно моніторити та аналізувати стан своїх систем, забезпечуючи стабільну та оптимальну роботу комп'ютерів та серверів.

Висновки

Моніторинг системи є важливим аспектом забезпечення стабільної та ефективної роботи комп'ютерів. Існує широкий спектр методів та інструментів для отримання інформації про стан системи на платформі Windows. Використання системних API, таких як Performance Counters, WMI, та готових бібліотек, дозволяє розробникам створювати потужні утиліти для моніторингу. Розробка власної утиліти для моніторингу системи на базі цих методів та інструментів дозволяє отримати гнучкий та зручний інструмент для спостереження за станом комп'ютерної системи, що може бути корисним як для звичайних користувачів, так і для професійних адміністраторів та розробників. З огляду на переглянуті рішення, щоб бути конкурентною, програма повинна вміти зчитувати дані процесора, температуру, переглядати стан комп'ютера та відображати актуальні характеристики системи.

2. Проектування програми

2.1 Обґрунтування засобів для розробки

Розробка утиліти для моніторингу системи на базі Python із використанням бібліотек Flet, platform, GPUtil, psutil, WinTmp та win32evtlog має на меті створення інструменту, який надаватиме користувачам детальну інформацію про стан їхньої операційної системи Windows. Ця утиліта буде корисною як для звичайних користувачів, так і для професійних системних адміністраторів.

Вибір технологій

Python було обрано як основну мову програмування завдяки її зручності, багатству бібліотек та активній спільноті. Python дозволяє швидко розробляти та тестувати код, що прискорює процес розробки.

Flet забезпечує створення кросплатформених графічних інтерфейсів користувача, що робить програму зручною та доступною. Завдяки Flet можна легко створити інтуїтивно зрозумілий інтерфейс, який забезпечить зручність використання утиліти.

platform дозволяє отримати базову інформацію про операційну систему, таку як версія ОС, архітектура процесора та інші важливі параметри. Це допомагає налаштувати утиліту під специфічні потреби користувача.

GPUtil використовується для моніторингу графічного процесора. Вона дозволяє отримувати інформацію про завантаження, температуру та інші показники роботи GPU, що є важливим для користувачів, які займаються графічними додатками або іграми.

psutil надає широкий спектр інструментів для моніторингу системних ресурсів, таких як CPU, пам'ять, диски та мережа. Вона дозволяє отримувати детальну інформацію про стан системи в реальному часі.

WinTmp забезпечує доступ до даних про температуру системи, що є важливим для запобігання перегріву та підтримки стабільної роботи комп'ютера.

win32evtlog використовується для читання системних журналів подій Windows. Це дозволяє аналізувати події, що відбуваються в системі, та своєчасно виявляти потенційні проблеми.

LibreHardwareMonitor надає розширені можливості для моніторингу апаратного забезпечення, включаючи температури, напруги, швидкість вентиляторів та інші параметри. Вона підтримує широкий спектр апаратного забезпечення і є ефективним інструментом для отримання детальної інформації про апаратний стан системи.

2.2 Високорівневий огляд програми

Програма "MoniSy" розроблена для моніторингу та аналізу операційної системи Windows. Її основна мета - забезпечити користувачам детальну інформацію про стан їхньої системи в реальному часі. Програма має широкий

спектр функцій, включаючи моніторинг використання ресурсів, аналіз поточних процесів та системних подій. Нижче наведено детальний опис архітектури та функціональних можливостей програми.

2.3 Архітектура програми

Програма "MoniSy" складається з кількох ключових компонентів, які взаємодіють між собою для забезпечення ефективного моніторингу та аналізу системи:

1. Інтерфейс користувача (UI):

- **Опис:** Інтерфейс користувача розроблений з використанням бібліотеки Flet, яка забезпечує створення кросплатформових графічних інтерфейсів. Інтерфейс інтуїтивно зрозумілий та зручний у використанні.
- **Функції:** Відображення даних у реальному часі, графіки використання ресурсів, сповіщення про критичні події, налаштування параметрів моніторингу.

2. Модуль збору даних:

- **Опис:** Цей модуль відповідає за збір даних з різних компонентів системи. Він використовує кілька бібліотек для отримання детальної інформації про стан системи.
- **Бібліотеки:**
 - **platform:** Збирає базову інформацію про операційну систему, версію ОС та архітектуру процесора.
 - **GPUtil:** Моніторинг графічного процесора (GPU), включаючи завантаження, температуру та інші показники.
 - **psutil:** Моніторинг системних ресурсів, таких як CPU, пам'ять, диски та мережа.
 - **WinTmp:** Отримання даних про температуру системи.
 - **win32evtlog:** Читання системних журналів подій Windows.
 - **LibreHardwareMonitor:** Моніторинг апаратного забезпечення, включаючи температури, напруги та швидкість вентиляторів.

3. Модуль обробки даних:

- **Опис:** Цей модуль відповідає за обробку зібраних даних та перетворення їх у зручний для користувача формат. Він також виконує аналіз даних для виявлення потенційних проблем.
- **Функції:** Аналіз продуктивності, виявлення аномалій, обробка подій системних журналів, обчислення середніх значень та історичних трендів.

4. Модуль зберігання даних:

- **Опис:** Відповідає за зберігання історичних даних для подальшого аналізу та побудови графіків.

- **Функції:** Зберігання даних у локальній базі даних або файлах, забезпечення доступу до історичних даних для аналізу трендів та продуктивності.

2.4 Функціональні можливості

1. Моніторинг використання CPU

Опис: Функція моніторингу використання CPU надає користувачам інформацію про завантаження процесора в реальному часі. Це включає збір даних про використання кожного ядра процесора та загального завантаження системи.

Вимоги:

- Збір та відображення інформації про завантаження процесора в реальному часі для ефективного контролю над ресурсами.
- Збір даних про завантаження кожного з ядер процесора в реальному часі.
- Відображення стану завантаження кожного з ядер процесора з мінімальною затримкою.
- Надання можливості перегляду інформації про завантаження окремого ядра процесора.
- Відображення середнього завантаження всіх ядер процесора.
- Відображення стану завантаження у вигляді графіку.
- Надання можливості отримання детальної інформації по завантаженню процесора за допомогою миші.

2. Відслідковування використання пам'яті

Опис: Ця функція забезпечує моніторинг використання оперативної та віртуальної пам'яті, як для всієї системи загалом, так і для окремих процесів.

Вимоги:

- Показувати статистику використання оперативної та віртуальної пам'яті для контролю над ресурсами.
- Відображення використання оперативної та віртуальної пам'яті в реальному часі.
- Перегляд загального обсягу використаної та доступної пам'яті.
- Моніторинг розподілу пам'яті між процесами та задачами.
- Надання сповіщень у випадку перевищення певних порогових значень використання пам'яті.

3. Моніторинг дискового простору

Опис: Моніторинг дискового простору допомагає користувачам отримати інформацію про використання та вільне місце на дисках, а також можливість оптимізації використання дискового простору.

Вимоги:

- Відображення використаного та вільного місця на дисках в реальному часі.
- Моніторинг активності дисків, включаючи швидкість читання/запису.
- Надання детальної інформації про кожен диск, включаючи тип файлової системи, загальну та вільну ємність.
- Сповіщення користувача про низький рівень вільного місця на диску.

4. Моніторинг графічного процесора (GPU)

Опис: Функція моніторингу GPU забезпечує збір та відображення даних про завантаження графічного процесора, температуру та інші важливі параметри.

Вимоги:

- Збір інформації про завантаження GPU в реальному часі.
- Відображення температури, використання пам'яті та інших ключових параметрів GPU.
- Надання можливості перегляду історичних даних для аналізу навантаження GPU.
- Сповіщення про перегрів чи інші критичні стани GPU.

5. Аналіз поточних процесів

Опис: Ця функція дозволяє відображати та аналізувати поточні процеси, їхнє завантаження на системні ресурси, а також надавати можливість керування процесами.

Вимоги:

- Відображення списку всіх запущених процесів із інформацією про їхнє завантаження CPU, пам'яті та дисків.
- Можливість завершення або зупинки небажаних процесів.
- Надання детальної інформації про кожен процес, включаючи шлях до виконуваного файлу та параметри командного рядка.
- Фільтрація процесів за різними критеріями (назва, використання ресурсів тощо).

6. Моніторинг системних подій

Опис: Функція моніторингу системних подій дозволяє відстежувати та аналізувати події, що відбуваються в операційній системі, шляхом читання журналів подій Windows.

Вимоги:

- Читання та відображення системних журналів подій Windows.
- Фільтрація та сортування подій за різними критеріями (тип події, час, джерело тощо).
- Надання можливості перегляду деталей кожної події.
- Сповіщення про критичні події або помилки в системі.

7. Система сповіщень

Опис: Система сповіщень інформує користувачів про важливі події, перевищення порогових значень та інші критичні ситуації.

Вимоги:

- Налаштування порогових значень для сповіщень (наприклад, температура, використання пам'яті, завантаження CPU).
- Відправка сповіщень у вигляді повідомлень на екрані або електронною поштою.
- Логи сповіщень для подальшого аналізу.

8. Візуалізація даних

Опис: Функція візуалізації даних надає графіки та діаграми для кращого розуміння використання системних ресурсів та аналізу трендів.

Вимоги:

- Відображення даних у вигляді графіків у реальному часі.
- Підтримка різних типів графіків (лінійні, стовпчикові тощо) для різних типів даних.
- Збереження історичних даних для аналізу трендів.
- Інтерактивність графіків (наприклад, можливість збільшення масштабу, перегляд деталей по точках).

9. Налаштування користувача

Опис: Функція налаштування користувача дозволяє персоналізувати параметри моніторингу та інтерфейс програми відповідно до потреб користувача.

Вимоги:

- Можливість налаштування частоти оновлення даних.
- Встановлення порогових значень для сповіщень.
- Налаштування інтерфейсу користувача (темна/світла тема, розташування панелей тощо).
- Збереження налаштувань користувача між сеансами.

Висновки

Програма "MoniSy" є потужним інструментом для моніторингу та аналізу операційної системи Windows. Вона надає користувачам детальну інформацію про стан системи в реальному часі, допомагає виявляти та усувати проблеми, що негативно впливають на продуктивність, і забезпечує стабільну та ефективну роботу комп'ютера. Використання сучасних бібліотек та технологій робить програму надійною, гнучкою та зручною у використанні.

3. Реалізація

Для реалізації проектованого рішення програми "MoniSy" я використав модуль **Flet** для побудови графічного інтерфейсу користувача. Flet забезпечує створення кросплатформових інтерфейсів, що робить програму зручною для використання на різних пристроях з операційною системою Windows.

Для отримання системних даних було використано поєднання системних та спеціалізованих модулів мови Python. Зокрема:

- **platform**: Використовується для отримання базової інформації про встановлену операційну систему, включаючи версію ОС та архітектуру процесора.
- **GPUtil**: Застосовується для збору інформації про графічний процесор (GPU), зокрема завантаження, температуру та інші показники.
- **psutil**: Дозволяє отримувати детальну інформацію про використання системних ресурсів, таких як пам'ять, процесор, диски та мережа.
- **WinTmp**: Використовується для отримання даних про температуру ядер процесора, що є критично важливим для моніторингу стану охолодження системи.
- **win32evtlog**: Виконує роль збирача інформації про системні логи, дозволяючи аналізувати події, що відбуваються в операційній системі.

Використання саме цих бібліотек та модулів було обумовлене наявністю готових рішень для ефективного отримання системної інформації та їх легкою інтеграцією в програму. Комбінування цих інструментів дозволило забезпечити збір практично всієї необхідної інформації про апаратне та програмне забезпечення комп'ютера. Це забезпечує повний контроль за станом системи, що дозволяє користувачам вчасно виявляти та усувати можливі проблеми, забезпечуючи стабільну та ефективну роботу комп'ютера.

3.1 Реалізація графічного інтерфейсу користувача

Для створення зручного та інтуїтивно зрозумілого інтерфейсу користувача в програмі "MoniSy" було використано можливості бібліотеки **Flet**. Ця бібліотека надає широкий спектр інструментів для розробки кросплатформових інтерфейсів, забезпечуючи при цьому адаптацію під різні теми оформлення та розміри екранів.

Використання стандартних значків та тем

Flet дозволяє використовувати стандартні значки, що робить інтерфейс впізнаваним та легким для навігації. Крім того, підтримка різних тем

оформлення дає можливість налаштовувати вигляд програми під уподобання користувача. Це включає як світлі, так і темні теми, що особливо важливо для зручного використання програми в різних умовах освітлення.

Обґрунтування обраних елементів дизайну кнопок переходу між вкладками

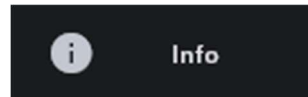


Рис. 3.1.1 – Кнопка для переходу на вкладку «інформація про систему»

Значок інформації є інтуїтивно зрозумілим для користувачів, які хочуть отримати інформацію про систему в контексті даної програми, що робить навігацію зручною та швидкою.

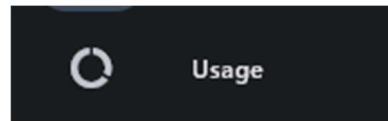


Рис. 3.1.2 – Кнопка для переходу на вкладку «Використання ресурсів»

Кнопка переходу до вкладки «Використання ресурсів» також є інтуїтивно зрозумілою, з огляду на секторний графік, який підказує що тут можна отримати інформацію що щось використовує певний відсоток чогось, а напис “Usage” доповнює зрозумілість контексту, що діаграма показуватиме відсоток використання ресурсів.

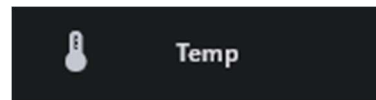


Рис. 3.1.3 – Кнопка для переходу на вкладку «Температура»

Кнопка переходу на до вкладки «Температура», є повністю зрозумілою, так як градусник це єдиний пристрій для виміру температури.

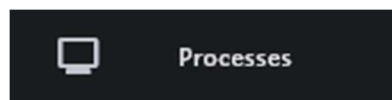


Рис. 3.1.4 – Кнопка для переходу на вкладку «Процеси»

Так як немає уставленого поняття значку «процес», я вибрав значок монітора, щоб зобразити, що щось відбувається у комп’ютері. Але завдати напису, вкладка стає зрозумілою.

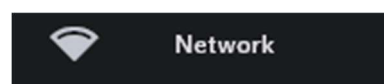


Рис. 3.1.5 – Кнопка для переходу на вкладку «Мережа»

Для відображення кнопки переходу на вкладку «Мережі», я обрав значок WI-FI, що робить інтуїтивно зрозумілим, що вкладка буде пов'язана з інтернетом, так як значок WI-FI є дуже відомим, та має широке застосування.

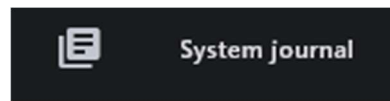


Рис. 3.1.6 – Кнопка для переходу на вкладку «Системні журнали»

Для відображення кнопки, яка переключає на вкладку «Системні журнали», я обрав значок під назвою LIBRARY_BOOKS. На мою думку, він передає ідею записів та логування, а напис робить його ще зрозумілішим.

Адаптивний дизайн

Інтерфейс програми автоматично підлаштовується під розмір екрану користувача. Це забезпечує зручне використання програми як на великих моніторах, так і на компактних екранах ноутбуків або планшетів. Всі елементи інтерфейсу динамічно змінюють свої розміри та розташування, що робить користувацький досвід максимально комфортним.



Рис. 3.1.7 – Інтерфейс при запуску

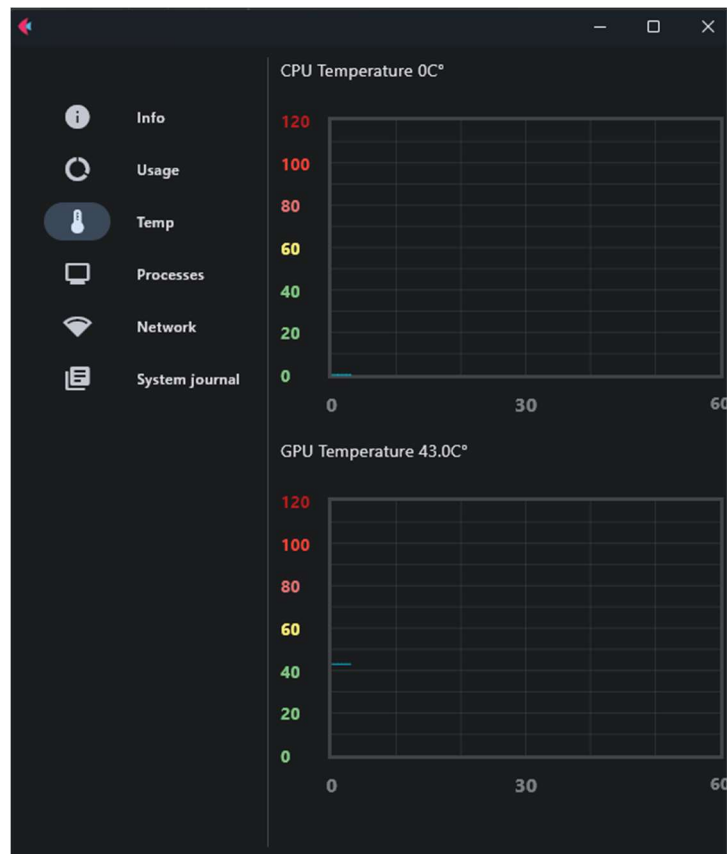


Рис. 3.1.8 – Інтерфейс при динамічній зміні вікна

Відображення інформації у реальному часі

Для відображення даних у реальному часі було прийняте рішення використовувати графіки. Це дозволяє користувачам бачити поточний стан системних ресурсів, таких як завантаження CPU, GPU, використання пам'яті та температуру. Графіки оновлюються з мінімальною затримкою, що забезпечує актуальність відображуваної інформації.

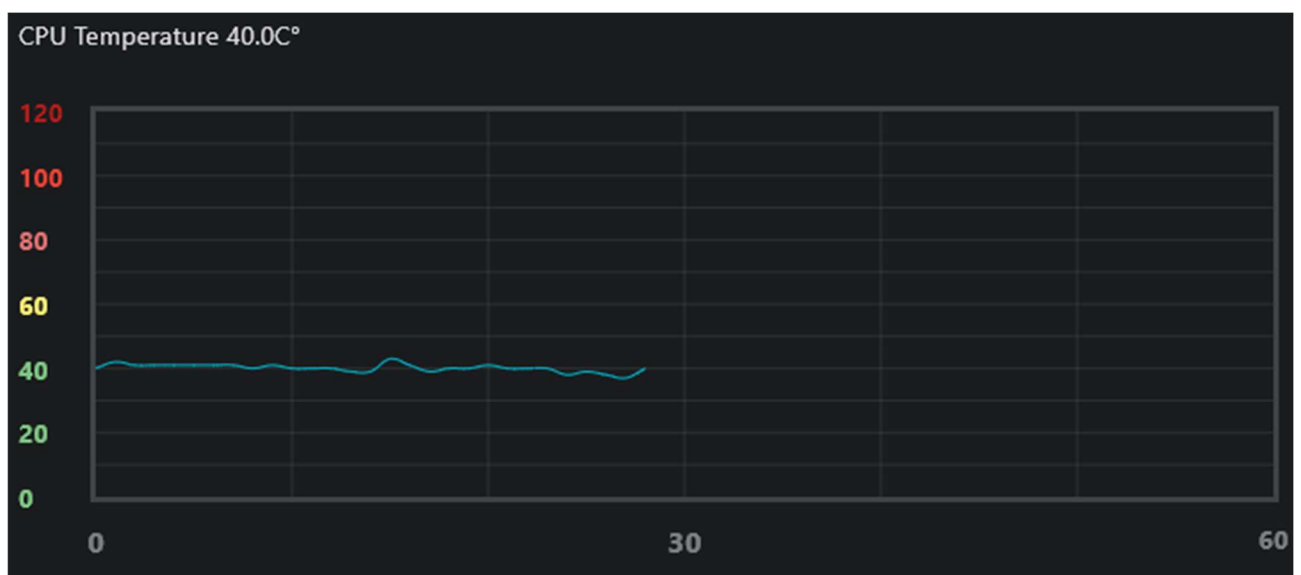


Рис. 3.1.9 – Графік температури

Як видно на рисунку, графік має дві осі та відображає температуру у певний момент часу, від відкриття вкладки. Вісь X відображає час у секундах, і має буфер в 1 хв. Вісь Y відображає температуру, та має градацію кольорів від зеленого що прийнято вважати хорошою температурою, до насиченого червоного, що означає – температура критична.

Буфер на 1 хвилину

Для тимчасового збереження та аналізу даних використовується буфер з тривалістю в 1 хвилину. Це означає, що користувачі можуть бачити зміни у завантаженні та температурі системних компонентів протягом останньої хвилини. Такий підхід дозволяє ефективно відслідковувати короткострокові коливання та оперативно реагувати на них.

Інтерактивність графіків

Графіки в програмі "MoniSy" мають високий рівень інтерактивності. Користувачі можуть наводити курсор на будь-яку точку графіку, щоб отримати детальну інформацію про конкретний момент часу. Це дозволяє краще розуміти динаміку змін та аналізувати стан системи з високою точністю.

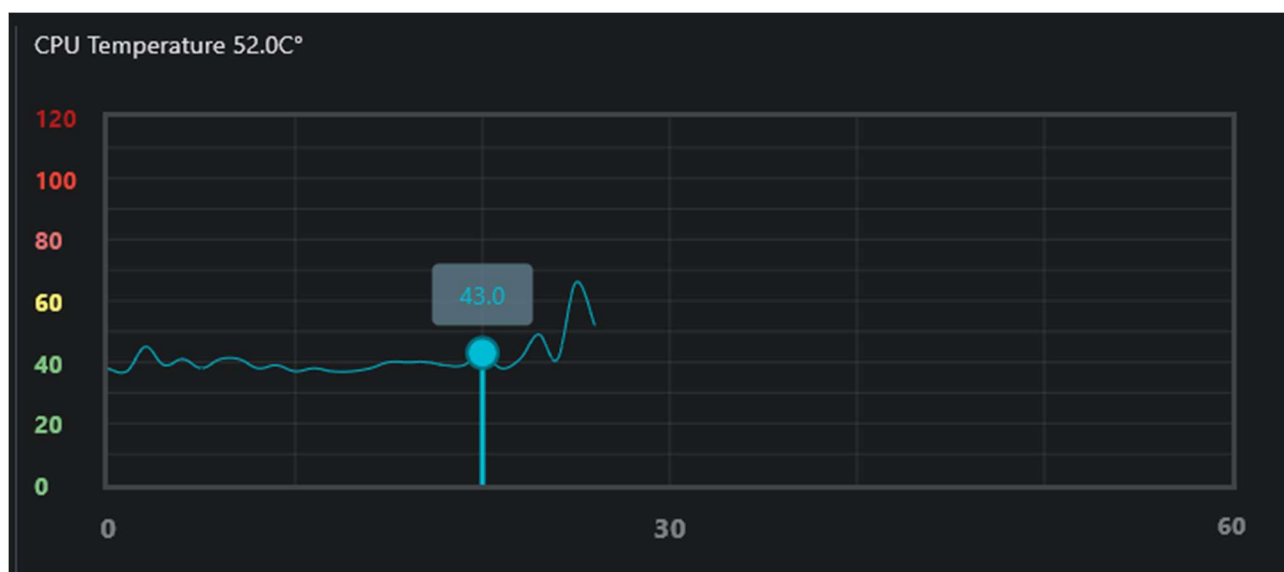


Рис. 3.1.10 – Графік температури якщо навести мишкою на конкретну область часу

Користувач має змогу отримати детальну інформацію про температуру, в той момент часу, який його цікавить. Це дозволить краще відслідковувати, та розуміти коли процесор нагрівався найбільше та найменше.

Також над двома графіками, видно актуальну температуру, що дозволяє володіти інформацією про стан процесора в даний момент часу.

Розміщення елементів інтерфейсу

Основний інтерфейс програми складається з двох ключових розділів:

1. Список з кнопками для переходу між вкладками
2. Основне вікно яке відображає вміст вкладок

Такий підхід до реалізації дизайну забезпечує користувачам максимально зручний та функціональний інтерфейс для моніторингу та аналізу стану системи, що сприяє ефективному управлінню ресурсами та підтримці стабільної роботи комп'ютера.

3.2 Високорівневий опис реалізації програмного рішення згідно розроблених вимог

Зробити реалізацію по структурній схемі

1. Архітектура програми:

- Мови програмування: Програма реалізована з використанням Python та C++ для оптимальної продуктивності та швидкодії.
- Модульна структура: Кожен аспект моніторингу та аналізу ресурсів відокремлений у власні компоненти для полегшення розширення та керування.

2. Моніторинг та аналіз ресурсів:

- Використання системних викликів та бібліотек: Для збору даних про використання CPU, пам'яті, дискового простору та мережевого трафіку у реальному часі.
- Оптимізована структура даних: Зберігання даних відбувається з урахуванням оптимальної організації для забезпечення ефективного доступу та аналізу.

3. Інтерфейс користувача:

- Графічний інтерфейс: Розроблений з використанням Python Flet для швидкого та сучасного інтерфейсу на базі фреймворку Flutter
- Налаштування інтерфейсу: Користувач може налаштовувати відображення показників.

4. Системні дані та журнали:

- Використання системних API Windows: Для отримання доступу до системних подій та журналів, що дозволяє програмі відслідковувати системні зміни та помилки.

5. Підтримка та поширення:

- Підтримка операційної системи Windows: Програма підтримує версії Windows починаючи з Windows 7 для обох архітектур, 32-х та 64-бітних.

3.3 Реалізація діаграми класів та функцій моніторингу, їх призначення і взаємодія

1. Клас SystemMonitor

Призначення: Клас, що відповідає за загальний моніторинг системи через консоль, координуючи збір інформації від інших класів.

Функції:

- `start_monitoring()`: Починає моніторинг системи, викликаючи `update_system_info()` в циклі з інтервалом в 1 секунду.
- `stop_monitoring()`: Зупиняє моніторинг системи.
- `update_system_info()`: Оновлює інформацію про систему, збираючи дані про використання CPU, пам'яті, диска та швидкість вентиляторів, і відображає їх у консолі.

Взаємодія: Використовує методи з класів `CPUUsageMonitor`, `MemoryUsageMonitor`, `DiskSpaceMonitor`, `NetworkTrafficMonitor`, `Temperature`, `SystemInformation`.

2. Клас CPUUsageMonitor

Призначення: Збір та надання інформації про стан центрального процесора (CPU).

Функції:

- `get_cpu_count()`: Повертає кількість логічних процесорів.
- `get_cpu_frequency()`: Повертає поточну частоту процесора.
- `get_cpu_utilization()`: Повертає поточний рівень використання процесора.

Взаємодія: Використовується класом `SystemMonitor` для отримання даних про CPU.

3. Клас MemoryUsageMonitor

Призначення: Збір та надання інформації про використання оперативної пам'яті (RAM).

Функції:

- `get_available_memory()`: Повертає обсяг доступної пам'яті.
- `get_total_memory()`: Повертає загальний обсяг пам'яті.
- `get_used_memory()`: Повертає обсяг використаної пам'яті.
- `get_memory_usage_percentage()`: Повертає відсоток використаної пам'яті.
- **Взаємодія:** Використовується класом `SystemMonitor` для отримання даних про пам'ять.

4. Клас `DiskSpaceMonitor`

Призначення: Збір та надання інформації про використання дискового простору.

Функції:

- `get_all_disks()`: Повертає список всіх дискових розділів.
- `get_disk_usage(disk)`: Повертає інформацію про використання дискового простору для заданого розділу.
- `get_all_disk_usage()`: Повертає інформацію про використання дискового простору для всіх розділів.

Взаємодія: Використовується класом `SystemMonitor` для отримання даних про дисковий простір.

5. Клас `NetworkTrafficMonitor`

Призначення: Збір та надання інформації про мережевий трафік.

Функції:

- `get_network_usage()`: Повертає інформацію про мережевий трафік для всіх мережевих інтерфейсів.

Взаємодія: Використовується класом `SystemMonitor` для отримання даних про мережеву активність.

6. Клас `Temperature`

Призначення: Збір та надання інформації про температуру компонентів системи.

Функції:

- `get_cpu_temperature()`: Повертає температуру центрального процесора.
- `get_gpu_temperature()`: Повертає температуру графічного процесора.

Взаємодія: Використовується класом `SystemMonitor` для отримання даних про температуру.

7. Клас `SystemInformation`

Призначення: Збір та надання базової інформації про систему та її компоненти.

Функції:

- `get_basic_system_info()`: Повертає базову інформацію про систему, включаючи платформу, архітектуру, процесор, оперативну пам'ять та графічні процесори.
- `get_event_logs(total_logs, logtype)`: Повертає журнали подій системи для заданого типу логів.

Взаємодія: Використовується класом `SystemMonitor` для отримання базової інформації про систему та журнали подій.

8. Клас `Processes`

Призначення: Збір та надання інформації про запущені процеси.

Функції:

- `get_processes()`: Повертає список запущених процесів, включаючи їх ідентифікатори, назви, використання CPU та пам'яті.

Взаємодія: Використовується класом `SystemMonitor` для отримання даних про активні процеси.

3.4 Реалізація функцій інтерфейсу для взаємодії з користувачами**Клас `SystemMonitor`**

Призначення: Клас для моніторингу системи через консоль.

Функції:

- `start_monitoring()`: Починає моніторинг системи.
- `stop_monitoring()`: Зупиняє моніторинг системи.
- `update_system_info()`: Оновлює інформацію про систему.

`MemoryUsageMonitor`

Призначення: Клас для моніторингу використання пам'яті.

Функції:

- `get_memory_usage_percentage()`: Повертає відсоток використаної пам'яті.
- `get_used_memory()`: Повертає кількість використаної пам'яті в байтах.
- `get_total_memory()`: Повертає загальну кількість пам'яті в байтах.

`CPUUsageMonitor`

Призначення: Клас для моніторингу використання процесора.

Функції:

- `get_cpu_utilization()`: Повертає відсоток використання процесора.
- `get_cpu_frequency()`: Повертає поточну частоту процесора.

DiskSpaceMonitor

Призначення: Клас для моніторингу використання дискового простору.

Функції:

- `get_all_disk_usage()`: Повертає інформацію про використання дискового простору для всіх дисків.

Temperature

Призначення: Клас для моніторингу температури процесора і графічного процесора.

Функції:

- `get_cpu_temperature()`: Повертає поточну температуру процесора.
- `get_gpu_temperature()`: Повертає поточну температуру графічного процесора.

Processes

Призначення: Клас для моніторингу запущених процесів.

Функції:

- `get_processes()`: Повертає список запущених процесів.

NetworkTrafficMonitor

Призначення: Клас для моніторингу мережевого трафіку.

Функції:

- `get_network_usage()`: Повертає інформацію про використання мережевого трафіку.

SystemInformation

Призначення: Клас для отримання загальної інформації про систему.

Функції:

- `get_basic_system_info()`: Повертає базову інформацію про систему.
- `get_event_logs(total_logs, logtype)`: Повертає журнали подій системи.

Взаємодія між класами:

1. `SystemMonitor` взаємодіє з усіма класами для збору та оновлення інформації про стан системи.
2. `MemoryUsageMonitor`, `CPUUsageMonitor`, `DiskSpaceMonitor`, `Temperature`, `Processes`, `NetworkTrafficMonitor`, `SystemInformation` надають інформацію, яка використовується `SystemMonitor` для відображення даних в інтерфейсі.

3.6 Опис алгоритму роботи графічного інтерфейсу

Алгоритм програми починається з імпорту необхідних модулів та бібліотек, таких як функції, графіки, бібліотека Flet, а також модуль time. Це підготовка для подальшого використання функцій, графічних елементів та часових інтервалів.

Основна функція програми називається `'main'` і приймає аргумент `'page'`, який є об'єктом сторінки від Flet. Всередині цієї функції створюється головний контейнер `'body'`, що представляє собою вертикальну колонку, куди будуть додаватися різні елементи інтерфейсу.

Функція `'change_chart'` відповідає за оновлення даних на графіках. Вона додає нові точки даних до графіку, якщо їх кількість менша за 60, або очищає графік, якщо кількість перевищує цей ліміт.

Функція `'render_usage'` виконує відстеження використання пам'яті та процесора. Вона запускає безкінечний цикл, який оновлює графіки використання пам'яті та процесора щосекунди або при першому запуску. Вона також створює і оновлює текстові метки з інформацією про використання RAM, CPU та дискового простору.

Функція `'render_temp'` аналогічно `'render_usage'`, але відповідає за відображення температури процесора і графічного процесора. Вона також запускає безкінечний цикл, який щосекунди оновлює графіки температури.

Функція `'render_system_info'` витягує основну системну інформацію, таку як операційна система, версія BIOS та інші, і відображає її у вигляді текстових меток у вертикальній колонці.

Функція `'render_processes'` відповідає за відображення активних процесів системи у вигляді таблиці. Вона сортує процеси за різними параметрами (наприклад, PID, ім'я, використання CPU або пам'яті) і оновлює таблицю відповідно до обраного сортування.

Функція `'render_network'` відображає інформацію про мережевий трафік. Вона збирає дані про використання мережі і відображає їх у вигляді текстових меток.

Функція `'render_logs'` відповідає за відображення системних журналів. Вона створює таблицю з інформацією про події системи (наприклад, ID події, час, джерело та повідомлення). Також доступні випадаючі меню для вибору типу журналу і кількості відображуваних записів.

Функція `'change_body'` визначає, яка саме функція повинна бути викликана при зміні вибору на навігаційній панелі. Вона оновлює вміст головного контейнера `'body'` залежно від обраного пункту меню.

Використовується навігаційна панель `'rail'`, яка дозволяє користувачеві перемикатися між різними розділами програми, такими як інформація про систему, використання ресурсів, температура, процеси, мережа та системні журнали.

Основний макет програми складається з навігаційної панелі зліва та головного контейнера праворуч. При першому запуску програми викликається функція `'render_system_info'` для відображення базової інформації про систему.

Наприкінці, якщо програма запускається безпосередньо, вона виконує виклик функції `'ft.app(target=main)'`, яка ініціалізує і запускає додаток з головною функцією `'main'`.

4. Тестування

Оскільки системні дані отримуються в режимі реального часу через графічний інтерфейс, використання стандартних Unit-тестів є недоцільним. Для тестування функціоналу програми я порівнював результати з відомими та перевіреними інструментами, такими як фірмова утиліта «Gigabyte Control Center 3.0», «Налаштування» та «Диспетчер задач» операційної системи. Також можна провести тестування вимог щодо оновлення графіку за заданий час.

1. Порівняння даних про систему та пристрій



Рис. 4.1 – Специфікації системи у робочому проєкті

Специфікації пристрою в налаштуваннях:

Device name Yurii
 Processor 12th Gen Intel(R) Core(TM) i5-12500H 2.50 GHz
 Installed RAM 16.0 GB (15.7 GB usable)
 Device ID 41A9F811-FF45-4C1E-9622-8DFBA282F328
 Product ID 00330-80000-00000-AA704
 System type 64-bit operating system, x64-based processor
 Pen and touch No pen or touch input is available for this display

Специфікації Windows в налаштуваннях:

Edition Windows 11 Pro
 Version 23H2
 Installed on 2024-02-22
 OS build 22631.3593
 Experience Windows Feature Experience Pack 1000.22700.1003.0

2. Порівняння інформації про використання ресурсів

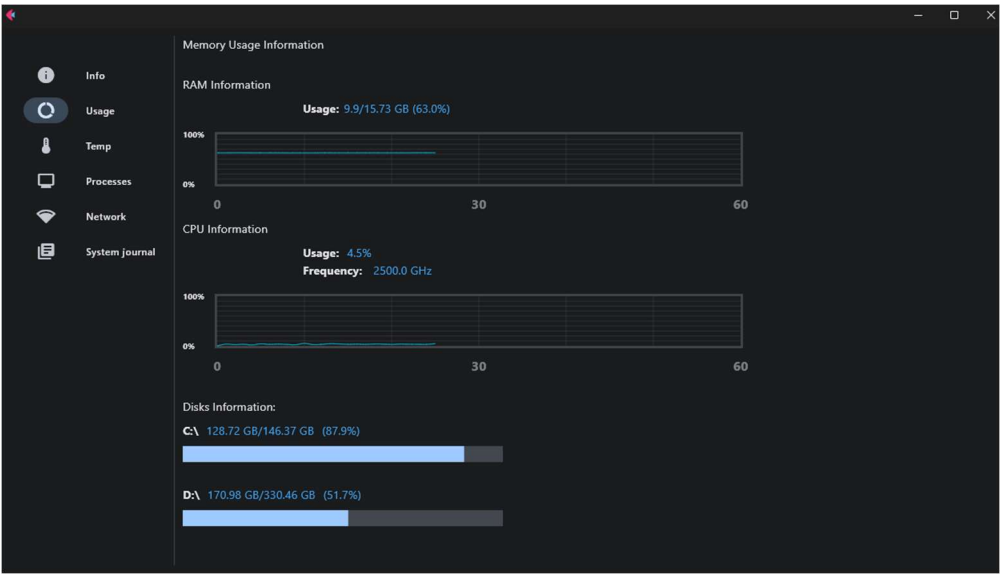


Рис. 4.2 – Використання ресурсів системи у робочому проєкті

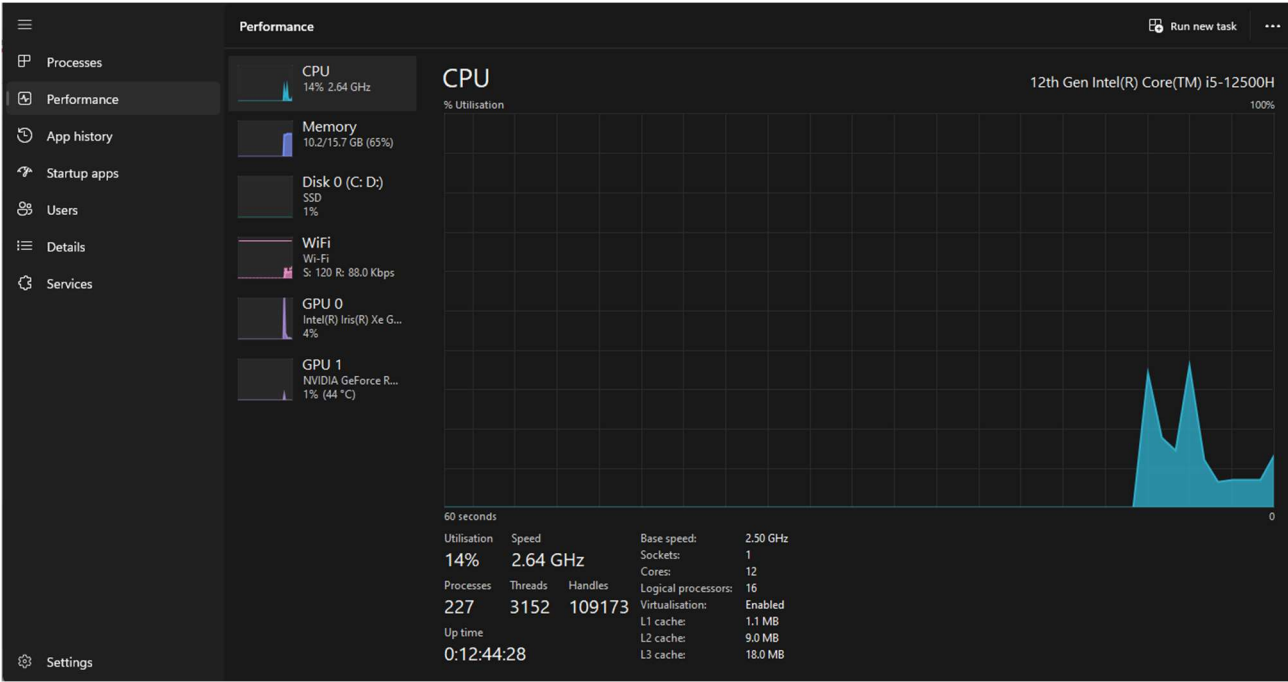


Рис. 4.3 – Використання ресурсів системи у «Диспетчері задач»

3. Температура процесора та відеокарти



Рис. 4.4 – Температури системи у робочому проєкті



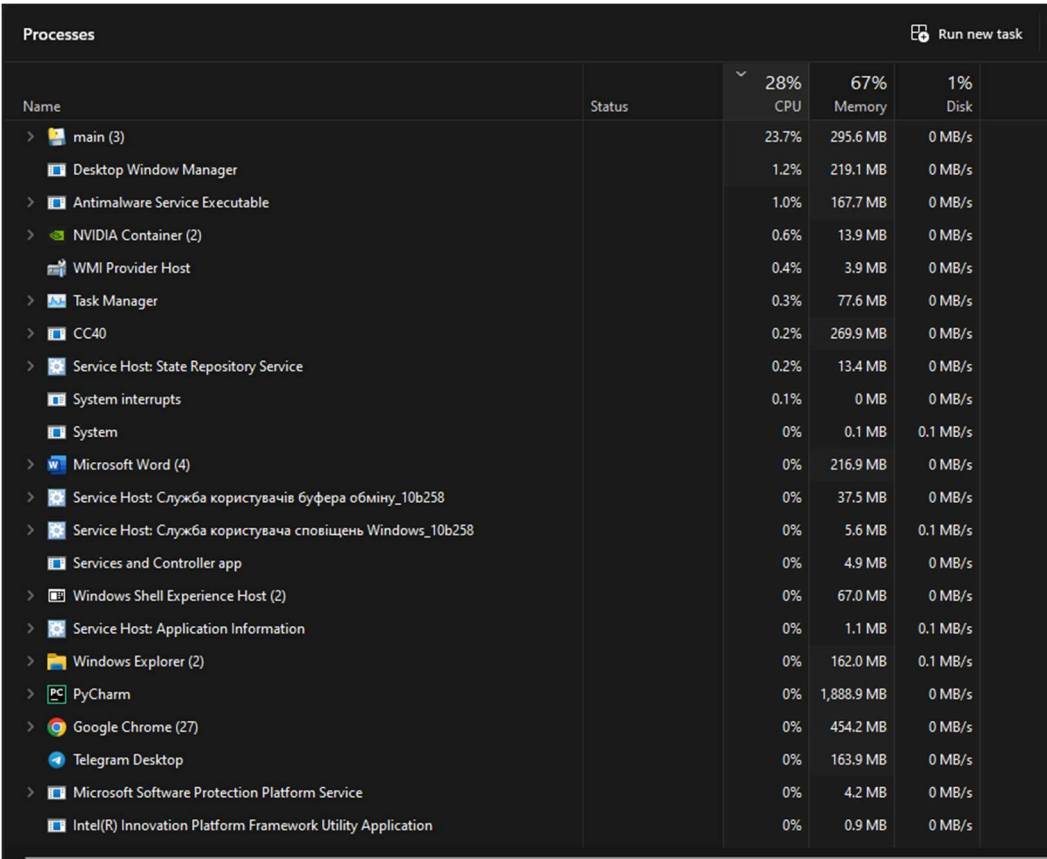
Рис. 4.5 – Температури системи у фірмовій утиліті ноутбуку

4. Активні процеси



pid	name	cpu_percent↓	memory
0	System Idle Process	1450.0	0.0078125
6192	flet.exe	62.5	364.625
1756	main.exe	37.5	150.0546875
4	System	12.5	0.140625
1772	dwm.exe	12.5	276.96875
3784	svchost.exe	12.5	22.19140625
172		0.0	0.1015625
204	Registry	0.0	64.0
704	WINWORD.EXE	0.0	426.16015625
780	smss.exe	0.0	1.06640625
908	svchost.exe	0.0	9.88671875
----	----	----	-----

Рис. 4.6 – Активні процеси у робочому проєкті



Name	Status	28% CPU	67% Memory	1% Disk
main (3)		23.7%	295.6 MB	0 MB/s
Desktop Window Manager		1.2%	219.1 MB	0 MB/s
Antimalware Service Executable		1.0%	167.7 MB	0 MB/s
NVIDIA Container (2)		0.6%	13.9 MB	0 MB/s
WMI Provider Host		0.4%	3.9 MB	0 MB/s
Task Manager		0.3%	77.6 MB	0 MB/s
CC40		0.2%	269.9 MB	0 MB/s
Service Host: State Repository Service		0.2%	13.4 MB	0 MB/s
System interrupts		0.1%	0 MB	0 MB/s
System		0%	0.1 MB	0.1 MB/s
Microsoft Word (4)		0%	216.9 MB	0 MB/s
Service Host: Служба користувачів буфера обміну_10b258		0%	37.5 MB	0 MB/s
Service Host: Служба користувача сповіщень Windows_10b258		0%	5.6 MB	0.1 MB/s
Services and Controller app		0%	4.9 MB	0 MB/s
Windows Shell Experience Host (2)		0%	67.0 MB	0 MB/s
Service Host: Application Information		0%	1.1 MB	0.1 MB/s
Windows Explorer (2)		0%	162.0 MB	0.1 MB/s
PyCharm		0%	1,888.9 MB	0 MB/s
Google Chrome (27)		0%	454.2 MB	0 MB/s
Telegram Desktop		0%	163.9 MB	0 MB/s
Microsoft Software Protection Platform Service		0%	4.2 MB	0 MB/s
Intel(R) Innovation Platform Framework Utility Application		0%	0.9 MB	0 MB/s

Рис. 4.7 – Активні процеси у «Диспетчері задач»

5. Інформація про мережу

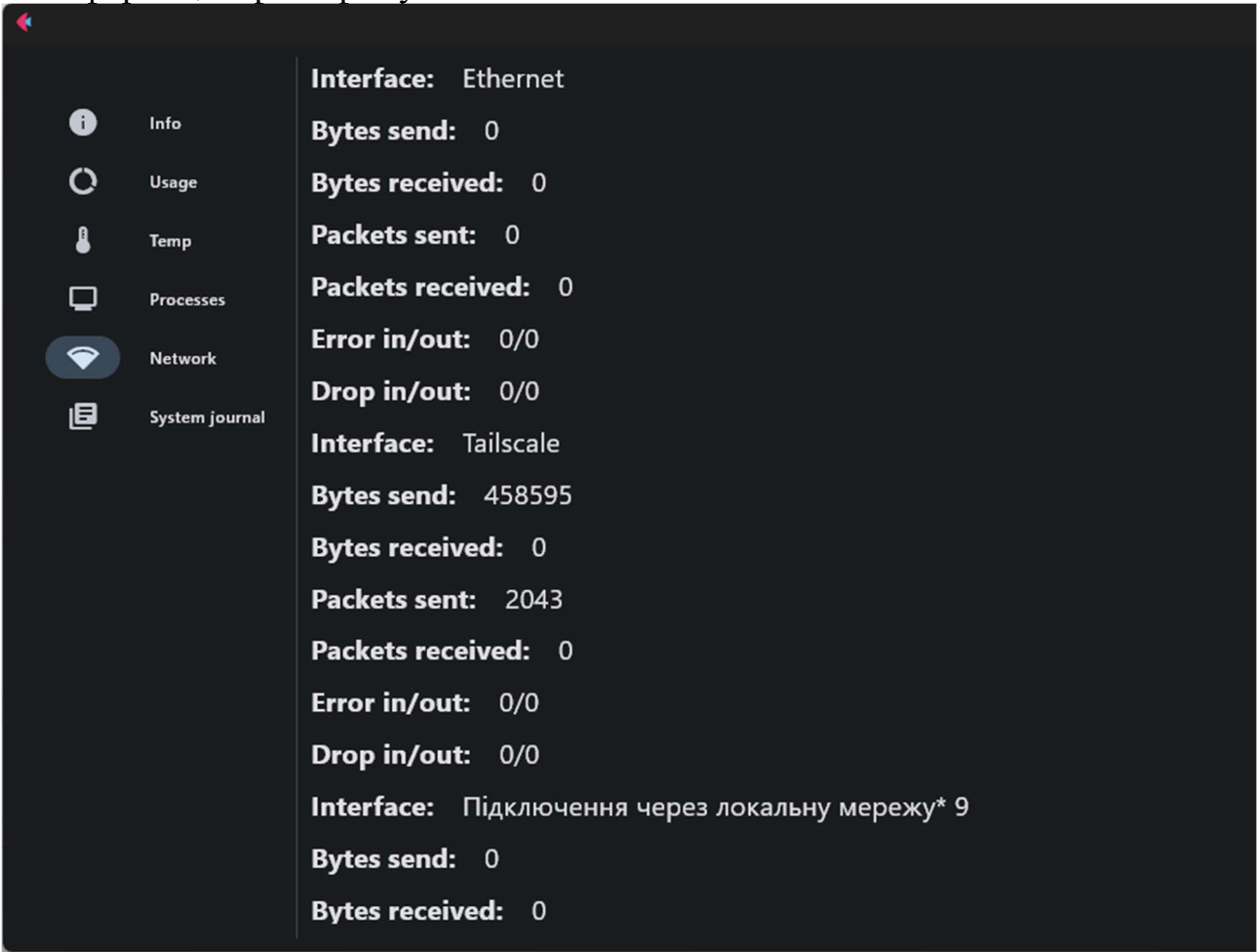


Рис. 4.8 – Інформація про мережу у робочому проєкті

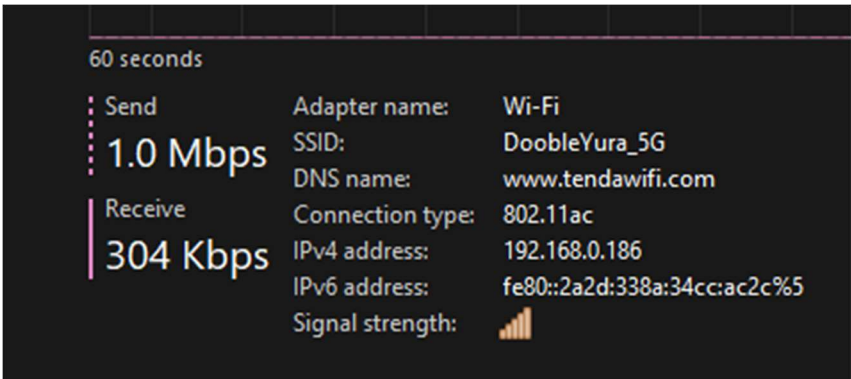
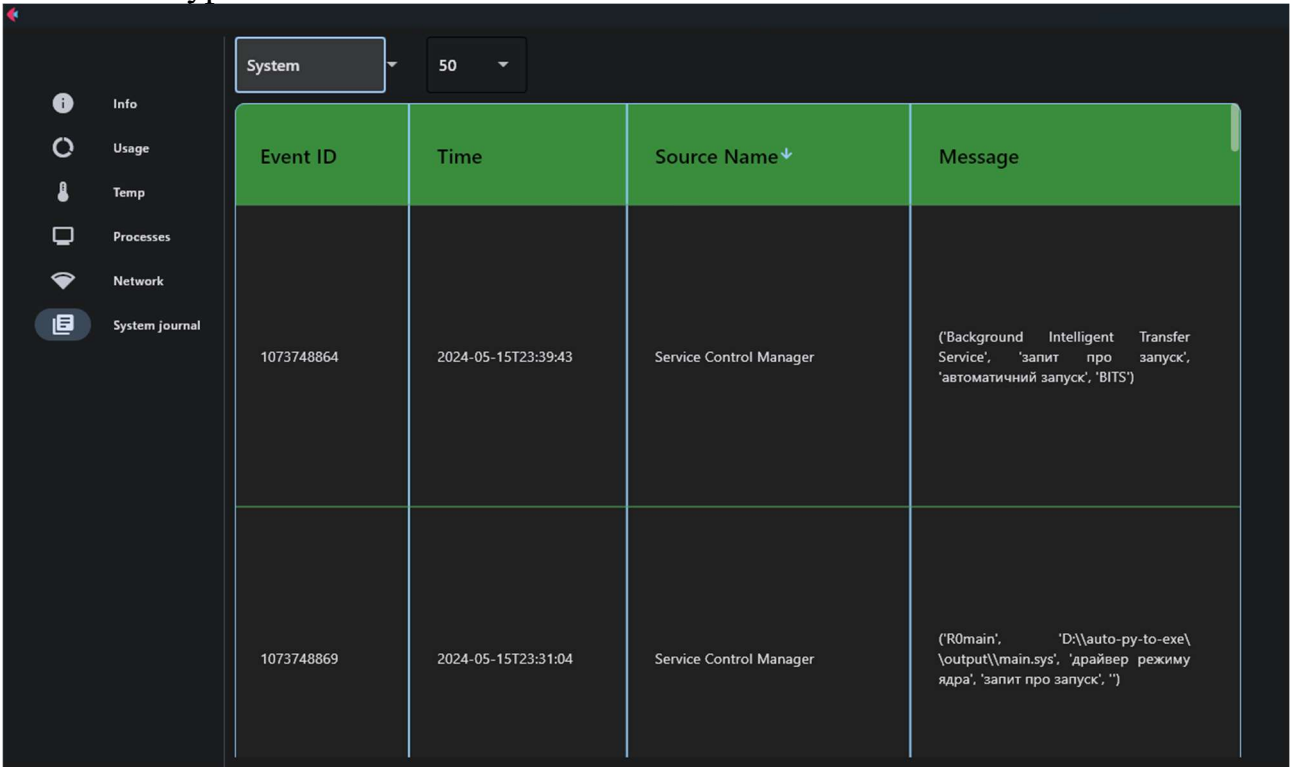


Рис. 4.9 – Інформація про мережу в «Диспетчері задач»

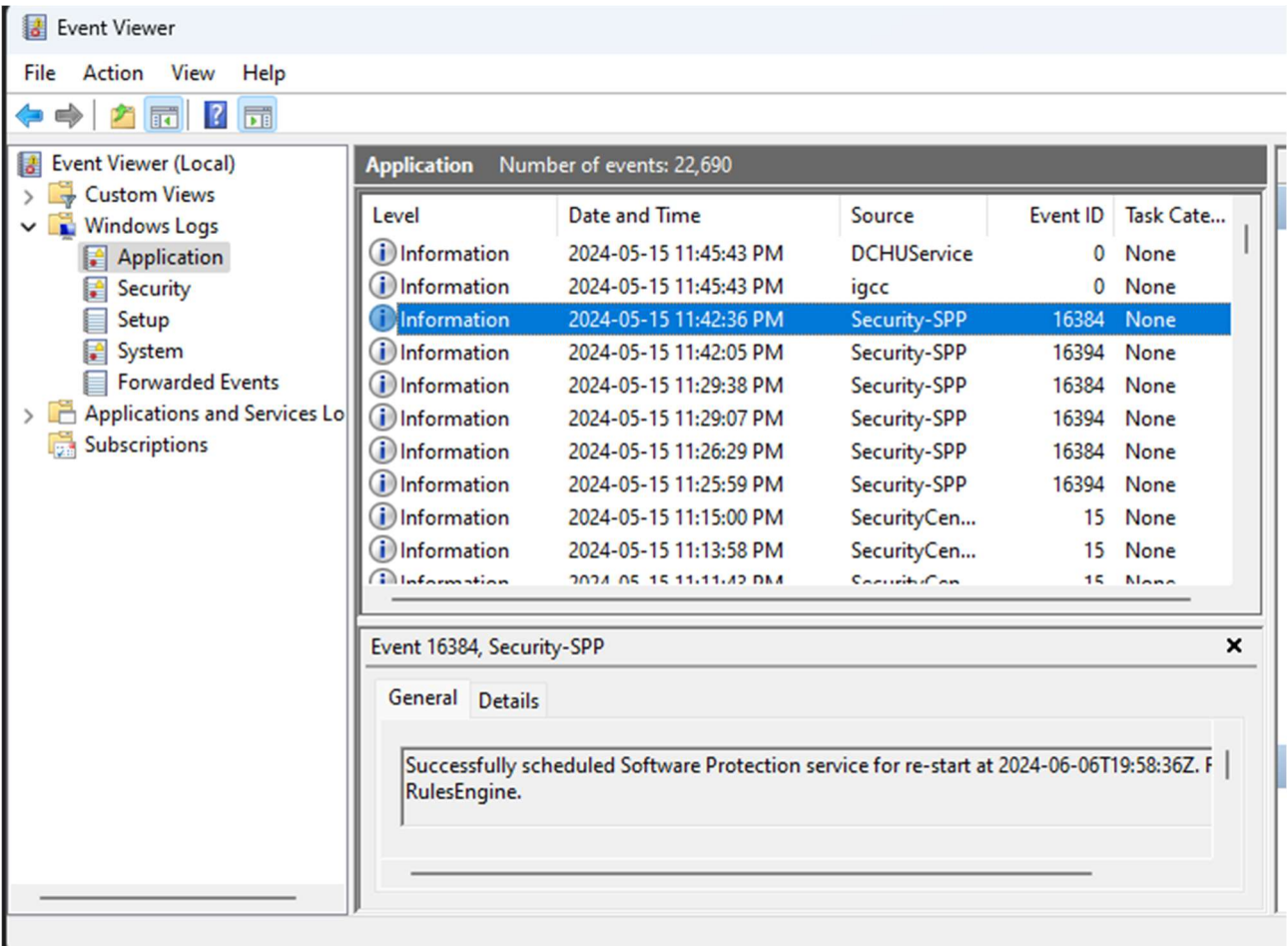
6. Системні журнали



The screenshot shows a system journal interface with a sidebar on the left containing icons for Info, Usage, Temp, Processes, Network, and System journal. The main area displays a table of events. At the top, there are filters for 'System' and '50'.

Event ID	Time	Source Name↓	Message
1073748864	2024-05-15T23:39:43	Service Control Manager	('Background Intelligent Transfer Service', 'запит про запуск', 'автоматичний запуск', 'BITS')
1073748869	2024-05-15T23:31:04	Service Control Manager	('R0main', 'D:\auto-py-to-exe\output\main.sys', 'драйвер режиму ядра', 'запит про запуск', '')

Рис 4.10 – Інформація про системні записи у робочому проєкті



The screenshot shows the Windows Event Viewer application. The left pane shows the tree view with 'Event Viewer (Local)' expanded, and 'Windows Logs' > 'Application' selected. The right pane shows a list of events with columns: Level, Date and Time, Source, Event ID, and Task Category. Event 16384 from Security-SPP is selected. Below the list, the details for event 16384 are shown, including the message: 'Successfully scheduled Software Protection service for re-start at 2024-06-06T19:58:36Z. RulesEngine.'

Level	Date and Time	Source	Event ID	Task Category
Information	2024-05-15 11:45:43 PM	DCHUService	0	None
Information	2024-05-15 11:45:43 PM	igcc	0	None
Information	2024-05-15 11:42:36 PM	Security-SPP	16384	None
Information	2024-05-15 11:42:05 PM	Security-SPP	16394	None
Information	2024-05-15 11:29:38 PM	Security-SPP	16384	None
Information	2024-05-15 11:29:07 PM	Security-SPP	16394	None
Information	2024-05-15 11:26:29 PM	Security-SPP	16384	None
Information	2024-05-15 11:25:59 PM	Security-SPP	16394	None
Information	2024-05-15 11:15:00 PM	SecurityCen...	15	None
Information	2024-05-15 11:13:58 PM	SecurityCen...	15	None
Information	2024-05-15 11:11:42 PM	SecurityCen...	15	None

Event 16384, Security-SPP

General Details

Successfully scheduled Software Protection service for re-start at 2024-06-06T19:58:36Z. RulesEngine.

Рис 4.11 – Інформація про системні журнали у Event Viewer

```
"D:\LPNU\SEMESTR 6\SPZ\KURSOVA\venv\Scripts
page update time: 0.013273000717163086
page update time: 1.0113024711608887
page update time: 1.009077548980713
page update time: 1.011054515838623
page update time: 1.0098967552185059
page update time: 1.0094492435455322
page update time: 1.0094530582427979
page update time: 1.010277271270752
page update time: 1.0092825889587402
page update time: 1.010822057723999
page update time: 1.010300874710083

Process finished with exit code 0
```

Рис 4.12 – Інформація про системні журнали у Event Viewer

Для проведення тестування на оновлення графіку, я встановив таймери на початку та кінці функції оновлення сторінки «Використання». Щоб відмалювати сторінку, моєму комп'ютеру потрібно приблизно 0.01327 секунди, що дорівнює приблизно 75 FPS. Якщо брати час оновлення графіка разом з часом на оновлення всієї сторінки, то похибка в найгіршому випадку становить +0.013 секунди, що є цілком природнім, та впливає на сприйняття користувачем.

Висновки

Після проведення порівняння значень, отриманих програмою "MoniSy", з іншими відомими утилітами та системними засобами, можна з упевненістю сказати, що дана програма відображає актуальні значення та записи в комп'ютері. Це забезпечує користувачам зручний доступ до детальної інформації про стан їхньої системи в реальному часі.

Програма "MoniSy" підтвердила свою надійність та точність у відображенні даних, що робить її корисним інструментом для моніторингу та аналізу системних ресурсів як для досвідчених користувачів, так і для системних адміністраторів.

5. Висновки

В цьому курсовому проєкті було розроблено програму для моніторингу системи через консоль та графічний інтерфейс на основі мови програмування Python. Програма надає користувачеві можливість отримати різноманітну інформацію про систему, таку як використання пам'яті, температура процесора, інформація про процеси, мережевий трафік та журнали подій Windows.

У результаті розробки програми було досягнуто наступних цілей:

1. Реалізовано функціонал для моніторингу та відстеження використання системних ресурсів.
2. Забезпечено доступ до інформації про процеси, які запущені на комп'ютері, включаючи їх використання пам'яті та CPU.
3. Надано можливість аналізу журналів подій Windows для виявлення та вирішення проблем в системі.
4. Реалізовано зручний інтерфейс користувача для отримання необхідної інформації.

Отже, цей проєкт є важливим інструментом для моніторингу та аналізу роботи комп'ютерних систем, що може бути корисним як для звичайних користувачів, так і для адміністраторів мережі та системних адміністраторів. Додатково, програма має потенціал для розширення функціональності та вдосконалення шляхом додавання нових можливостей та оптимізації інтерфейсу користувача.

Список використаних джерел

1. Garg, & Verma, G. (2017). Operating Systems: An Introduction. Mercury Learning & Information.
2. Panek. (2020). Windows operating system fundamentals (1st edition). Sybex. – 419 p.
3. David A. Solomon (2017). “Windows Internals”.
4. Габрусев В.Ю. (2007). Основи операційних систем: ядро, процес, потік. – 96с.
5. Бондаренко М.Ф. Операційні системи: навч. посібник / М.Ф. Бондаренко, О.Г. Качко. – Харків: Компанія СМІТ, 2008. – 432 с.
6. Johnson M. Hart. Windows System Programming, 4th edition / Hart Johnson. – Addison-Wesley, 2010. – 656 p.
7. Thomas W. Doeppner. Operating Systems In Depth: Design and Programming / W. Thomas. – John Wiley & Sons, 2010. – 462 p.

Додатки

Додаток 1 – Оцінка виконання програми за допомогою User Story

User Story 1: Моніторинг використання CPU

Оцінка часу виконання: 4 дні

Я, як користувач, хочу мати можливість переглядати використання CPU моєї системи, щоб слідкувати за її продуктивністю та виявляти можливі проблеми.

Користь для користувача:

Забезпечення можливості моніторингу та аналізу використання CPU для забезпечення оптимальної продуктивності та вчасного виявлення можливих проблем.

Критерії готовності:

Функції для збору інформації про використання CPU реалізовані.

Створений інтерфейс для відображення цієї інформації.

Пройдено тестування та відлагодження коду.

Оновлена документація з описом нового функціоналу.

User Story 2: Моніторинг використання пам'яті

Оцінка часу виконання: 4 дні

Я, як користувач, хочу бачити статистику використання оперативної та віртуальної пам'яті моєї системи, щоб контролювати ресурси та виявляти проблеми.

Користь для користувача:

Надання можливості контролювати використання оперативної та віртуальної пам'яті для оптимізації ресурсів та вчасного виявлення проблем.

Критерії готовності:

Реалізовані функції для збору інформації про використання пам'яті.

Інтерфейс для відображення цих даних створений.

Проведено тестування та відлагодження коду.

Оновлена документація з описом нового функціоналу.

User Story 3: Моніторинг дискового простору

Оцінка часу виконання: 2 дні

Я, як користувач, хочу бачити інформацію про залишковий дисковий простір на моїй системі, щоб контролювати використання місця та уникати проблем зі зберіганням.

Користь для користувача:

Забезпечення можливості моніторингу залишкового дискового простору для попередження проблем зі зберіганням та оптимізації використання місця.

Критерії готовності:

Розроблені функції для отримання даних про диск.

Інтерфейс для відображення цих даних реалізований.

Проведено тестування та відлагодження коду.

Оновлена документація з описом нового функціоналу.

User Story 4: Моніторинг мережевого трафіку

Оцінка часу виконання: 5 днів

Я, як користувач, хочу бачити швидкість передачі та отримання даних через мережу, щоб контролювати мережеві ресурси та виявляти проблеми зі зв'язком.

Користь для користувача:

Надання можливості моніторингу мережевого трафіку для контролю ресурсів та вчасного виявлення проблем зі зв'язком.

Критерії готовності:

Написані функції для збору інформації про мережевий трафік.

Інтерфейс для відображення цих даних створений.

Проведено тестування та відлагодження коду.

Оновлена документація з описом нового функціоналу.

User Story 5: Перевірка стану системи в реальному часі

Оцінка часу виконання: 5 днів

Я, як користувач, хочу бачити списки поточних процесів та служб з їхньою активністю та моніторити використання системних ресурсів, щоб виявляти проблеми та аномалії.

Користь для користувача:

Забезпечення можливості моніторингу активності процесів та служб для вчасного виявлення проблем та аномалій у використанні системних ресурсів.

Критерії готовності:

Реалізовані функції для отримання інформації про поточні процеси та служби.

Інтерфейс для відображення цих даних розроблений.

Проведено тестування та відлагодження коду.

Оновлена документація з описом нового функціоналу.

User Story 6: Моніторинг системних подій та журналів

Оцінка часу виконання: 2 дні

Я, як користувач, хочу бачити системні події та журнали для виявлення можливих проблем та аномалій у роботі системи.

Користь для користувача:

Надання можливості моніторингу системних подій та журналів для вчасного виявлення проблем та аномалій у роботі системи.

Критерії готовності:

Функції для отримання інформації про системні події та журнали реалізовані.
Створений інтерфейс для відображення цих даних.
Пройдено тестування та відлагодження коду.
Оновлена документація з описом нового функціоналу.

User Story 7: Моніторинг температури та вентиляції

Оцінка часу виконання: 6 днів

Я, як користувач, хочу бачити інформацію про температуру компонентів та швидкість обертання вентиляторів для контролю над температурним режимом моєї системи.

Користь для користувача:

Забезпечення можливості моніторингу температури та швидкості обертання вентиляторів для попередження перегріву та забезпечення оптимального температурного режиму системи.

Критерії готовності:

Функції для отримання інформації про температуру та швидкість обертання вентиляторів реалізовані.
Інтерфейс для відображення цих даних створений.
Пройдено тестування та відлагодження коду.
Оновлена документація з описом нового функціоналу.

User Story 8: Системні інформаційні дані

Оцінка часу виконання: 4 дні

Я, як користувач, хочу бачити основну інформацію про мою систему, таку як операційна система, версія ядра, тип процесора, обсяг оперативної та віртуальної пам'яті.

Користь для користувача:

Надання можливості перегляду основної інформації про систему для швидкого доступу до важливих даних про конфігурацію системи.

Критерії готовності:

Функції для отримання системної інформації реалізовані.

Створений інтерфейс для відображення цих даних.

Проведено тестування та відлагодження коду.

Оновлена документація з описом нового функціоналу

User Story 9: Налаштування сповіщень**Оцінка часу виконання: 4 дні**

Я, як користувач, хочу мати можливість налаштувати сповіщення для різних параметрів системи, щоб оперативно реагувати на проблеми та аномалії.

Користь для користувача:

Забезпечення можливості налаштування сповіщень для оперативного реагування на проблеми та аномалії в роботі системи.

Критерії готовності:

Реалізовані функції для налаштування сповіщень.

Додана можливість встановлення порогових значень для різних параметрів системи.

Пройдено тестування та відлагодження коду.

Оновлена документація з описом нового функціоналу.

User Story 10: Ефективність роботи**Оцінка часу виконання: 7 днів**

Я, як користувач, хочу бачити зведену інформацію про ефективність роботи моєї системи, щоб отримувати рекомендації щодо оптимізації використання ресурсів та підвищення її продуктивності.

Користь для користувача:

Надання можливості перегляду зведеної інформації про ефективність роботи системи та отримання рекомендацій для підвищення продуктивності та оптимізації ресурсів.

Критерії готовності:

Написані функції для збору і аналізу інформації про використання ресурсів системи.

Розроблений інтерфейс для відображення зведеної інформації та надання рекомендацій.

Пройдено тестування та відлагодження коду.

Оновлена документація з описом нового функціоналу.

Додаток 2 – Експорт програми в ехе

Для зручності користувачів, проект "MoniSy" було експортовано у виконавчий файл формату .exe за допомогою утиліти **py2exe**. Це дозволяє запускати програму без необхідності попереднього встановлення Python та всіх необхідних бібліотек.

Переваги використання .exe файлу

1. **Простота розповсюдження:** Кінцевий користувач отримує готовий до запуску файл, що спрощує процес інсталяції та використання програми.
2. **Інтеграція всіх залежностей:** Всі бібліотеки та модулі, необхідні для роботи програми, включені всередині .exe файлу. Це забезпечує стабільну роботу програми на будь-якій системі Windows без додаткових налаштувань.

Доступ до захищених ресурсів

Для отримання доступу до температури процесора та інших захищених файлів, режим запуску програми за замовчуванням встановлено як «Запуск від імені адміністратора». Це дозволяє програмі отримувати всі необхідні привілеї для коректної роботи з системними API та іншими критичними ресурсами.

Процес експорту

Процес експорту включає такі основні кроки:

1. **Підготовка скрипту для експорту:** Написання та налаштування конфігураційного файлу для py2exe, що визначає, які файли та залежності необхідно включити у кінцевий .exe файл.
2. **Виконання утиліти py2exe:** Запуск утиліти для створення виконавчого файлу на основі підготовленого скрипту.
3. **Тестування виконавчого файлу:** Перевірка роботи створеного .exe файлу на різних системах Windows для забезпечення стабільності та сумісності.

Завдяки цим крокам, кінцевий продукт є зручним та надійним для користувачів, забезпечуючи легкий доступ до всіх функціональних можливостей програми "MoniSy".

Додаток 3 – Вихідний код програми

Main.py

```

from functions import *
from charts import *
import flet as ft
import time

last_page_index = 0

def main(page: ft.Page):
    body = ft.Column([], alignment=ft.MainAxisAlignment.START, expand=True)

    def change_chart(chart, value):
        if len(chart.data_series[0].data_points) < 60:
            chart.data_series[0].data_points.append(ft.LineChartDataPoint(
                len(chart.data_series[0].data_points), value), )
        else:
            chart.data_series[0].data_points = []

    def render_usage(e):
        timer = time.time()
        first = True
        destination_label = e.control.destinations[e.control.selected_index].label
        ram_progress = get_usage_chart()
        cpu_progress = get_usage_chart()

        while True:
            if destination_label != e.control.destinations[e.control.selected_index].label:
                break
            if e.control.destinations[e.control.selected_index].label == "Usage":
                if time.time() - timer > 1 or first:
                    first = False
                    ram_usage = MemoryUsageMonitor.get_memory_usage_percentage() #
75%
                    cpu_usage = CPUUsageMonitor().get_cpu_utilization()
                    change_chart(ram_progress, ram_usage)
                    change_chart(cpu_progress, cpu_usage)

                    used_memory = round(MemoryUsageMonitor.get_used_memory() /
1024 ** 3, 2)
                    total_memory = round(MemoryUsageMonitor.get_total_memory() /
1024 ** 3, 2)

```

```

        ram_label = ft.Container(
            content=ft.Row([
                ft.Text(f"Usage: ",
style=ft.TextStyle(weight=ft.FontWeight.BOLD)),
                ft.Text(
                    f"{used_memory}/"
                    f"{total_memory} GB"
                    f"
({MemoryUsageMonitor.get_memory_usage_percentage()}%)"
                    ,
                    color=ft.colors.BLUE_400,
                )
            ],
            spacing=2,
        ),
        margin=ft.Margin(left=150, right=0, top=0, bottom=10),
    )
    cpu_label = ft.Container(
        content=ft.Column([
            ft.Row([
                ft.Text(f"Usage:",
style=ft.TextStyle(weight=ft.FontWeight.BOLD)),
                ft.Text(
                    f"{cpu_usage}%",
                    color=ft.colors.BLUE_400,
                )
            ],
            ft.Row([
                ft.Text("Frequency: ",
style=ft.TextStyle(weight=ft.FontWeight.BOLD)),
                ft.Text(
                    f"{CPUUsageMonitor().get_cpu_frequency().current} GHz",
                    color=ft.colors.BLUE_400,
                ),
            ])
        ],
        spacing=2,
    ),
    margin=ft.Margin(left=150, right=0, top=0, bottom=10),
)

```

```

disks = []
disks_information = DiskSpaceMonitor().get_all_disk_usage()
for index, disk in enumerate(disks_information.keys()):
    disks.append(
        ft.Container(content=ft.Column([
            ft.Row(
                [
                    ft.Text(f'{disk}',
style=ft.TextStyle(weight=ft.FontWeight.BOLD)),
                    ft.Text(f'{round(disks_information[disk]['used'] / 1024 ** 3,
2)} GB"
                    f'/{round(disks_information[disk]['total'] / 1024 ** 3,
2)} GB"',
                    color=ft.colors.BLUE_400),
                    ft.Text(f'({round(disks_information[disk]['percent'],
2)}%)",
                    color=ft.colors.BLUE_400)
                ]
            ),
            ft.ProgressBar(value=disks_information[disk]['percent'] / 100,
width=400, height=20),
        ]
    ),
    margin=ft.Margin(left=0, right=0, top=0, bottom=20),
),
)

body.controls = [
    ft.Text("Memory Usage Information\n"),
    ft.Text("RAM Information"),
    ram_label,
    ram_progress,
    ft.Text("CPU Information"),
    cpu_label,
    cpu_progress,
    ft.Text("\nDisks Information:"),

    ] + disks
timer = time.time()
page.update()

```

else:

```

        break

def render_temp(e):
    timer = time.time()
    first = True
    destination_label = e.control.destinations[e.control.selected_index].label

    cpu_temperature_chart = get_temp_chart()
    gpu_temperature_chart = get_temp_chart()

    while True:
        if destination_label != e.control.destinations[e.control.selected_index].label:
            break
        if e.control.destinations[e.control.selected_index].label == "Temp":
            if time.time() - timer > 1 or first:
                first = False

                cpu_temperature = Temperature().get_cpu_temperature()
                if cpu_temperature:
                    cpu_temperature = cpu_temperature
                else:
                    cpu_temperature = 0

                gpu_temperature = Temperature().get_gpu_temperature()
                if gpu_temperature:
                    gpu_temperature = gpu_temperature
                else:
                    gpu_temperature = 0

                change_chart(cpu_temperature_chart, cpu_temperature)
                change_chart(gpu_temperature_chart, gpu_temperature)

                body.controls = [
                    ft.Text(f"CPU Temperature {cpu_temperature}C°\n"),
                    cpu_temperature_chart,
                    ft.Text(f"GPU Temperature {gpu_temperature}C°\n"),
                    gpu_temperature_chart,
                ]
                page.update()
                timer = time.time()
            else:
                break

def render_system_info(e=None):
    information = SystemInformation().get_basic_system_info()

```

```

information_labels = ft.Columnn()

for key, value in information.items():
    information_labels.controls.append(
        ft.Row(
            [
                ft.Text(f" {key}:", style=ft.TextStyle(weight=ft.FontWeight.BOLD),
size=20),
                ft.Text(f" {value}", size=20),
            ]
        )
    )
body.controls = [
    information_labels,
]
page.update()

def render_processes(e):
    destination_label = e.control.destinations[e.control.selected_index].label

    processes_table = ft.DataTable(
        width=700,
        bgcolor=ft.colors.GREEN_600,
        border=ft.border.all(1, ft.colors.BLUE_200),
        border_radius=10,
        vertical_lines=ft.border.BorderSide(2, ft.colors.BLUE_200),
        horizontal_lines=ft.border.BorderSide(1, "green"),
        sort_column_index=2,
        heading_row_color=ft.colors.BLACK12,
        heading_row_height=100,
        divider_thickness=0,
        data_row_color=ft.colors.GREY_900,
        columns=[
            ft.DataColumn(
                ft.Text("pid", color=ft.colors.BLACK, size=20),
                on_sort=lambda e: change_sort_index(e.column_index),
            ),
            ft.DataColumn(
                ft.Text("name", color=ft.colors.BLACK, size=20),
                on_sort=lambda e: change_sort_index(e.column_index),
            ),
            ft.DataColumn(
                ft.Text("cpu_percent", color=ft.colors.BLACK, size=20),
                on_sort=lambda e: change_sort_index(e.column_index),
            ),

```



```

        ft.DataColumn(
            ft.Text("memory", color=ft.colors.BLACK, size=20),
            on_sort=lambda e: change_sort_index(e.column_index),
        ),
    ],
)

def change_sort_index(index):
    processes_table.sort_column_index = index
    processes_table.sort_ascending = not processes_table.sort_ascending

body.controls = [
    ft.Column([processes_table], scroll=ft.ScrollMode.ALWAYS, height=650),
]
while True:
    if destination_label != e.control.destinations[e.control.selected_index].label:
        break
    if e.control.destinations[e.control.selected_index].label == "Processes":

        processes = Processes().get_processes()

        sorted_processes = sorted(processes,
                                   key=lambda x: x[[y for y in
x.keys()][processes_table.sort_column_index]],
                                   reverse=not processes_table.sort_ascending, )

        data_rows = []
        for process in sorted_processes:
            data_rows.append(
                ft.DataRow([ft.DataCell(ft.Text(process.get("pid"))),
                             ft.DataCell(ft.Text(process.get("name"))),
                             ft.DataCell(ft.Text(process.get("cpu_percent"))),
                             ft.DataCell(ft.Text(process.get("memory")))]
                )

        if processes:
            processes_table.rows = data_rows

        page.update()

    else:
        break

def render_network(e):

```

```

destination_label = e.control.destinations[e.control.selected_index].label
data = []
body.controls = [
    ft.Column(data, scroll=ft.ScrollMode.ALWAYS, height=650),
]
while True:
    if destination_label != e.control.destinations[e.control.selected_index].label:
        break
    if e.control.destinations[e.control.selected_index].label == "Network":
        informations = NetworkTrafficMonitor().get_network_usage()
        for information in informations:
            for key, value in information.items():
                data.append(
                    ft.Row(
                        [
                            ft.Text(f'{key}:"',
style=ft.TextStyle(weight=ft.FontWeight.BOLD), size=20),
                            ft.Text(f' {value} "', size=20),
                        ]
                    )
                )

            page.update()
            data = []
    else:
        break

def render_logs(e):
    destination_label = e.control.destinations[e.control.selected_index].label

    processes_table = ft.DataTable(
        width=1000,
        bgcolor=ft.colors.GREEN_600,
        border=ft.border.all(1, ft.colors.BLUE_200),
        border_radius=10,
        vertical_lines=ft.border.BorderSide(2, ft.colors.BLUE_200),
        horizontal_lines=ft.border.BorderSide(1, "green"),
        sort_column_index=2,
        heading_row_color=ft.colors.BLACK12,
        heading_row_height=100,
        divider_thickness=0,
        data_row_color=ft.colors.GREY_900,
        data_row_min_height=100,
        data_row_max_height=300,
        columns=[

```

```

ft.DataColumn(
    ft.Text("Event ID", color=ft.colors.BLACK, size=20),
    on_sort=lambda e: change_sort_index(e.column_index),
),
ft.DataColumn(
    ft.Text("Time", color=ft.colors.BLACK, size=20),
    on_sort=lambda e: change_sort_index(e.column_index),
),
ft.DataColumn(
    ft.Text("Source Name", color=ft.colors.BLACK, size=20),
    on_sort=lambda e: change_sort_index(e.column_index),
),
ft.DataColumn(
    ft.Text("Message", color=ft.colors.BLACK, size=20),
    on_sort=lambda e: change_sort_index(e.column_index),
),
],
)

def change_sort_index(index):
    processes_table.sort_column_index = index
    processes_table.sort_ascending = not processes_table.sort_ascending
    update_logs_DataRow(max_logs=int(dd_max.value), value=dd.value)

def dropdown_changed(e):
    dd.value = dd.value
    update_logs_DataRow(value=dd.value)
    page.update()

def dropdown_max_changed(e):
    dd_max.value = dd_max.value
    update_logs_DataRow(max_logs=int(dd_max.value))
    page.update()

dd = ft.Dropdown(
    width=150,
    value='System',
    options=[
        ft.dropdown.Option("System"),
        ft.dropdown.Option("Application"),
        ft.dropdown.Option("Security"),
        ft.dropdown.Option("Setup"),
        ft.dropdown.Option("Forwarded Events"),
    ],

```

```

        on_change=dropdown_changed,
    )

    dd_max = ft.Dropdown(
        width=100,
        value="50",
        options=[
            ft.dropdown.Option("50"),
            ft.dropdown.Option("100"),
            ft.dropdown.Option("200"),
            ft.dropdown.Option("500"),
            ft.dropdown.Option("1000"),
        ],
        on_change=dropdown_max_changed,
    )

    body.controls = [
        ft.Row([dd, ft.Text("   "), dd_max]),
        ft.Column([processes_table], scroll=ft.ScrollMode.ALWAYS, height=650),
    ]

    def update_logs_DataRow(value="System", max_logs=50):

        processes = SystemInformation().get_event_logs(total_logs=max_logs,
logtype=value)

        sorted_processes = sorted(processes,
                                key=lambda x: x[
                                    [y for y in x.keys()]
                                    [processes_table.sort_column_index]
                                ],
                                reverse=not processes_table.sort_ascending, )

        data_rows = []
        for process in sorted_processes:
            data_rows.append(
                ft.DataRow([ft.DataCell(ft.Text(process.get("Event ID"))),
                            ft.DataCell(ft.Text(process.get("Time"))),
                            ft.DataCell(
                                ft.Container(
                                    content=ft.Text(process.get("Source Name")),
text_align=ft.TextAlign.JUSTIFY),
                                    width=200),
                            ),
                            ft.DataCell(

```

```

        ft.Container(
            content=ft.Text(process.get("Message"),
text_align=ft.TextAlign.JUSTIFY),
            width=250)

        ))

    )

    if processes:
        processes_table.rows = data_rows

    page.update()

    update_logs_DataRow()

def change_body(e):
    global last_page_index
    destination_label = e.control.destinations[e.control.selected_index].label
    if last_page_index != e.control.selected_index:
        last_page_index = e.control.selected_index
        if destination_label == 'Usage':
            render_usage(e)
        elif destination_label == "Temp":
            render_temp(e)
        elif destination_label == "Info":
            render_system_info(e)
        elif destination_label == "Processes":
            render_processes(e)
        elif destination_label == "Network":
            render_network(e)
        elif destination_label == "System journal":
            render_logs(e)
        else:
            body.controls = [ft.Column([ft.Text(f"{destination_label}")])]
            page.update()

rail = ft.NavigationRail(
    selected_index=0,
    label_type=ft.NavigationRailLabelType.ALL,
    extended=True,
    min_width=100,
    min_extended_width=200,
    group_alignment=-0.9,
    destinations=[
        ft.NavigationRailDestination(

```

```

        icon=ft.icons.INFO,
        label="Info"
    ),
    ft.NavigationRailDestination(
        icon=ft.icons.DATA_USAGE,
        label="Usage",
    ),
    ft.NavigationRailDestination(
        icon=ft.icons.THERMOSTAT,
        label="Temp",
    ),
    ft.NavigationRailDestination(
        icon=ft.icons.MONITOR,
        label="Processes"
    ),
    ft.NavigationRailDestination(
        icon=ft.icons.NETWORK_WIFI,
        label="Network"
    ),
    ft.NavigationRailDestination(
        icon=ft.icons.LIBRARY_BOOKS,
        label="System journal"
    ),
],
on_change=change_body,
)

layout = ft.Row(
    [
        rail,
        ft.VerticalDivider(width=1),
        body
    ],
    expand=True,
)

page.add(layout)
render_system_info()

if __name__ == "__main__":
    ft.app(target=main)

```

functions.py

```

import platform
import GPUtil
import psutil
import time
import WinTmp
import win32evtlog

class SystemMonitor:
    def __init__(self):
        self.running = False

    def start_monitoring(self):
        self.running = True
        while self.running:
            self.update_system_info()
            time.sleep(1)

    def stop_monitoring(self):
        self.running = False

    def update_system_info(self):
        cpu_usage = psutil.cpu_percent()
        memory_usage = psutil.virtual_memory().percent
        disk_usage = psutil.disk_usage('/').percent
        fans = psutil.sensors_fans()
        print("\n==== System Monitoring =====")
        print(f"CPU Usage: {cpu_usage}%")
        print(f"Memory Usage: {memory_usage}%")
        print(f"Disk Usage: {disk_usage}%")
        print(f"Fans Speed {fans}")

    def set_thresholds(self):
        pass

class CPUUsageMonitor:
    def get_cpu_count(self):
        """Get the number of logical CPUs."""
        try:
            return psutil.cpu_count()
        except Exception:
            return 0

```

```

def get_cpu_frequency(self):
    """Get the current CPU frequency."""
    try:
        return psutil.cpu_freq()
    except Exception:
        return 0
def get_cpu_utilization(self):
    """Get the current CPU utilization."""
    try:
        return psutil.cpu_percent()
    except Exception:
        return 0

class MemoryUsageMonitor:
    def __init__(self):
        pass

    @staticmethod
    def get_available_memory():
        """
        Get the available physical memory in bytes.
        """
        try:
            return psutil.virtual_memory().available
        except Exception:
            return 0

    @staticmethod
    def get_total_memory():
        """
        Get the available physical memory in bytes.
        """
        try:
            return psutil.virtual_memory().total
        except Exception:
            return 0

    @staticmethod
    def get_used_memory():
        """
        Get the used physical memory in bytes.
        """
        try:
            return psutil.virtual_memory().used
        except Exception:
            return 0

```



```

@staticmethod
def get_memory_usage_percentage():
    """
    Get the percentage of used physical memory.
    """
    try:
        return psutil.virtual_memory().percent
    except Exception as e:
        print(e)
        return 0

class DiskSpaceMonitor:
    def __init__(self):
        pass

    def get_all_disks(self):
        """
        Get a list of all disk partitions.
        """
        return psutil.disk_partitions(all=True)

    def get_disk_usage(self, disk):
        """
        Get disk usage information for a specific disk.
        """
        return psutil.disk_usage(disk)

    def get_all_disk_usage(self):
        """
        Get disk usage information for all disks.
        """
        all_disks = self.get_all_disks()
        disk_usage_info = {}
        for disk in all_disks:
            usage = self.get_disk_usage(disk.mountpoint)
            disk_usage_info[disk.mountpoint] = {
                "total": usage.total,
                "used": usage.used,
                "free": usage.free,
                "percent": usage.percent
            }
        return disk_usage_info

```

```

class NetworkTrafficMonitor:
    def get_network_usage(self):
        net_stats = psutil.net_io_counters(pernic=True)
        networks = []
        for interface, stats in net_stats.items():
            networks.append({
                'Interface': interface,
                "Bytes send": stats.bytes_sent,
                "Bytes received": stats.bytes_recv,
                "Packets sent": stats.packets_sent,
                "Packets received": stats.packets_recv,
                "Error in/out": f"{stats.errin}/{stats.errout}",
                "Drop in/out": f"{stats.dropin}/{stats.dropout}",
            })
        return networks

```

```

class Temperature:

```

```

    def get_cpu_temperature(self):
        """ CPU temperature """
        return WinTmp.CPU_Temp()

```

```

    def get_gpu_temperature(self):
        """ GPU temperature """
        return WinTmp.GPU_Temp()

```

```

class SystemInformation:

```

```

    def get_basic_system_info(self):
        system_info = {'Platform': platform.platform(), 'Architecture': "
".join(platform.architecture()),
                        'System_name': platform.system() + " " + platform.release(), 'Version':
platform.version(),
                        'Machine': platform.machine(), 'Processor': platform.processor(),
                        'Cpu Count': psutil.cpu_count(),
                        "Ram Size": str(round(psutil.virtual_memory().total / 1024**3, 2)) + "
GB",
                        }

```

```

        gpus = GPUUtil.getGPUs()
        for index, gpu in enumerate(gpus):
            system_info[f'GPU {index}'] = gpu.name

```

```

return system_info

def get_event_logs(self, total_logs=50, logtype="System"):
    print(logtype)
    server = None # None implies local machine
    hand = win32evtlog.OpenEventLog(server, logtype)
    flags = win32evtlog.EVENTLOG_BACKWARDS_READ |
win32evtlog.EVENTLOG_SEQUENTIAL_READ
    total = 0
    logs = []
    try:
        events = win32evtlog.ReadEventLog(hand, flags, 0)
        while events:
            for event in events:

                logs.append(
                    {"Event ID": event.EventID,
                     "Time": event.TimeGenerated,
                     "Source Name": event.SourceName,

                     "Message": str(event.StringInserts),
                    }
                )
            if total > total_logs:
                break
            total += 1
            if total > total_logs:
                break
            events = win32evtlog.ReadEventLog(hand, flags, 0)
    finally:
        win32evtlog.CloseEventLog(hand)
    return logs

```

```

class Processes:
    def get_processes(self):
        processes = []
        for process in psutil.process_iter():
            try:
                # Get process details
                process_info = {
                    'pid': process.pid,
                    'name': process.name(),
                    'cpu_percent': process.cpu_percent(),
                    "memory": process.memory_info().rss / 1024 / 1024,

```

```

    }
    processes.append(process_info)

except psutil.Error:
    pass
return processes

```

charts.py

```
import flet as ft
```

```

def get_usage_chart():
    usage_chart = ft.LineChart(
        data_series=[
            ft.LineChartData(
                data_points=[
                    # ft.LineChartDataPoint(0, 0),
                ],
                stroke_width=1,
                color=ft.colors.CYAN,
                curved=True,
                stroke_cap_round=True,
            )
        ],
        border=ft.border.all(3, ft.colors.with_opacity(0.2, ft.colors.ON_SURFACE)),
        horizontal_grid_lines=ft.ChartGridLines(
            interval=10, color=ft.colors.with_opacity(0.2, ft.colors.ON_SURFACE),
width=0.5
        ),
        vertical_grid_lines=ft.ChartGridLines(
            interval=10, color=ft.colors.with_opacity(0.2, ft.colors.ON_SURFACE),
width=0.5
        ),
        left_axis=ft.ChartAxis(
            labels=[
                ft.ChartAxisLabel(
                    value=0,
                    label=ft.Text("0%", size=10, weight=ft.FontWeight.BOLD),
                ),
                ft.ChartAxisLabel(
                    value=20,
                    label=ft.Text("20%", size=10, weight=ft.FontWeight.BOLD),
                ),
                ft.ChartAxisLabel(

```

```

        value=40,
        label=ft.Text("40%", size=10, weight=ft.FontWeight.BOLD),
    ),
    ft.ChartAxisLabel(
        value=60,
        label=ft.Text("60%", size=10, weight=ft.FontWeight.BOLD),
    ),
    ft.ChartAxisLabel(
        value=80,
        label=ft.Text("80%", size=10, weight=ft.FontWeight.BOLD),
    ),
    ft.ChartAxisLabel(
        value=100,
        label=ft.Text("100%", size=10, weight=ft.FontWeight.BOLD),
    ),

    ],
    labels_size=40,
),
bottom_axis=ft.ChartAxis(
    labels=[
        ft.ChartAxisLabel(
            value=0,
            label=ft.Container(
                ft.Text(
                    "0",
                    size=16,
                    weight=ft.FontWeight.BOLD,
                    color=ft.colors.with_opacity(0.5, ft.colors.ON_SURFACE),
                ),
                margin=ft.margin.only(top=10),
            ),
        ),
        ft.ChartAxisLabel(
            value=30,
            label=ft.Container(
                ft.Text(
                    "30",
                    size=16,
                    weight=ft.FontWeight.BOLD,
                    color=ft.colors.with_opacity(0.5, ft.colors.ON_SURFACE),
                ),
                margin=ft.margin.only(top=10),
            ),
        ),
    ],
),

```

```

        ft.ChartAxisLabel(
            value=60,
            label=ft.Container(
                ft.Text(
                    "60",
                    size=16,
                    weight=ft.FontWeight.BOLD,
                    color=ft.colors.with_opacity(0.5, ft.colors.ON_SURFACE),
                ),
                margin=ft.margin.only(top=10),
            ),
        ],
        labels_size=32,
    ),
    tooltip_bgcolor=ft.colors.with_opacity(0.8, ft.colors.BLUE_GREY),
    min_y=0,
    max_y=100,
    min_x=0,
    max_x=60,
    expand=False,
    width=700,
    height=100,
)
return usage_chart

def get_temp_chart():
    temp_chart = ft.LineChart(
        data_series=[
            ft.LineChartData(
                data_points=[
                    # ft.LineChartDataPoint(0, 0),
                ],
                stroke_width=1,
                color=ft.colors.CYAN,
                curved=True,
                stroke_cap_round=True,
            )
        ],
        border=ft.border.all(3, ft.colors.with_opacity(0.2, ft.colors.ON_SURFACE)),
        horizontal_grid_lines=ft.ChartGridLines(
            interval=10, color=ft.colors.with_opacity(0.2, ft.colors.ON_SURFACE),
            width=0.5
        ),
    )

```

```

        vertical_grid_lines=ft.ChartGridLines(
            interval=10, color=ft.colors.with_opacity(0.2, ft.colors.ON_SURFACE),
width=0.5
        ),
        left_axis=ft.ChartAxis(
            labels=[
                ft.ChartAxisLabel(
                    value=0,
                    label=ft.Text("0", size=14, weight=ft.FontWeight.BOLD,
color=ft.colors.GREEN_300),

                ),
                ft.ChartAxisLabel(
                    value=20,
                    label=ft.Text("20", size=14, weight=ft.FontWeight.BOLD,
color=ft.colors.GREEN_300),

                ),
                ft.ChartAxisLabel(
                    value=40,
                    label=ft.Text("40", size=14, weight=ft.FontWeight.BOLD,
color=ft.colors.GREEN_300),

                ),
                ft.ChartAxisLabel(
                    value=60,
                    label=ft.Text("60", size=14, weight=ft.FontWeight.BOLD,
color=ft.colors.YELLOW_300),

                ),
                ft.ChartAxisLabel(
                    value=80,
                    label=ft.Text("80", size=14, weight=ft.FontWeight.BOLD,
color=ft.colors.RED_300),

                ),
                ft.ChartAxisLabel(
                    value=100,
                    label=ft.Text("100", size=14, weight=ft.FontWeight.BOLD,
color=ft.colors.RED_500),

                ),
                ft.ChartAxisLabel(
                    value=120,
                    label=ft.Text("120", size=14, weight=ft.FontWeight.BOLD,
color=ft.colors.RED_900),

                ),

            ],
            labels_size=40,

```

```

),
bottom_axis=ft.ChartAxis(
    labels=[
        ft.ChartAxisLabel(
            value=0,
            label=ft.Container(
                ft.Text(
                    "0",
                    size=16,
                    weight=ft.FontWeight.BOLD,
                    color=ft.colors.with_opacity(0.5, ft.colors.ON_SURFACE),
                ),
                margin=ft.margin.only(top=10),
            ),
        ),
        ft.ChartAxisLabel(
            value=30,
            label=ft.Container(
                ft.Text(
                    "30",
                    size=16,
                    weight=ft.FontWeight.BOLD,
                    color=ft.colors.with_opacity(0.5, ft.colors.ON_SURFACE),
                ),
                margin=ft.margin.only(top=10),
            ),
        ),
        ft.ChartAxisLabel(
            value=60,
            label=ft.Container(
                ft.Text(
                    "60",
                    size=14,
                    weight=ft.FontWeight.BOLD,
                    color=ft.colors.with_opacity(0.5, ft.colors.ON_SURFACE),
                ),
                margin=ft.margin.only(top=10),
            ),
        ),
    ],
    labels_size=40,
),
tooltip_bgcolor=ft.colors.with_opacity(0.8, ft.colors.BLUE_GREY),
min_y=0,
max_y=120,

```



```

        min_x=0,
        max_x=60,
        expand=False,
        width=700,
        height=260,
    )
    return temp_chart

```

WinTmp.py

```

import clr
import os

clr.AddReference(
    os.path.join(
        os.path.dirname(os.path.abspath(__file__)), "LibreHardwareMonitorLib.dll"
    )
)
from LibreHardwareMonitor import Hardware

hw = Hardware.Computer()
hw.IsCpuEnabled = hw.IsGpuEnabled = hw.IsMemoryEnabled = \
    hw.IsMotherboardEnabled = hw.IsStorageEnabled = True
hw.Open()

def GPU_Temp():
    for h in hw.Hardware:
        h.Update()

        if h.HardwareType in (Hardware.HardwareType.GpuNvidia,
Hardware.HardwareType.GpuAmd, Hardware.HardwareType.GpuIntel):
            for sensor in h.Sensors:
                if sensor.SensorType == Hardware.SensorType.Temperature and "GPU
Core" in sensor.Name:
                    return sensor.Value

def CPU_Temp():
    for h in hw.Hardware:
        h.Update()

        if h.HardwareType == Hardware.HardwareType.Cpu:
            for sensor in h.Sensors:
                if sensor.SensorType == Hardware.SensorType.Temperature:
                    return sensor.Value

```