НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

УДК 004.056:336

«ПОГОДЖЕНО»	**«ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ»**
Декан факультету	Завідувач кафедри
Інформаційних технологій	Комп'ютерних систем і мереж
Болбот І.М., д.п.н., професор	Касаткін Д.Ю., к.т.н., доцент
____ _____ 2024 р.	**____ _____ 2024 р.**

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему Дослідження продуктивності та безпеки комп'ютерної системи через аналіз користувачів, процесів та захист коду в бенчмарку

Спеціальність 123 – Комп'ютерна інженерія
(код і назва)

Освітня програма Комп'ютерні системи і мережі
(назва)

Орієнтація освітньої програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Гарант освітньої програми

д.т.н., професор _____ Лахно В.А.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник магістерської кваліфікаційної роботи

к.т.н., доцент _____ Гусєв Б.С.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ Волков А.Є.
(підпис) (ПІБ студента)

КИЇВ - 2024

РЕФЕРАТ

Пояснювальна записка: 119 сторінок, 34 рисунки, 8 таблиць, 3 додатки, 27 джерел.

BENCHMARK, ЗАХИСТ КОДУ, МЕРЕЖЕВА АКТИВНІСТЬ, СТАБІЛЬНІСТЬ СИСТЕМИ, CPU, GPU, MEMORY.

Метою є розробити та дослідити комплексний підхід до аналізу продуктивності та безпеки комп'ютерної системи шляхом оцінки впливу користувачів, процесів та методів захисту коду, використовуючи бенчмаркінг.

Об’єкт дослідження - комп’ютерна система як сукупність апаратного та програмного забезпечення, що забезпечує обчислювальні та комунікаційні функції.

Предмет дослідження - взаємозв'язок продуктивності, безпеки коду та мережевої активності в рамках функціонування комп'ютерної системи.

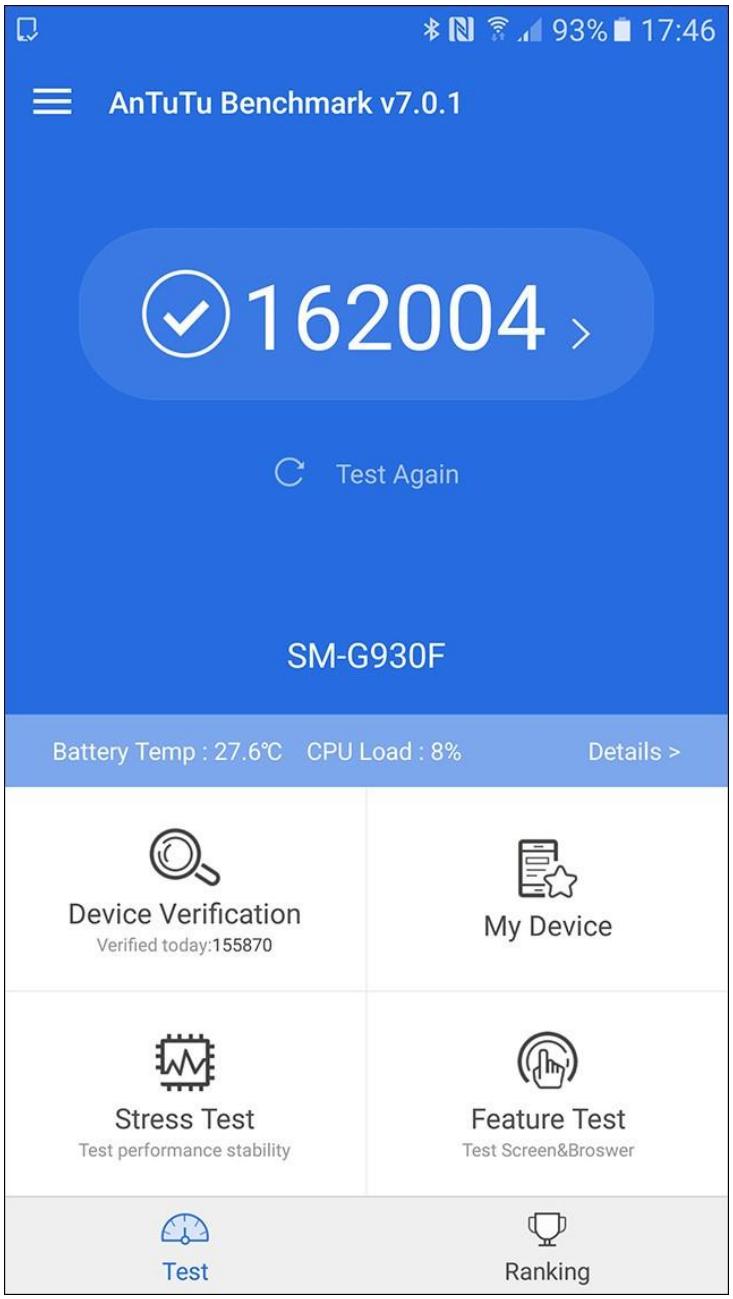
Завдання

1. Проаналізувати існуючі підходи до оцінки продуктивності та безпеки комп'ютерних систем.
2. Розробити програмне забезпечення для бенчмаркінгу, яке дозволяє контролювати ключові параметри продуктивності та безпеки системи, включаючи процесорні та мережеві ресурси.
3. Розробити механізми логування подій та їх аналіз для виявлення аномалій у роботі системи.
4. Реалізувати функції криптографічного захисту коду, включаючи створення цифрових підписів, обфускацію та перевірку цілісності файлів.
5. Провести серію тестів для визначення продуктивності різних компонентів системи та аналізу мережевих з'єднань на наявність підозрілих активностей.

Наукова новизна дослідження полягає у впровадженні інтегрованого підходу до аналізу комп'ютерної системи, що враховує одночасно параметри продуктивності та безпеки. Розроблене програмне забезпечення дозволяє виявляти аномалії у функціонуванні системи, аналізувати підозрілі мережеві з'єднання, а також перевіряти цілісність коду за допомогою цифрових підписів, що забезпечує комплексний підхід до діагностики комп'ютерної системи.

Практична значимість роботи полягає у можливості застосування розробленого програмного забезпечення для моніторингу та оцінки безпеки і продуктивності комп'ютерних систем у реальних умовах експлуатації. Програма дозволяє здійснювати бенчмаркінг ресурсів, контролювати мережеву активність та своєчасно реагувати на загрози, що забезпечує стабільну та безпечну роботу системи на підприємствах, в організаціях та для окремих користувачів.

ЗМІСТ



Перелік умовних позначень

ПЗ	–	Програмне забезпечення
GPU	–	Графічний процесор
CPU	–	Центральний процесор

ВСТУП

З розвитком інформаційних технологій та зростанням обсягу даних питання продуктивності та безпеки комп’ютерних систем стають дедалі важливішими для різноманітних галузей, включаючи бізнес, державні установи та особисте користування. У сучасних умовах стабільна робота ІТ-інфраструктури є фундаментом для безперебійного виконання бізнес-процесів, забезпечення надійності обробки даних та захисту від зовнішніх загроз. Оцінка продуктивності комп'ютерних систем дозволяє виявляти слабкі місця в їх роботі, попереджати перевантаження ресурсів та підвищувати ефективність виконання завдань.

У відповідь на зростаючі вимоги до безпеки та продуктивності розроблено програмне забезпечення, яке дозволяє комплексно моніторити використання системних ресурсів, таких як центральний процесор, оперативна пам’ять та накопичувачі, а також забезпечувати контроль над мережею, виявляти шкідливі з'єднання та оцінювати загрози безпеці. Це ПЗ надає можливості аналізу поведінки користувачів та процесів у системі, а також здійснює моніторинг якості роботи системи в реальному часі, що є основою для глибокого бенчмаркінгу та оптимізації роботи комп’ютера. Завдяки цьому, користувачі отримують змогу швидко реагувати на перевантаження або підозрілу активність, що значно підвищує рівень безпеки та стабільності системи. У сучасному кіберпросторі існує безліч загроз, серед яких найбільшу небезпеку становлять шкідливе програмне забезпечення, зловмисні мережеві підключення та кібернапади. Разом з цим зростає потреба в продуктивності, особливо для виконання критично важливих завдань, що робить моніторинг системних ресурсів необхідною складовою підтримки стабільної роботи систем. Розроблене ПЗ не лише дозволяє виявляти та ізолювати кіберзагрози, але й забезпечує ефективне

управління ресурсами, пропонуючи комплексний підхід до аналізу, контролю та оптимізації продуктивності.

Дане дослідження є актуальним, оскільки воно поєднує в собі вивчення як продуктивності, так і безпеки, включаючи детальний аналіз використання ресурсів, контроль за мережею, оцінку процесів на наявність шкідливих елементів, а також захист коду від несанкціонованих змін. Такий підхід дозволяє об’єднати контроль над продуктивністю та безпекою системи в єдиному інструменті, що є новим підходом до забезпечення стабільності комп'ютерних систем. Результати дослідження мають практичну цінність для різних категорій користувачів — від IT-спеціалістів до кінцевих користувачів, які прагнуть отримати максимальну продуктивність і безпеку від своєї системи.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність теми

В умовах розвитку інформаційних технологій та цифровізації продуктивність і безпека комп'ютерних систем стають важливими критеріями ефективності роботи як окремих користувачів, так і цілих підприємств та організацій. Сьогодні комп'ютерні системи відіграють роль критично важливої інфраструктури, яка підтримує безперервність бізнес-процесів, обробку даних, комунікацію та надання послуг. Відповідно, зростає потреба у вдосконаленні систем захисту та підвищенні продуктивності, що робить дослідження у цій галузі вкрай актуальним.

Продуктивність комп'ютерної системи напряму впливає на швидкість і якість виконання завдань. Низька продуктивність може призвести до затримок у роботі, уповільнення обробки даних і навіть зупинки бізнес-процесів. В умовах, коли багато підприємств працюють в режимі реального часу, швидке та ефективне виконання завдань є вирішальним фактором конкурентоспроможності. Бенчмаркінг системних ресурсів, включаючи процесор, оперативну пам'ять, накопичувачі, дозволяє оцінити поточний стан системи та визначити її здатність до виконання робочих навантажень. Отже, оптимізація продуктивності комп'ютерних систем є необхідною для підвищення загальної ефективності та забезпечення високої якості послуг. Однією з важливих переваг використання бенчмаркінгу є можливість виявлення "вузьких місць" у продуктивності. Наприклад, аналіз завантаження процесора та використання пам'яті дозволяє виявити процеси, що займають надмірну кількість ресурсів, або програми, які можуть негативно впливати на загальну продуктивність системи. Сучасні кіберзагрози стають все складнішими, а частота атак на комп'ютерні системи зростає. Неналежний рівень безпеки може призвести до витоку конфіденційної інформації, фінансових втрат і навіть руйнування репутації

компаній. У зв’язку з цим питання захисту комп’ютерних систем від шкідливих програм та зловмисних дій є одним із пріоритетних. Особливо актуальним є захист від таких загроз, як несанкціонований доступ, підробка даних, атак на відмову у обслуговуванні (DoS) та інші методи кіберзлочинців. Програми для аналізу мережевих з'єднань, виявлення підозрілих активностей та проведення бенчмаркінгу дозволяють своєчасно реагувати на потенційні загрози. Наприклад, система аналізу логів може виявити аномалії у поведінці процесів, що є свідченням можливої атаки. Аналіз мережевих з'єднань допомагає розпізнати підозрілі IP-адреси, які можуть бути джерелом загрози. Отже, інтеграція безпеки у процеси моніторингу системи є вкрай актуальною задачею для запобігання атакам та захисту критичної інфраструктури.

У сучасних умовах захист коду та обфускація стають актуальними елементами забезпечення безпеки програмного забезпечення. Використання цифрових підписів та обфускації коду дозволяє захистити програмні продукти від несанкціонованого доступу та змін. Цифровий підпис гарантує, що код не був змінений після його створення, що важливо для захисту від шкідливих програм, які можуть вносити зміни у виконуваний код. Обфускація, у свою чергу, ускладнює розуміння та декомпіляцію коду зловмисниками, знижуючи ризики для користувачів. Впровадження криптографічних методів у процесі розробки та виконання програмних продуктів дозволяє знизити ризик зловживань та підвищити довіру користувачів до програмного забезпечення. В умовах, коли кіберзагрози постійно вдосконалюються, інтеграція цих методів у систему бенчмаркінгу та моніторингу комп'ютерної системи є важливим фактором захисту даних і ресурсів.

Зазвичай оцінка продуктивності та безпеки розглядаються окремо, проте сучасні вимоги до інформаційної безпеки вимагають інтегрованого підходу. Поєднання моніторингу продуктивності, безпеки та аналізу

поведінки процесів у єдиній системі дозволяє отримати більш комплексне уявлення про роботу комп'ютерної системи. Використання бенчмаркінгу для моніторингу продуктивності у поєднанні з інструментами захисту коду та аналізу мережевих з'єднань сприяє своєчасному виявленню аномалій та підвищенню загальної безпеки. Сучасні методи інтегрованого аналізу, що охоплюють різні аспекти роботи комп'ютерної системи, дозволяють краще зрозуміти природу можливих загроз, вплив процесів на продуктивність та стан безпеки. Актуальність такого підходу обумовлена необхідністю створення комплексних інструментів, які одночасно забезпечують стабільність роботи, продуктивність та захист від загроз.

Таким чином, актуальність дослідження продуктивності та безпеки комп'ютерної системи, проведеного в рамках цієї роботи, обумовлена зростанням вимог до стабільності та захищеності комп'ютерних систем в умовах стрімкого розвитку цифрових технологій. Впровадження розроблених підходів дозволить своєчасно виявляти проблеми, оптимізувати використання ресурсів, запобігати кіберзагрозам і, зрештою, підвищити ефективність роботи всієї IT-інфраструктури.

1.2 Мета дослідження

Метою є розробити та дослідити комплексний підхід до аналізу продуктивності та безпеки комп'ютерної системи шляхом оцінки впливу користувачів, процесів та методів захисту коду, використовуючи бенчмаркінг.

Об’єкт дослідження - комп’ютерна система як сукупність апаратного та програмного забезпечення, що забезпечує обчислювальні та комунікаційні функції.

Предмет дослідження - взаємозв'язок продуктивності, безпеки коду та мережевої активності в рамках функціонування комп'ютерної системи.

Завдання

1. Проаналізувати існуючі підходи до оцінки продуктивності та безпеки комп'ютерних систем.
2. Розробити програмне забезпечення для бенчмаркінгу, яке дозволяє контролювати ключові параметри продуктивності та безпеки системи, включаючи процесорні та мережеві ресурси.
3. Розробити механізми логування подій та їх аналіз для виявлення аномалій у роботі системи.
4. Реалізувати функції криптографічного захисту коду, включаючи створення цифрових підписів, обфускацію та перевірку цілісності файлів.
5. Провести серію тестів для визначення продуктивності різних компонентів системи та аналізу мережевих з'єднань на наявність підозрілих активностей.

Наукова новизна дослідження полягає у впровадженні інтегрованого підходу до аналізу комп'ютерної системи, що враховує одночасно параметри продуктивності та безпеки. Розроблене програмне забезпечення дозволяє виявляти аномалії у функціонуванні системи, аналізувати підозрілі мережеві з'єднання, а також перевіряти цілісність коду за допомогою цифрових підписів, що забезпечує комплексний підхід до діагностики комп'ютерної системи.

Практична значимість роботи полягає у можливості застосування розробленого програмного забезпечення для моніторингу та оцінки безпеки і продуктивності комп'ютерних систем у реальних умовах експлуатації. Програма дозволяє здійснювати бенчмаркінг ресурсів, контролювати мережеву активність та своєчасно реагувати на загрози, що забезпечує стабільну та безпечну роботу системи на підприємствах, в організаціях та для окремих користувачів.

1.3 Визначення негативних наслідків використання застарілих гаджетів

Негативні наслідки використання застарілих гаджетів:

Використання застарілих гаджетів у сучасному світі може мати кілька негативних наслідків, які варто враховувати. Першим з них є обмежені можливості та функціонал таких пристроїв. Застарілі гаджети зазвичай не мають доступу до останніх оновлень програмного забезпечення та операційних систем, що призводить до обмеження їх функціональних можливостей. Наприклад, старі смартфони можуть не підтримувати нові функції та додатки, що доступні для новіших моделей. Це може обмежити користувачів у використанні нових технологій та послуг, які вимагають сучасних пристроїв.

Другим негативним наслідком є потенційна загроза безпеці та приватності. Застарілі гаджети часто не отримують оновлень безпеки, що робить їх більш вразливими до кібератак та злому системи. Вони можуть мати вразливості, які можуть бути використані зловмисниками для доступу до особистих даних користувача. Крім того, старі пристрої можуть бути менш здатними захищати особисту інформацію, так як в них відсутні останні заходи безпеки.

Третім негативним наслідком є втрата конкурентоспроможності на ринку праці. У сучасному світі знання та вміння використовувати новітні технології стають все важливішими для отримання роботи та розвитку кар'єри. Використання застарілих гаджетів може обмежити можливості працівника і знизити його конкурентоспроможність. Роботодавці все частіше шукають працівників, які володіють навичками роботи з сучасними пристроями та технологіями. Тому використання застарілих гаджетів може стати перешкодою у просуванні по кар'єрній драбині та вплинути на можливість отримати високооплачувану роботу.

Крім того, використання застарілих гаджетів може також викликати технологічне відставання. У світі, де технологічний прогрес швидко розвивається, залишатися зі старими пристроями може призвести до втрати можливостей та набагато повільнішого виконання завдань. Нові технології та пристрої можуть забезпечити більшу продуктивність, ефективність та зручність в роботі та особистому житті. Отже, використання застарілих гаджетів має свої негативні наслідки, такі як обмежені можливості, загроза безпеці, втрата конкурентоспроможності та технологічне відставання. У сучасному світі, де технології швидко розвиваються, важливо оновлювати свої пристрої та використовувати сучасні технології для забезпечення більшого комфорту, ефективності та безпеки.

1.4 Відомі технологічні рішення

1.4.1 AnTuTu Benchmark

AnTuTu Benchmark є популярним тестом продуктивності для мобільних пристроїв, який оцінює їх характеристики, включаючи процесор, графіку, пам'ять та інші аспекти. Це комерційний продукт, розроблений компанією AnTuTu[1]. Розробка повноцінного AnTuTu Benchmark виходить за межі обсягу одного короткого коду в рамках цього чату. Однак, я можу надати загальний огляд того, як AnTuTu Benchmark може працювати та які характеристики можуть вимірюватись[2]. Основна ідея AnTuTu Benchmark полягає в тому, що він виконує різні завдання, які вимагають великої обчислювальної потужності, пам'яті, графічних можливостей тощо. Потім результати вимірювань обробляються і формуються у вигляді загального балу або окремих показників для кожної характеристики[2].

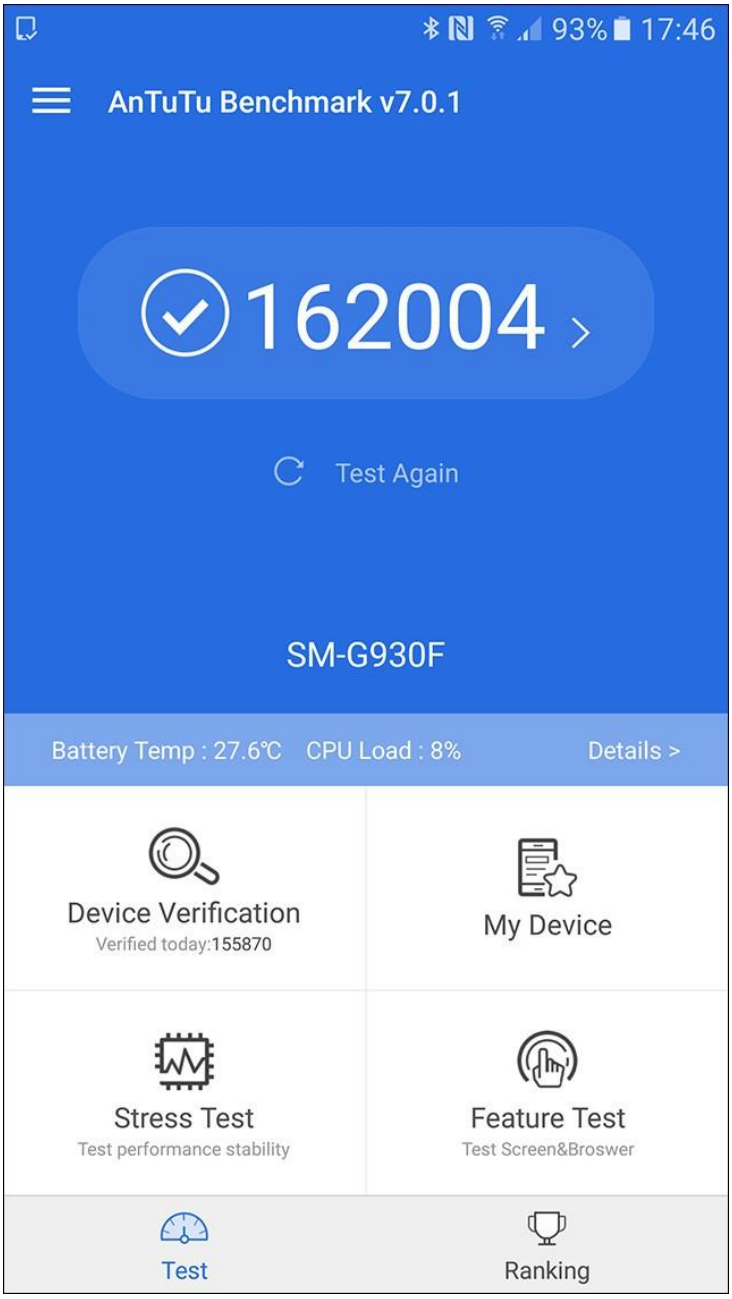


Рисунок 1.1 - Antutu benchmark[1]

Наприклад, можуть бути виміряні такі характеристики[1]:

1. Виконання складних обчислювальних завдань або тестування швидкості виконання інструкцій.
2. Тестування продуктивності 3D-графіки, таких як рендеринг, освітлення, текстурування і швидкість кадрів на різних сценах.
3. Зчитування і запис даних, швидкість доступу до пам'яті.

4. Швидкість читання і запису даних на пам'ять пристрою, включаючи внутрішню пам'ять і карту пам'яті.
5. Швидкість передачі даних через Wi-Fi, Bluetooth, мережу 4G або 5G.
6. Вимірювання часу завантаження додатків, тестування швидкості реакції на дотик, швидкість роботи сенсорів тощо.

Загальний бал або результат тесту може бути використаний для порівняння продуктивності між різними мобільними пристроями.

Конкретні деталі реалізації AnTuTu Benchmark є комерційною інформацією, і самостійне створення повноцінного клону AnTuTu Benchmark вимагає значних зусиль і досліджень. AnTuTu Benchmark використовується мільйонами користувачів по всьому світу і є одним з найвідоміших інструментів для тестування продуктивності мобільних пристроїв. AnTuTu Benchmark включає тести, що вимірюють продуктивність процесора, графіки, оперативної пам'яті, швидкості зберігання даних та інших компонентів пристрою. Інтерфейс AnTuTu Benchmark є простим і інтуїтивно зрозумілим. Користувачі можуть легко запустити тест та отримати результати зазначених аспектів продуктивності. AnTuTu Benchmark надає користувачам загальний бал, який відображає продуктивність пристрою в порівнянні з іншими пристроями на ринку. AnTuTu Benchmark регулярно оновлюється, щоб врахувати нові моделі пристроїв та оновлення апаратного та програмного забезпечення.

Існують випадки, коли деякі виробники пристроїв намагалися маніпулювати результатами тесту, оптимізуючи пристрої саме для AnTuTu Benchmark і не завжди відображаючи реальну продуктивність. AnTuTu Benchmark зосереджений переважно на вимірюванні продуктивності апаратного забезпечення, іноді недостатньо враховуючи оптимізацію програмного забезпечення, що може вплинути на загальну продуктивність пристрою. AnTuTu Benchmark використовує певні тестові сценарії, які не можуть повністю відображати всі можливості та використання пристрою у

реальних умовах. З релізом нових версій AnTuTu Benchmark можуть змінюватись тестові алгоритми та параметри, що можуть вплинути на порівняння результатів між різними версіями програми.

Незважаючи на деякі недоліки, AnTuTu Benchmark залишається популярним інструментом для співставлення та порівняння продуктивності смартфонів та планшетів. Проте, при оцінці продуктивності пристроїв варто враховувати й інші бенчмарк тести та реальні умови експлуатації.

1.4.2 Geekbench

Geekbench є ще одним популярним інструментом для тестування продуктивності комп'ютерів і мобільних пристроїв. Він розроблений компанією Primate Labs і дозволяє оцінити швидкість обробки процесора та пам'яті пристрою[5].

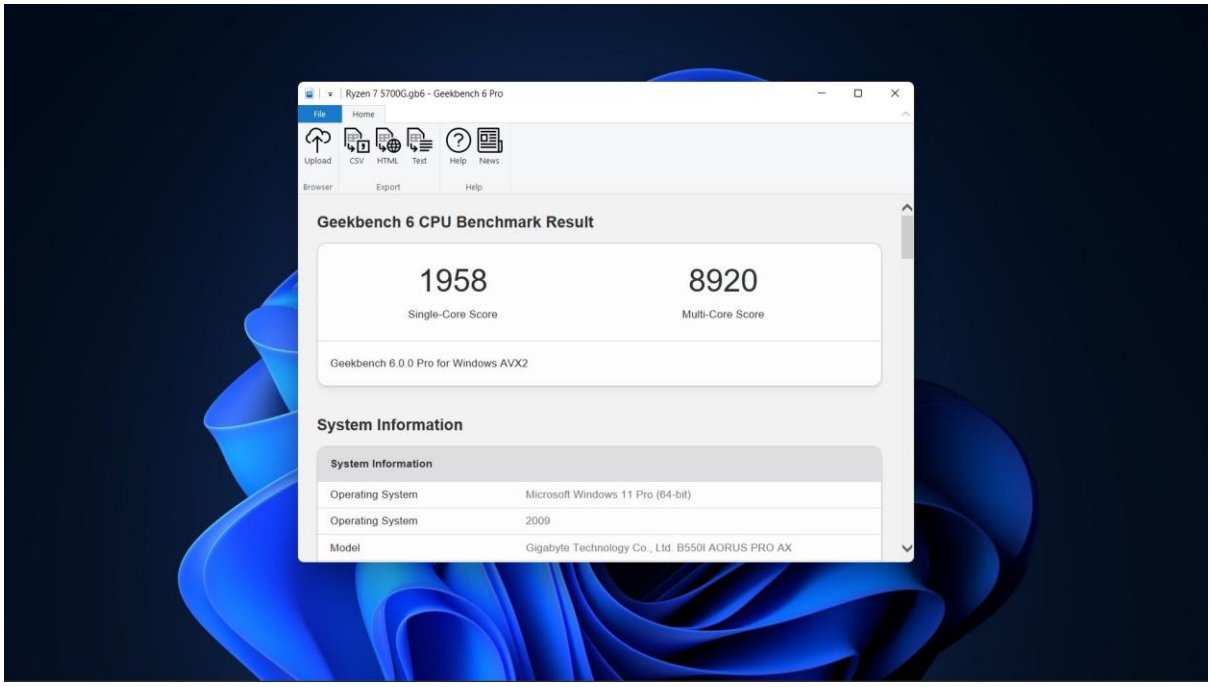


Рисунок 1.2 - Geekbench[3]

Geekbench[3] проводить різні обчислювальні завдання, включаючи ряд фіксованих операцій, математичних розрахунків і роботу з пам'яттю,

для вимірювання швидкості процесора. Geekbench може проводити тести як в однопотоковому, так і в багатопотоковому режимах, що дозволяє оцінити продуктивність пристрою при використанні багатьох паралельних процесів. Geekbench також тестує швидкість доступу до оперативної пам'яті та його продуктивність. Geekbench підтримує тестування на різних операційних системах, включаючи Windows, macOS, Linux, iOS та Android, що дозволяє порівняти продуктивність пристроїв на різних платформах.

Результати тестування Geekbench представлені у вигляді числових оцінок для однопотокового та багатопотокового режимів. Ці оцінки можуть використовуватись для порівняння продуктивності різних пристроїв.

Geekbench є комерційним продуктом, але він надає безкоштовну пробну версію, яка дозволяє виконувати базові тести. Повна версія має додаткові функції та можливості. Geekbench підтримує багато різних операційних систем, включаючи iOS, Android, macOS, Windows та Linux. Це дозволяє порівнювати продуктивність різних пристроїв, незалежно від їхньої платформи. Інтерфейс Geekbench є зручним та інтуїтивно зрозумілим. Користувачі можуть легко запустити тест та отримати результати без складних налаштувань або конфігурацій. Geekbench оцінює продуктивність не тільки процесора, але й інших компонентів, таких як оперативна пам'ять та різні аспекти графічного прискорювача. Це дає більш повну картину продуктивності пристрою. Geekbench надає окремі показники для однопотокової та багатопотокової продуктивності, що дозволяє оцінити різні аспекти роботи процесора. Geekbench використовує власну шкалу балів для оцінки продуктивності, що дозволяє порівняти пристрої між собою і визначити їхнє місце в рейтингу.

Однак, Geekbench сконцентрований на вимірюванні продуктивності процесора та оперативної пам'яті, тому він може не враховувати інші важливі компоненти, такі як графічний прискорювач чи швидкість зберігання даних. Деякі виробники пристроїв можуть оптимізувати свої

пристрої саме для Geekbench, що може вплинути на реальну продуктивність поза межами тесту. Geekbench надає результати лише для тестових умов, але не враховує фактичне використання пристрою в реальному середовищі. Незважаючи на ці недоліки, Geekbench залишається корисним інструментом для порівняння продуктивності пристроїв. Проте, варто враховувати й інші бенчмарк тести та реальні умови експлуатації, щоб отримати більш повну оцінку продуктивності пристрою. Важливо врахувати, що результати Geekbench можуть бути вплинути багатьма факторами, включаючи конфігурацію пристрою, оптимізацію програмного забезпечення та інші фактори, тому вони слід розглядати як орієнтовні показники, а не абсолютну міру продуктивності.

1.4.3 Novabench

Novabench є програмою для тестування продуктивності комп'ютерів. Вона вимірює швидкість обчислень, графічну продуктивність, швидкість доступу до диску та інші параметри системи.[3]

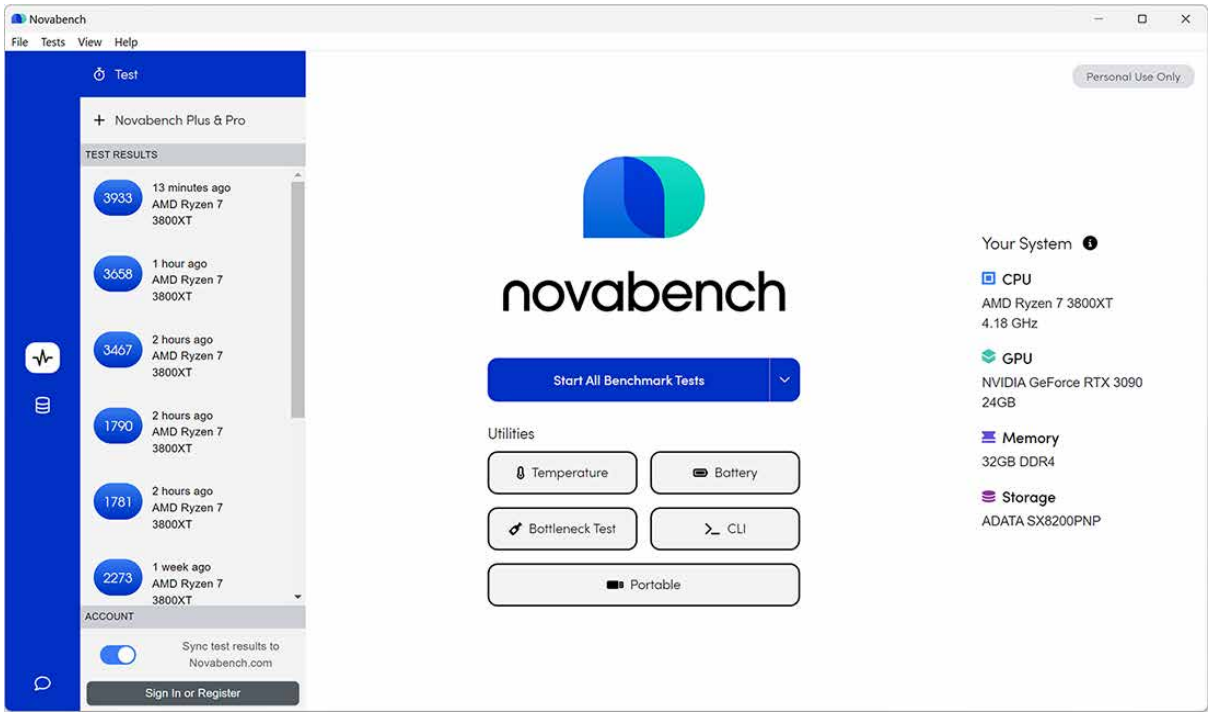


Рисунок 1.3 - Novabench

CPU тест виконує ряд обчислювальних завдань, щоб виміряти продуктивність процесора, включаючи швидкість обробки даних та мультимедіа[4]. GPU тест - тестує графічну продуктивність шляхом виконання різних графічних завдань, таких як рендеринг сцен, обчислення фізики та інші. RAM тест вимірює швидкість доступу до оперативної пам'яті і виконання різних операцій з пам'яттю. Disk тест - тестує швидкість читання та запису на диски, включаючи твердотільні накопичувачі (SSD) та жорсткі диски (HDD). Novabench дозволяє порівнювати результати з іншими користувачами, щоб побачити, як продуктивність вашої системи порівнюється з іншими конфігураціями. Novabench має безкоштовну версію, а також платну версію з додатковими функціями, такими як розширені тести і зручне управління результатами. Ця програма допомагає користувачам оцінити продуктивність своєї системи та порівняти її з іншими конфігураціями, а також може бути корисною при виконанні тестування та оптимізації комп'ютера. Novabench має простий та зрозумілий інтерфейс, що дозволяє користувачам легко запускати тести та отримувати результати без складних налаштувань. Novabench оцінює різні аспекти продуктивності комп'ютера, включаючи процесор, оперативну пам'ять, графічний прискорювач та швидкість зберігання даних. Це дозволяє отримати загальну картину продуктивності системи.

Novabench надає окремі показники для кожного компонента, що дозволяє оцінити їхню продуктивність і визначити можливі обмеження. Novabench використовує власну шкалу балів для оцінки продуктивності, що дозволяє порівнювати комп'ютери між собою і визначати їхнє місце в рейтингу. Novabench також має деякі додаткові інструменти, такі як тестування мережі та тестування дисплея, що розширює його функціональність. Однак, Novabench має обмежений набір тестів порівняно з іншими бенчмарк програмами. Він не охоплює деякі специфічні аспекти

продуктивності, які можуть бути важливі для деяких користувачів. Деякі системи можуть бути оптимізовані для досягнення високих балів у Novabench, що може впливати на реальну продуктивність в різних сценаріях використання. Novabench не надає глибокого аналізу результатів тестів або можливостей порівняння з попередніми результатами. Загалом, Novabench є простим і зручним інструментом для оцінки загальної продуктивності комп'ютера, але він може бути обмежений у порівнянні з більш спеціалізованими програмами.

1.4.4. 3DMark

3DMark є популярною програмою для тестування та оцінки графічної продуктивності комп'ютерів і мобільних пристроїв. Вона розроблена компанією UL (formerly Futuremark) і використовується для вимірювання продуктивності графічного процесора (GPU) та системи в цілому під час виконання високорівневих 3D-завдань.[6]

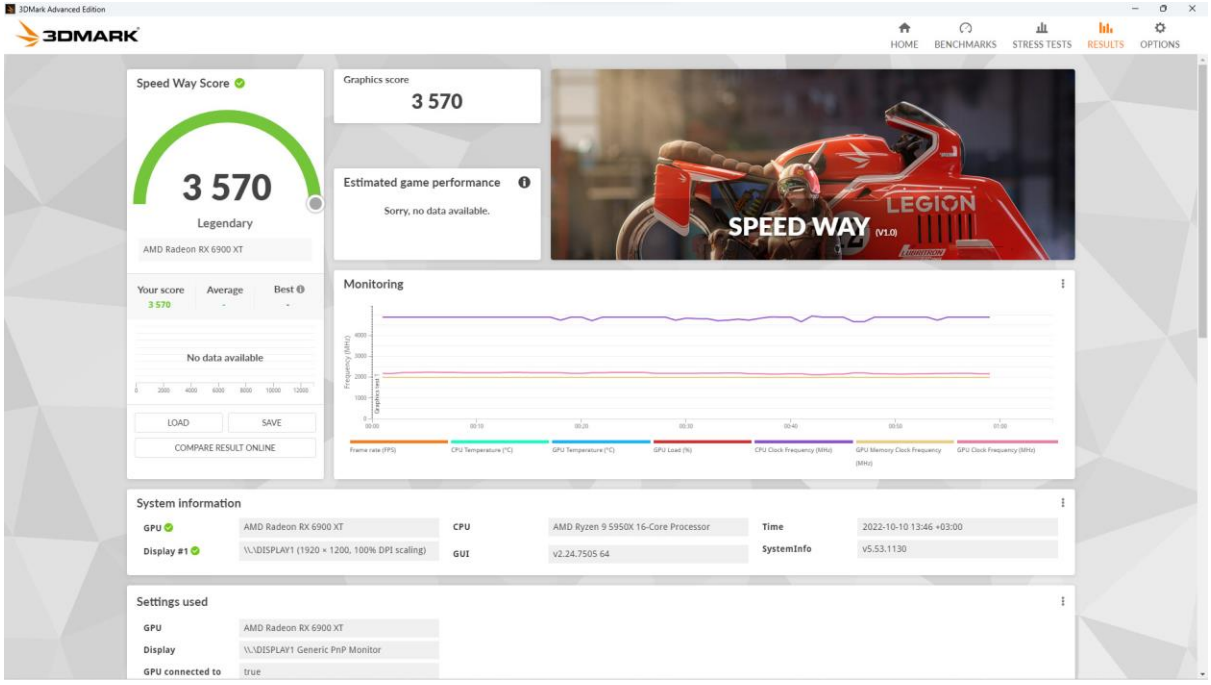


Рисунок 1.4 - 3DMark

3DMark*[6]* має різні тести та сценарії, які демонструють графічну продуктивність пристрою. Це включає тести на фізичну модель, освітлення, частоту кадрів (FPS) та інші. Після завершення тестування 3DMark надає балі для кожного тесту, що дозволяє порівняти продуктивність вашого пристрою з іншими системами. 3DMark пропонує різні режими тестування, включаючи базові тести для низькопотужних пристроїв та розширені тести для високопотужних систем. 3DMark має онлайн-таблицю результатів, де користувачі можуть завантажити свої результати тестування та порівняти їх з результатами інших користувачів. 3DMark також надає спеціальні тести, які перевіряють певні аспекти графічної продуктивності, такі як віртуальна реальність (VR), висока роздільна здатність, швидкість фізичної моделювання тощо.

3DMark доступний для різних платформ, включаючи Windows, macOS, Android та iOS. Він є комерційним продуктом і надає безкоштовну пробну версію, а також платні версії з розширеними функціями та тестами. 3DMark пропонує широкий спектр тестів, які оцінюють графічну продуктивність пристрою в різних сценаріях. Це включає тестування швидкості рендерингу, обробки фізичних ефектів, шейдерів та багато іншого. Тести 3DMark побудовані на реалістичних графічних сценах, що дозволяє оцінити продуктивність пристрою в реальних умовах використання. 3DMark підтримує різні операційні системи, включаючи Windows, Android та iOS, що дозволяє порівнювати продуктивність пристроїв на різних платформах. 3DMark надає оцінку в балах, яка дозволяє користувачам порівнювати свої пристрої з іншими моделями та визначати їхнє місце в рейтингу. 3DMark відображає детальну інформацію про апаратну специфікацію пристрою, включаючи тип процесора, кількість ядер, оперативну пам'ять, версію графічного драйвера та багато іншого.

Деякі тести 3DMark можуть вимагати потужного апаратного забезпечення, що може бути обмеженням для менш потужних систем. Певні

версії 3DMark є комерційними продуктами та потребують покупки. Це може бути недоліком для користувачів, які не бажають інвестувати додаткові кошти у програмне забезпечення для тестування продуктивності. 3DMark пропонує власні тестові сценарії, які можуть бути обмежені відображенням лише певних типів графічних обчислень. Це може не відображати повну картину реального використання графічного обладнання в різних програмах або іграх. Результати тестів можуть варіюватись в залежності від використовуваної версії 3DMark та апаратної платформи. Це може ускладнити порівняння результатів між різними системами.

1.4.5 Cinebench

Cinebench - це популярний програмний інструмент для вимірювання продуктивності процесора та графічної підсистеми (GPU). Він використовується в багатьох галузях, зокрема в аудіо/відео продакшні, геймінгу та комп'ютерній графіці, для оцінки швидкості обробки та візуалізації на різних системах. [9]

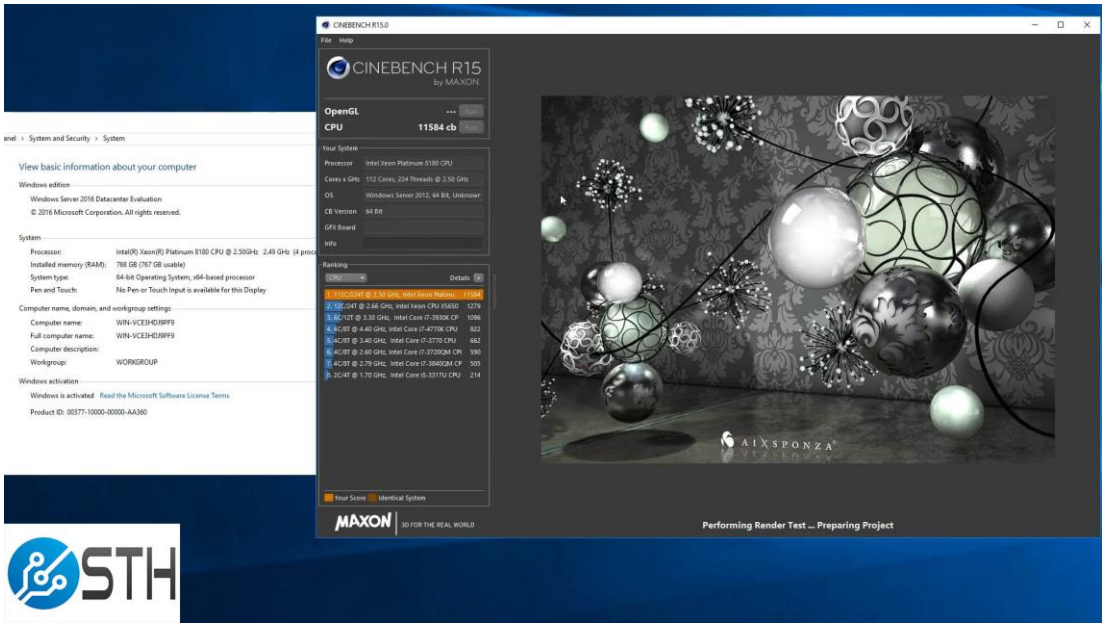


Рисунок 1.5 - Cinebench

Cinebench має простий інтерфейс, що дозволяє користувачам легко запускати тести і отримувати результати[7]. Програма пропонує тест на одновідновлення та багатовідновлення, що дозволяє оцінити швидкість обчислень, мультіпотоковість та інші параметри процесора. Cinebench також має тест на візуалізацію, що дає змогу перевірити продуктивність графічної підсистеми, зокрема швидкість відтворення 3D-сцен та обробку графічних ефектів. Cinebench надає можливість налаштувати деякі параметри тесту, такі як роздільність вікна тестування та кількість потоків, що використовуються. Один з важливих аспектів Cinebench - можливість порівняти результати з іншими системами, оскільки програма надає баллову оцінку продуктивності, яка є стандартизованою і легко інтерпретованою. Cinebench вимірює лише продуктивність процесора та графічної підсистеми, а не інших аспектів системи, таких як оперативна пам'ять, швидкість зчитування/запису на диск тощо[7]. Cinebench зорієнтований на конкретні сценарії використання, такі як рендеринг 3D-графіки, і може не відображати повну картину продуктивності системи для інших завдань або додатків. Оскільки Cinebench був розроблений певний час тому, він може не враховувати останні розробки технологій процесорів та графічних карт. Деякі компоненти системи можуть бути оптимізовані під час роботи з Cinebench, що може призвести до несвідомого перекручення результатів тестування. Важливо врахувати, що Cinebench є одним із багатьох інструментів для вимірювання продуктивності, і його використання повинно бути доповнене іншими методами тестування для отримання повної карти продуктивності системи.

1.4.6 PassMark

PassMark PerformanceTest - це програмний продукт, розроблений компанією PassMark, який дозволяє вимірювати та порівнювати продуктивність різних компонентів комп'ютерної системи. [8]

Рисунок 1.6 - PassMark

PassMark PerformanceTest пропонує широкий спектр тестів для оцінки продуктивності різних компонентів, таких як процесор, оперативна пам'ять, жорсткий диск, графічна підсистема тощо[8]. Він дозволяє користувачам провести всебічний аналіз продуктивності своєї системи. Програма має інтуїтивно зрозумілий інтерфейс, що дозволяє користувачам легко вибрати тести, налаштувати параметри і запустити тестування з мінімальними зусиллями. Після завершення тестування PerformanceTest

надає детальні звіти з результатами, які включають числові оцінки, графіки та порівняльні дані. Що допомагає користувачам отримати об'єктивну оцінку продуктивності своєї системи та порівняти її з іншими[8]. PassMark PerformanceTest має велику базу даних порівняльних результатів, яка дозволяє користувачам порівняти свої результати з іншими системами і оцінити їх в контексті загального спектру продуктивності. PerformanceTest також надає можливість проводити апаратне тестування, таке як тестування температури, напруги, швидкості вентиляторів тощо, що дозволяє контролювати стан та ефективність апаратних компонентів.

Однак, PassMark PerformanceTest є платним програмним продуктом, що може бути недоступним для безкоштовного використання деякими користувачами. Результати тестування PerformanceTest можуть бути сильно залежними від якості та характеристик апаратних компонентів системи. При оцінці продуктивності необхідно враховувати, що старі або пошкоджені компоненти можуть призводити до неправильних результатів. Деякі функції PerformanceTest можуть вимагати певні характеристики апаратного забезпечення або підтримку певних технологій. Недостатньо потужна або застаріла система може не забезпечувати повний функціонал тестувального пакету. Важливо враховувати, що PassMark PerformanceTest є одним із багатьох інструментів для вимірювання продуктивності комп'ютерних систем, і його використання повинно бути доповнене іншими методами тестування для отримання повної карти продуктивності системи.

1.4.7 PCMark 10

PCMark 10 - це програмний продукт, розроблений компанією UL (раніше Futuremark), який дозволяє вимірювати та порівнювати продуктивність комп'ютерних систем у різних сценаріях використання. [10]

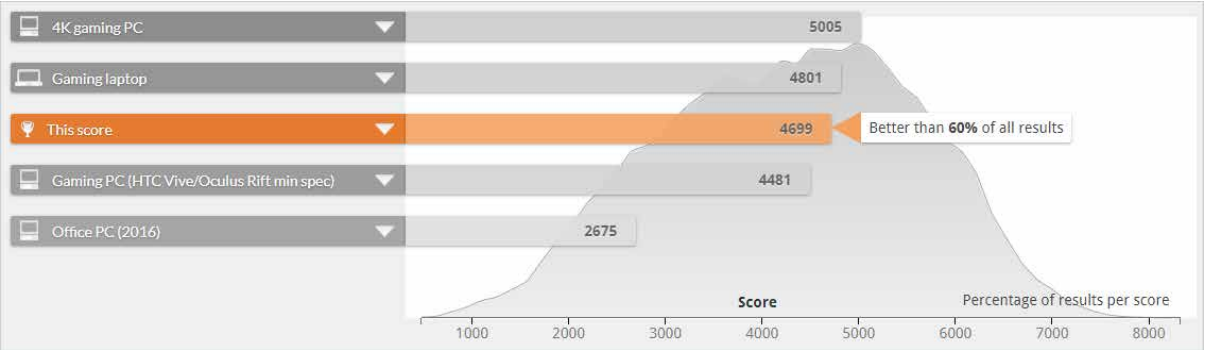


Рисунок 1.7 - PCMark 10

PCMark 10 пропонує різноманітні тести, які відображають реальні сценарії використання комп'ютера, включаючи роботу з офісними додатками, відтворення мультимедіа, веб-перегляд, фото- та відео-редагування тощо, що дозволяє оцінити продуктивність системи в контексті різних завдань[10]. Програма має інтуїтивно зрозумілий інтерфейс, що дозволяє користувачам легко вибирати тести, налаштовувати параметри і запускати тестування зі зручністю. Тести PCMark 10 спроектовані таким чином, щоб відображати реальні умови використання комп'ютера. Вони враховують взаємодію з різними додатками та завданнями, що забезпечує більш точну оцінку продуктивності системи у реальних умовах. Після завершення тестування PCMark 10 надає детальні звіти з результатами, які включають числові оцінки, графіки та порівняльні дані. Це допомагає користувачам зрозуміти, як ефективно працює їхня система у різних сценаріях використання. PCMark 10 має велику базу даних результатів, що дозволяє користувачам порівняти свої результати з іншими системами і отримати контекстуальну інформацію щодо продуктивності своєї системи. PCMark 10 є комерційним програмним продуктом, що може бути недоступним для безкоштовного використання деякими користувачами.

Результати тестування PCMark 10 можуть бути сильно залежними від якості та характеристик апаратних компонентів системи. При оцінці продуктивності необхідно враховувати, що старі або пошкоджені

компоненти можуть призводити до неправильних результатів. PCMark 10 підтримує переважно операційні системи Windows, що може обмежувати доступ до цього програмного продукту для користувачів інших платформ. Важливо враховувати, що PCMark 10 є одним з багатьох інструментів для вимірювання продуктивності комп'ютерних систем, і його використання повинно бути доповнене іншими методами тестування для отримання повної карти продуктивності системи.

1.5 Огляд існуючих алгоритмів та методів

У розробці бенчмарк тестів для смартфонів та комп'ютерів важливо враховувати різноманітні алгоритми та методи, які дозволяють ефективно вимірювати та порівнювати продуктивність цих пристроїв. [10] Нижче наведений огляд деяких існуючих підходів, які використовуються у цьому контексті. Один з найпоширеніших алгоритмів для бенчмарк тестів - це алгоритм вимірювання продуктивності CPU (центрального процесора). Він базується на виконанні різноманітних обчислювальних завдань, таких як швидкість обробки даних, кількість операцій на секунду та обсяги пам'яті, які може обробити процесор за певний час. Цей алгоритм використовується для визначення продуктивності процесора у відносних одиницях та порівняння його з іншими пристроями.[11] Ще один важливий аспект бенчмарк тестів - вимірювання продуктивності графічного процесора (GPU). Для цього використовуються алгоритми, які оцінюють швидкість обробки графічних операцій, таких як рендерінг 3D-сцен, обробка текстур та шейдерні обчислення.[10] Ці алгоритми надають змогу порівнювати продуктивність різних GPU та визначати їхні можливості у відношенні до графічних завдань. Крім того, бенчмарк тести також можуть включати оцінку продуктивності пам'яті, дискового простору та мережевого з'єднання пристрою. Для вимірювання продуктивності пам'яті використовуються алгоритми, які тестують швидкість читання та запису

даних, а також обсяг доступної пам'яті для різних завдань. [10] Вимірювання продуктивності дискового простору включає оцінку швидкості доступу до файлів та передачі даних. Оцінка мережевого з'єднання може включати швидкість передачі даних, стабільність з'єднання та пінг (затримку) при взаємодії з серверами.

Деякі бенчмарк тести також можуть використовувати методику симуляції навантаження реальних застосунків або ігор. Це дозволяє оцінити продуктивність пристрою в умовах, які максимально наближені до реального використання.[10] Наприклад, можна виміряти швидкість завантаження веб-сторінок, продуктивність при запуску важких застосунків або графічно інтенсивних ігор.

Висновки до розділу 1

Отже, в даному розділі було розглянуто актуальність та мету дипломного проекту, а також існуючі алгоритми та методи що використовуються для оцінки продуктивності системи. У загальному контексті розробки бенчмарк тестів для смартфонів та комп'ютерів, важливо обирати алгоритми та методи, які надають об'єктивні та репрезентативні результати продуктивності пристроїв.

Проблема використання застарілих гаджетів є актуальною у сучасному світі. Розширення технологій та постійний розвиток ринку електроніки призводять до швидкого застаріння пристроїв, що впливає на їх продуктивність і функціональні можливості. Це може мати негативні наслідки для користувачів, такі як обмежені можливості використання нових програм і додатків, низька продуктивність, вразливість до кібератак і т.д. Були наведені приклади таких інструментів, як AnTuTu Benchmark, Geekbench, Novabench, 3DMark, Cinebench, PassMark і PCMark 10. Кожен з цих інструментів має свої особливості та використовується для вимірювання різних аспектів продуктивності гаджетів, таких як процесор, графічний прискорювач, оперативна пам'ять тощо. У розділі було проведено огляд існуючих алгоритмів та методів для вимірювання продуктивності гаджетів і виявлення негативних наслідків використання застарілих пристроїв.

РОЗДІЛ 2 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Використані технології

2.1.1 Python

Python – багатоцільова мова програмування, яка дозволяє писати код, що добре читається. Відносний лаконізм мови Python дозволяє створити програму, яка буде набагато коротше свого аналога, написаного на іншій мові.[12] Python - багатоплатформова мова програмування. Це означає, що програми на Python можна запускати в різних операційних системах без будь-яких змін.[13] Ще однією перевагою Python є його стандартна бібліотека, яка встановлюється разом з Python і містить готові інструменти для роботи з операційною системою, веб-сторінками, базами даних, різними форматами даних, для побудови графічного інтерфейсу програм тощо.[13] Програми, написані на мові програмування Python, можуть бути як невеликими скриптами, так і складними системами.

Плюси Python[14]:

- Python має простий і зрозумілий синтаксис, що робить його легким для вивчення навіть для початківців.
- Python забезпечує швидку розробку завдяки великому набору стандартних бібліотек та фреймворків.
- Python підтримує різні парадигми програмування, включаючи процедурне, об'єктно-орієнтоване і функціональне програмування.
- Python використовується в багатьох сферах, таких як наука, веб-розробка, машинне навчання, аналітика даних, автоматизація та багато інших.
- Python має активне співтовариство розробників, яке надає багато ресурсів, бібліотек і модулів.

2.1.2 Tkinter

Tkinter є стандартною бібліотекою Python для розробки графічного інтерфейсу користувача (GUI). Вона надає можливості для створення вікон, кнопок, тексти, полів для введення даних та інших елементів, які дозволяють взаємодіяти з користувачем.[15]

Основні компоненти Tkinter[23]:

1. Вікно (tkinter.Tk) - це головне вікно програми, яке містить усі інші елементи GUI.
2. Кнопка (tkinter.Button) - елемент, який дозволяє виконувати певні дії після натискання.
3. Мітка (tkinter.Label) - елемент для відображення текстової інформації або зображень.
4. Поле для введення (tkinter.Entry) - елемент, який дозволяє користувачу вводити текст або дані.
5. Список (tkinter.Listbox) - елемент, який дозволяє відображати список елементів для вибору.
6. Меню (tkinter.Menu) - елемент, який створює розкривне меню з пунктами меню.
7. Поле прокрутки (tkinter.Scrollbar) - елемент, який дозволяє прокручувати вміст інших елементів, таких як список або текстове поле.
8. Фрейм (tkinter.Frame) - елемент, який групує інші елементи разом для організації розміщення.

Основний процес створення GUI з Tkinter включає створення вікна, створення різних елементів GUI, розміщення їх на вікні і надання їм функціональності за допомогою обробників подій.[22] [24]

2.1.3 Psutil

Psutil це міжплатформна бібліотека Python, яка надає інтерфейс для отримання інформації про системні ресурси та процеси. [16] Це розшифровується як «процесні та системні утиліти» та пропонує широкий спектр функцій для моніторингу та керування різними аспектами системи.

Бібілотека psutil дозволяє отримувати інформацію про запущені процеси, таку як ідентифікатори процесів, імена, використання процесора та пам’яті, час створення процесу тощо. Він також надає методи керування процесами, включаючи завершення або призупинення процесів. Бібліотека надає функції для моніторингу інформації, пов’язаної з системою, включаючи використання ЦП, використання пам’яті, використання диска, статистику мережі, стан акумулятора (на підтримуваних платформах) тощо. Ці показники можуть бути корисними для моніторингу системи, керування ресурсами та аналізу продуктивності. psutil дає змогу збирати інформацію про мережеві підключення, встановлені процесами, включаючи такі відомості, як локальні та віддалені адреси, стан підключення та тип протоколу. Це також дозволяє вам керувати пов’язаними з мережею аспектами, такими як припинення процесів, пов’язаних із певними з’єднаннями. Системні утиліти psutil містять утиліти для виконання різноманітних завдань, пов’язаних із системою, таких як отримання часу завантаження системи, отримання кількості ядер ЦП, запит загальносистемного часу ЦП, керування пріоритетами загальносистемних процесів тощо.

Загалом psutil спрощує процес отримання та керування інформацією, пов’язаною з системою, у міжплатформенний спосіб. Він широко використовується в програмах Python для моніторингу системи, управління ресурсами, автоматизації процесів і аналізу продуктивності.

2.1.4 Platform

Platform це модуль стандартної бібліотеки Python, який забезпечує узгоджений інтерфейс для доступу до інформації про базову платформу, таку як операційна система, апаратне забезпечення та середовище виконання. [22] Він пропонує низку функцій і атрибутів для отримання різноманітних системних деталей і виконання операцій, пов’язаних із платформою.

Таблиця 2.1 - Функції бібліотеки platform

Функція	**Опис**
platform.system()	Повертає назву операційної системи (наприклад, «Windows», «Linux», «Darwin» для macOS).
platform.release()	Повертає випускну версію операційної системи.
platform.version()	Повертає рядок, який представляє версію операційної системи.
platform.machine()	Повертає тип машини або архітектуру (наприклад, 'x86_64', 'arm64').
platform.python_implementation()	Повертає назву реалізації Python (наприклад, 'CPython', 'Jython', 'IronPython').
platform.python_version()	Повертає версію Python у вигляді рядка.
platform.processor()	Повертає ім’я чи ідентифікатор процесора.
platform.node()	Повертає мережеве ім’я машини (ім’я хоста).
platform.architecture()	Повертає розрядну архітектуру операційної системи та інтерпретатора Python (наприклад, ('64bit', 'WindowsPE')).

platform.system_alias()	Забезпечує зіставлення загальних імен платформ з канонічними іменами.
platform.uname()	Повертає іменований кортеж з детальною інформацією про систему: назва системи, вузол, випуск, версія, машина, процесор.
platform.platform()	Повертає рядок, що підсумовує платформу з якомога більшою кількістю інформації.
platform.system_alias(platform, aliased=True)	Зіставляє задану назву платформи з канонічною назвою для кращої сумісності між платформами.

Модуль platform корисний, коли вам потрібно написати незалежний від платформи код або коли вам потрібно отримати інформацію, пов’язану з системою, специфічну для середовища виконання. Він забезпечує зручний і стандартизований спосіб доступу до таких деталей, дозволяючи вашому коду легко адаптуватися до різних платформ.

2.1.5 Yara

YARA — це популярний інструмент для аналізу шкідливого програмного забезпечення, створений для допомоги дослідникам у виявленні та класифікації зразків шкідливого коду. Основна мета YARA — надати простий і гнучкий спосіб визначення та ідентифікації сімейств шкідливого програмного забезпечення на основі текстових або бінарних шаблонів.

YARA дозволяє створювати текстові правила з чітко визначеною структурою, що включає мета-дані, набори шаблонів (рядків або байт-послідовностей) і логічні умови. Такий підхід забезпечує можливість точного визначення шкідливого програмного забезпечення на основі заданих критеріїв. YARA підтримує текстові рядки, регулярні вирази та

байт-послідовності, що дозволяє створювати як прості, так і складні патерни для аналізу бінарних файлів чи текстових даних. Інструмент сумісний із Windows, Linux та macOS, що забезпечує його використання на різних операційних системах. Завдяки гнучким правилам та можливості використання різноманітних шаблонів, YARA забезпечує високу точність ідентифікації шкідливих програм, а також класифікацію їх за сімействами. Структура правил дозволяє адаптувати YARA для виявлення специфічних загроз, залежно від потреб дослідників або організацій.

Інструмент застосовується в різних сферах, таких як аналіз шкідливого ПЗ, тестування файлових систем, моніторинг серверів і робочих станцій, а також навчання в галузі кібербезпеки. YARA є потужним інструментом для дослідників шкідливого програмного забезпечення завдяки його функціональності, точності та можливості адаптації під різні завдання. Його використання сприяє підвищенню ефективності систем кібербезпеки, що робить YARA важливим елементом сучасних захисних рішень.

2.1.6 Cryptography.hazmat

Cryptography.hazmat є модулем бібліотеки Python Cryptography, який надає низькорівневі інтерфейси для роботи з криптографічними примітивами. Його функціонал призначений для досвідчених розробників, які мають глибокі знання криптографії та потребують максимальної гнучкості в реалізації безпечних рішень. Забезпечує інтерфейси для роботи з основними криптографічними алгоритмами, такими як AES, RSA, ECC, HMAC тощо. Примітиви підтримують стандартні методи шифрування, хешування, підпису та перевірки. Підтримує генерацію, імпорт та експорт ключів у різних форматах, таких як PEM і DER. Дозволяє налаштовувати параметри алгоритмів, такі як розмір ключа, режим шифрування (CBC, GCM тощо) та хеш-функції (SHA-256, SHA-3). Легко інтегрується в

екосистему Python разом із іншими модулями, такими як cryptography.x509, pyOpenSSL, або paramiko. Cryptography.hazmat — це потужний інструмент для створення кастомних криптографічних рішень, який потребує обережного та вдумливого підходу.

2.1.7 PyArmor

PyArmor — це інструмент, розроблений для захисту вихідного коду Python від несанкціонованого доступу, реверс-інжинірингу та піратства. PyArmor обфускує Python-скрипти, перетворюючи вихідний код у менш зрозумілий для людини формат і додаючи механізми ліцензування для контролю доступу до скриптів. PyArmor змінює структуру вихідного коду Python, роблячи його важким для читання та аналізу. Зберігає функціональність програми, водночас приховуючи логіку та алгоритми. Додає механізми ліцензування, які дозволяють обмежити виконання скриптів на певних пристроях або у визначені терміни. PyArmor може перетворювати скрипти у захищені файли, сумісні з C-файлами, що ускладнює їх декомпіляцію. Захищені скрипти шифруються, що робить їх недоступними без виконуваного середовища та відповідного ключа. PyArmor надає простий у використанні інтерфейс командного рядка для створення захищених версій проєктів. PyArmor — це ефективний інструмент для захисту Python-коду, який підходить як для розробників комерційних проєктів, так і для організацій, що прагнуть захистити свої інтелектуальні ресурси.

2.2 Розробка вимог

2.2.1 Функціональні вимоги

Функціональні вимоги описують, що саме повинна робити система або додаток. Вони визначають функціональність, яку система має

забезпечити для задоволення потреб користувачів, а також специфічні дії, які повинна виконувати програма. Наприклад, у нашому випадку функціональні вимоги включають моніторинг системних ресурсів, ідентифікацію мережевих підключень, зберігання логів і виявлення шкідливого ПЗ. Чітке визначення функціональних вимог допомагає розробникам зрозуміти, які функції необхідно реалізувати, і забезпечує цілеспрямовану роботу всієї команди, що прискорює розробку та підвищує якість продукту.

Таблиця 2.2 - Функціональні вимоги

ID	**Функціональна вимога**	**Опис**	**Варіанти використання**
FR-1	Ідентифікація системи	Визначення параметрів операційної системи, таких як назва, версія, архітектура та тип машини.	Отримання інформації про ОС користувача для підтримки сумісності або налагодження.
FR-2	Інформація про Python	Отримання версії та типу реалізації Python.	Перевірка сумісності коду з різними версіями Python.
FR-3	Інформація про обладнання	Отримання даних про процесор, архітектуру та мережеве ім'я машини.	Використання для діагностики апаратної сумісності.
FR-4	Моніторинг продуктивності	Збір та візуалізація показників використання системних ресурсів: CPU, пам’яті, накопичувачів.	Відстеження продуктивності для забезпечення стабільної роботи додатків.
FR-5	Моніторинг мережевої активності	Збір даних про активні мережеві підключення та виявлення підозрілих з'єднань.	Аналіз підключень для виявлення можливих загроз.
FR-6	Логування подій та аналіз аномалій	Запис подій в лог та виявлення аномальних подій або помилок у роботі системи.	Збір даних для налагодження та аналізу стабільності системи.

FR-7	Захист коду	Виконання обфускації та цифрового підпису коду для захисту від несанкціонованого доступу.	Гарантування цілісності коду та захист від модифікації.
FR-8	Сканування на шкідливе ПЗ	Сканування дисків та файлів за допомогою Yara-правил для виявлення шкідливого програмного забезпечення.	Виявлення потенційних загроз у файловій системі.
FR-9	Графічна візуалізація	Відображення графіків використання ресурсів для спрощення аналізу стану системи.	Оцінка загальної продуктивності через графічні показники.
FR-10	Управління користувачами та процесами	Збір інформації про процеси, запущені користувачами, та класифікація їх на системні та користувацькі.	Відстеження активності користувачів та системних процесів для оптимізації роботи системи та підвищення безпеки.
FR-11	Очищення логів	Видалення старих логів для звільнення місця та зниження навантаження на систему.	Забезпечення підтримки у контролі розміру журналу та забезпеченні доступного дискового простору.

2.2.2 Нефункціональні вимоги

Нефункціональні вимоги описують, як система повинна працювати з точки зору продуктивності, надійності, безпеки, зручності користування, сумісності та масштабованості. Вони стосуються технічних характеристик системи, які забезпечують ефективне функціонування та взаємодію з користувачем. Наприклад, вимоги до швидкості роботи програми, безпеки (обфускація коду, цифрові підписи) та зручності користування є важливими нефункціональними вимогами, які забезпечують зручність і стабільність системи.

Таблиця 2.3 - Нефункціональні вимоги

ID	Нефункціональна вимога	Опис
NFR-1	Швидкість виконання	Система повинна виконувати запити на отримання інформації про систему, мережу та обладнання швидко.
NFR-2	Відповідність безпеки	Використання методів обфускації та цифрових підписів для захисту коду та забезпечення його цілісності.
NFR-3	Сумісність	Підтримка роботи на різних операційних системах (Windows, Linux, macOS).
NFR-4	Зручність використання	Інтуїтивно зрозумілий графічний інтерфейс для перегляду стану ресурсів, мережевої активності та логів.
NFR-5	Можливість розширення	Система повинна дозволяти додавання нових функцій або зміни існуючих без значної перебудови коду.
NFR-6	Надійність	Програма повинна стійко працювати під час обробки різних запитів без несподіваних збоїв.
NFR-7	Масштабованість	Система повинна масштабуватися для обробки великих обсягів даних, наприклад, у процесі сканування.
NFR-8	Вимоги до ресурсів	Програма повинна ефективно використовувати системні ресурси, не створюючи надмірного навантаження.
NFR-9	Інтероперабельність	Система повинна легко інтегруватися з іншими системами моніторингу та аналізу.

Нефункціональні вимоги щодо безпеки допомагають запобігти потенційним загрозам для системи та її користувачів. У нашому випадку вимоги до обфускації коду та використання цифрових підписів допомагають захистити систему від несанкціонованого доступу та шкідливих змін.

2.3 Опис методів

Розроблена програма є комплексною системою для моніторингу продуктивності, безпеки та управління комп’ютерною системою. Вона забезпечує наступні основні функції та можливості:

- Журнал подій (Logging)

Програма реєструє інформаційні події, попередження та помилки в журналі. Це дозволяє відстежувати важливі події у програмі та виявляти потенційні проблеми. Існує можливість перегляду всіх записів, а також виділення аномалій (записів з попередженнями або помилками), що спрощує діагностику проблем. Користувач може повністю очистити журнал подій. log_event(event) фіксує події рівня INFO у журналі додатку. Використовується для відстеження нормального перебігу виконання програми, наприклад, успішного створення вкладок, запуску програми тощо. log_warning(event) записує попередження рівня WARNING. Зазвичай використовується для сповіщення про потенційні ризики чи проблеми, які не зупиняють виконання програми, але вимагають уваги. log_error(event) фіксує критичні помилки рівня ERROR, які впливають на виконання програми. Використовується для повідомлення про невдалі операції або непередбачувані ситуації.

- Моніторинг мережі

Програма показує активне підключення (Wi-Fi або Ethernet), обсяг переданих і отриманих даних. Програма перевіряє мережеві з’єднання на наявність підозрілих IP-адрес і відображає їх користувачу. Доступне вікно з деталями мережевих інтерфейсів, яке показує IP-адреси для кожного інтерфейсу.

- Безпека

Програма дозволяє створити пару ключів (приватний і публічний) для шифрування і цифрового підписання. Користувач може підписувати файли приватним ключем, що додає рівень захисту, гарантує цілісність файлів. Є можливість перевірки цілісності файлів за допомогою публічного ключа. Програма може виконувати обфускацію (маскування) коду, що підвищує захист від несанкціонованого доступу до його змісту.

- Моніторинг батареї

Відображає статус батареї (підключена чи ні) та рівень заряду у відсотках. Вираховує показник продуктивності батареї, дозволяючи оцінити її стан.

- Моніторинг процесора (CPU)

Відображає дані про тип процесора і кількість ядер. Показує поточне завантаження процесора по ядрах та вираховує показник продуктивності, що може використовуватись для діагностики навантаження на систему.

- Моніторинг пам'яті (RAM)

Відображає обсяг загальної і використаної пам’яті. Візуалізує обсяг використаної пам’яті та надає оцінку її продуктивності.

- Моніторинг сховища (Storage)

Показує загальний і використаний обсяг кожного розділу диску. Вимірює швидкість читання/запису на SSD, надаючи показник продуктивності для оцінки швидкодії.

- Сканування на шкідливе ПЗ

Можливість сканування окремих дисків або всіх доступних дисків на комп’ютері на наявність шкідливого ПЗ, використовуючи правила YARA. Програма виявляє і відображає шкідливі файли на основі відповідних шаблонів.

- Моніторинг процесів та користувачів

Відображає активні процеси для кожного користувача, включаючи системні, користувацькі і сторонні процеси. Програма аналізує навантаження на систему, використання пам’яті і CPU, надаючи рекомендації щодо подальшого використання комп’ютера.

Загалом, програма є багатофункціональним інструментом для моніторингу системи, зокрема продуктивності, безпеки та управління ресурсами комп’ютера. Вона надає користувачам можливість отримати детальну інформацію про стан системи, виявити можливі загрози та забезпечити цілісність файлів і захист даних.

Таблиця 2.4 Опис функцій

Функція	**Опис**
log_event(event)	Записує подію інформаційного рівня в журнал додатку.
log_warning(event)	Записує подію рівня попередження в журнал додатку.
log_error(event)	Записує подію рівня помилки в журнал додатку.
analyze_logs()	Зчитує журнал, фільтрує записи з попередженнями та помилками і повертає список аномалій.

count_log_levels()	Підраховує кількість записів "INFO", "WARNING" і "ERROR" у журналі та повертає ці значення.
clear_logs()	Очищає всі записи в журналі.
create_logs_tab()	Створює вкладку "Logs" у графічному інтерфейсі, показує опції для роботи з журналом та діаграму рівнів журналу.
display_logs()	Відкриває вікно, що показує повний вміст журналу.
display_anomalies()	Відкриває вікно з показом аномалій (попереджень або помилок) у журналі.
clear_logs_action()	Очищає журнал та записує цю подію в журнал.

Продовження таблиці 2.4

on_closing()	Записує подію закриття програми і виходить з програми.
create_network_tab()	Створює вкладку "Network" у графічному інтерфейсі для моніторингу мережних з'єднань і відображення підозрілих підключень.
get_current_connection()	Перевіряє активні мережні інтерфейси та визначає поточне підключення (наприклад, Wi-Fi, Ethernet).
update_network_info()	Оновлює інформацію про мережу з деталями про поточне підключення.
update_suspicious_info()	Оновлює відображення для виявлених підозрілих підключень.
detect_suspicious_connections()	Сканує активні мережні з'єднання, позначає підозрілі IP-адреси і повертає їх.
is_suspicious(ip)	Перевіряє, чи входить IP-адреса до списку підозрілих IP.

show_interface_info()	Відображає інформацію про мережеві інтерфейси у новому вікні.
update_graph()	Оновлює візуалізацію мережного використання на вкладці "Network".
create_security_tab()	Створює вкладку "Security" у графічному інтерфейсі для обробки криптографічних функцій безпеки, включаючи обфускацію коду та цифрове підписання.
generate_keys()	Генерує і зберігає нову пару ключів RSA (приватний та публічний ключі).
sign_file()	Цифрово підписує заданий файл за допомогою приватного ключа та зберігає підпис.
verify_signature()	Перевіряє цифровий підпис файлу за допомогою публічного ключа.
obfuscate_code()	Виконує обфускацію коду у вказаному файлі за допомогою зовнішньої команди.

Продовження таблиці 2.4

create_battery_tab()	Створює вкладку "Battery" у графічному інтерфейсі, відображаючи стан батареї та рівень заряду з візуалізацією.
get_battery_info()	Отримує та форматують інформацію про батарею, включаючи статус та відсоток заряду.
get_battery_percent()	Отримує поточний відсоток заряду батареї.
calculate_battery_mark()	Розраховує показник продуктивності батареї на основі відсотка заряду.
create_cpu_tab()	Створює вкладку "CPU" у графічному інтерфейсі, відображаючи продуктивність процесора та рівень його використання.
get_cpu_info()	Отримує та форматують інформацію про процесор, включаючи тип процесора і кількість ядер.
calculate_cpu_mark()	Розраховує показник продуктивності процесора на основі відсотка його використання.
create_memory_tab()	Створює вкладку "Memory" у графічному інтерфейсі, показуючи інформацію про використання пам'яті з круговою діаграмою.
get_memory_info()	Отримує та форматують інформацію про використання пам'яті.
calculate_memory_mark()	Розраховує показник продуктивності пам'яті на основі доступної пам'яті.
create_storage_tab()	Створює вкладку "Storage" у графічному інтерфейсі, відображаючи інформацію про використання сховища та продуктивність SSD.
get_storage_info()	Отримує та форматують інформацію про використання сховища для кожного розділу.

get_ssd_speed()	Отримує швидкість SSD через WMI (для Windows) для оцінки продуктивності.
calculate_ssd_mark()	Розраховує показник продуктивності для швидкості SSD.

Продовження таблиці 2.4

collect_yara_files(directory)	Збирає та повертає список файлів правил YARA (.yara або .yar) з каталогу.
create_scan_tab()	Створює вкладку "Scan" у графічному інтерфейсі для сканування на шкідливе ПЗ, з опціями для повного або вибіркового сканування.
update_scan_tab(scan_info)	Оновлює відображення вкладки сканування наданою інформацією.
start_scan()	Запускає процес сканування на шкідливе ПЗ на основі обраної опції.
stop_scan()	Зупиняє процес сканування на шкідливе ПЗ.
scan_directory(directory, rules)	Сканує каталог на наявність шкідливого ПЗ за допомогою правил YARA та відстежує прогрес.
scan_for_malware(scan_path)	Сканує вказаний шлях на наявність шкідливого ПЗ, використовуючи скомпільовані правила YARA.
scan_all_disks()	Ініціює повне сканування всіх дисків на наявність шкідливого ПЗ за допомогою правил YARA.
display_results(matches)	Відображає результати сканування у вкладці, показуючи виявлені шкідливі програми.
create_process_tab()	Створює вкладку "Processes & Users" у графічному інтерфейсі, відображаючи інформацію про процеси для кожного користувача.

get_summary()	Надає зведення використання системних ресурсів з рекомендаціями на основі навантаження на CPU та пам'ять.
create_summary_chart()	Створює і зберігає діаграму зведення використання CPU і пам'яті.
format_size(size)	Форматує заданий розмір у байтах у більш читабельний формат (наприклад, KB, MB, GB).

2.4 Архітектура програмного продукту

Діаграма використання (Use Case Diagram) - це один із видів діаграм уніфікованої мови моделювання (UML), який демонструє функціональність системи та взаємодію акторів (користувачів, систем або зовнішніх систем) з цією системою. Вона допомагає визначити основні можливості системи та показує, які дії можуть виконувати актори у контексті даної системи.[19]

Рисунок 2.1 - Діаграма використання користувача

Користувач може:

1. Ініціювати створення вкладки "Scan" (UC1), що відкриває розділ для налаштування параметрів сканування.
2. Налаштувати опції сканування (UC2), обираючи, наприклад, конкретний диск для перевірки або повне сканування.
3. Ввести шлях диска (UC3), який сканується у разі вибору конкретного диска.
4. Розпочати сканування (UC4), активуючи процес перевірки.

5. Зупинити сканування (UC5), якщо це необхідно, наприклад, у випадку помилок або за бажанням.

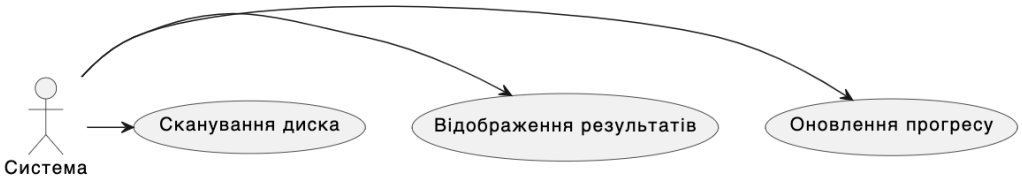


Рисунок 2.2 - Діаграма використання системи

Система автоматично:

1. Оновлює прогрес сканування (UC6), відображаючи його користувачу в реальному часі.
2. Після завершення сканування відображає результати (UC7), повідомляючи про виявлене шкідливе ПЗ або відсутність загроз.
3. Виконує процес сканування диска (UC8), перевіряючи файли і порівнюючи їх із заданими правилами для виявлення шкідливого ПЗ.

Діаграма активності (Activity Diagram) є одним із видів діаграм уніфікованої мови моделювання (UML), який використовується для візуального представлення послідовності активностей або дій, що відбуваються в системі або процесі. [20] Вона використовується для моделювання бізнес-процесів, поведінки системи та послідовності взаємодій між різними об'єктами або акторами.

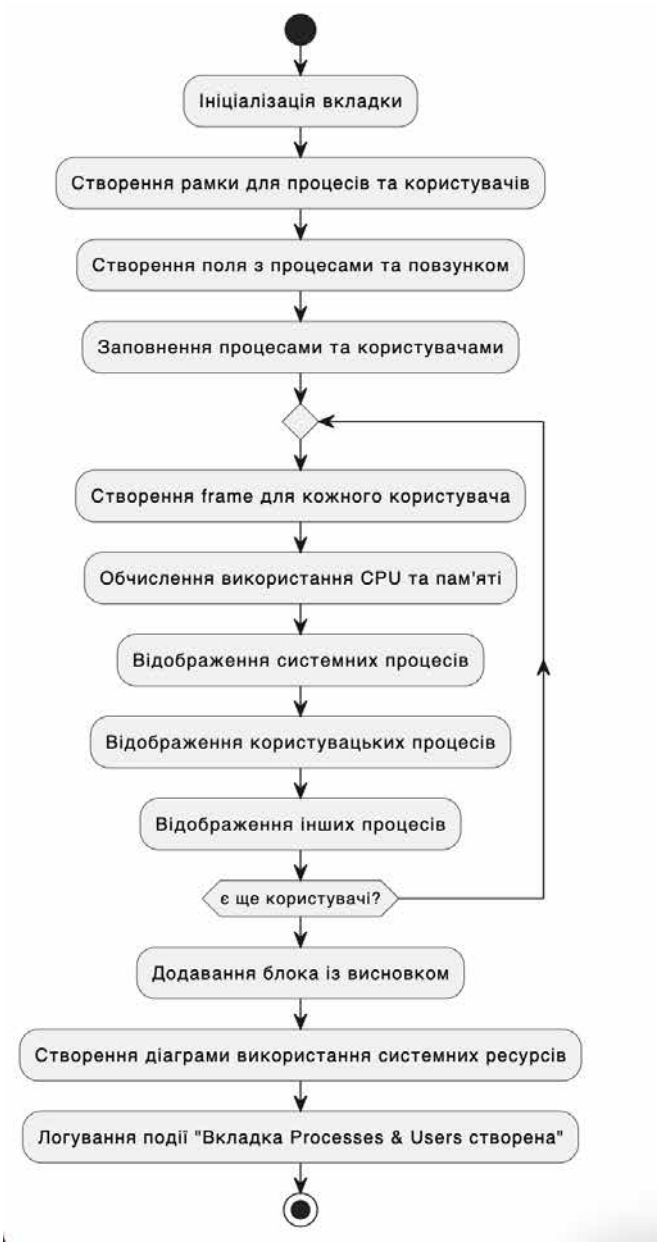


Рисунок 2.3 - Діаграма активності Processes & Users

Діаграма активності показує процес створення та наповнення вкладки "Processes & Users" у програмі для моніторингу системних ресурсів. Спочатку ініціалізується вкладка для відображення інформації про активні процеси та користувачів системи. У вкладці створюється головний блок, де відображатимуться всі дані про процеси користувачів, а також додається область з повзунком для зручної навігації, якщо інформація займає більше простору, ніж дозволяє екран. Після цього програма завантажує інформацію про процеси всіх користувачів і починає обробляти її. Для

кожного користувача формується окремий блок, де відображаються процеси, які він запустив. Розраховується використання CPU та пам'яті, що дозволяє оцінити, наскільки ресурсоємними є ці процеси. Користувацькі процеси розділяються на кілька груп: системні, користувацькі та інші. Відповідно, спершу показуються процеси, пов'язані з системою, далі – ті, що запущені безпосередньо користувачем, і завершують відображення інші процеси, які не належать до жодної з попередніх категорій.

Після обробки всіх користувачів додається підсумковий блок, де відображається загальний висновок про стан системи, зокрема щодо навантаження на ресурси. Далі генерується діаграма використання системних ресурсів, яка наочно показує загальне завантаження процесора та пам'яті. На завершення подія створення вкладки "Processes & Users" фіксується в журналі, що забезпечує моніторинг та відстеження функціонування програми. Таким чином, вкладка готова до використання і надає користувачеві повний огляд стану системних ресурсів та активності процесів.

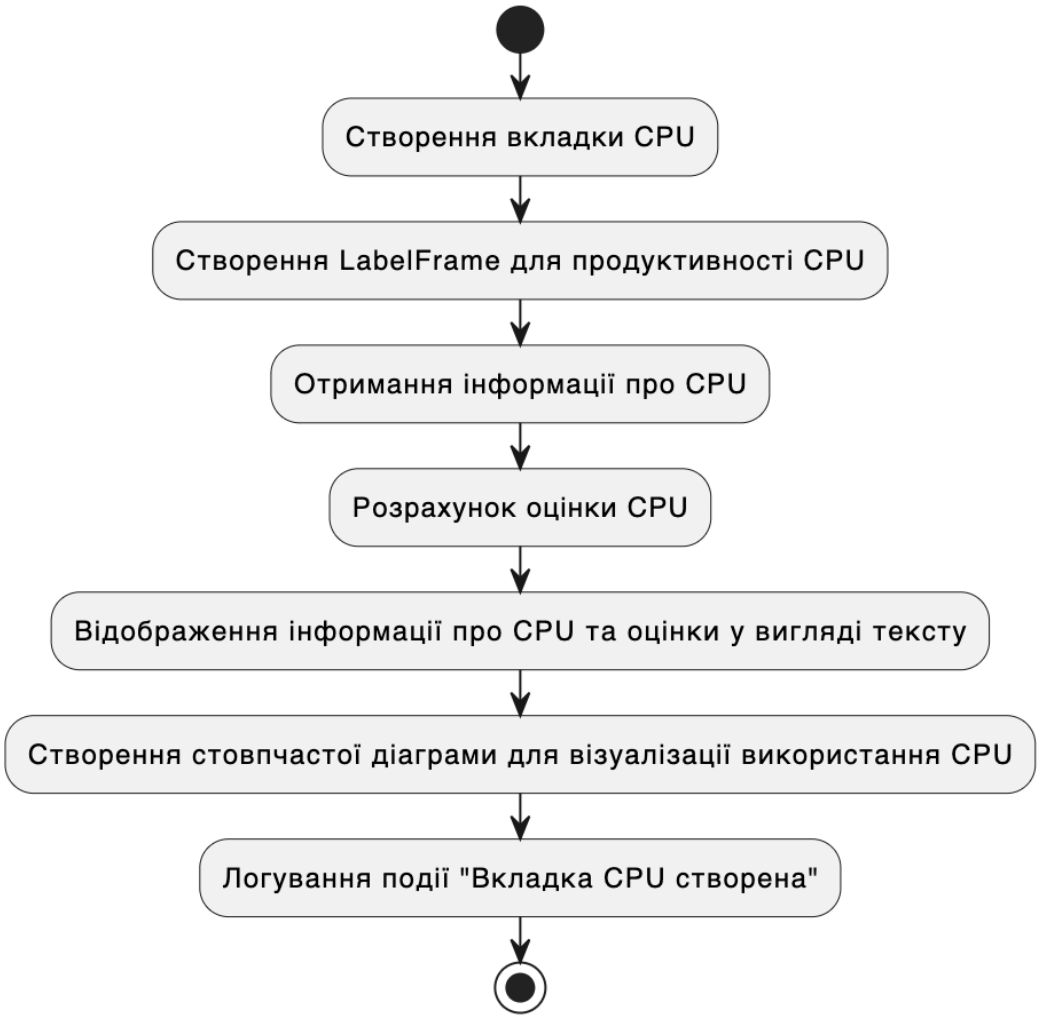


Рисунок 2.4 - Діаграма активності CPU

Діаграма активності демонструє процес створення та наповнення вкладки для відображення інформації про продуктивність CPU у програмі моніторингу системних ресурсів. Процес починається зі створення вкладки CPU, де буде розміщено усі елементи, пов'язані з продуктивністю центрального процесора. Спершу створюється основний контейнер (LabelFrame), який задає заголовок для блоку продуктивності CPU і слугує головною рамкою для всіх відображених даних. Наступний крок передбачає отримання інформації про CPU, такої як назва процесора, кількість ядер, поточне навантаження тощо. Після цього здійснюється розрахунок оцінки продуктивності CPU (CPU Mark), яка слугує індикатором загального рівня завантаження процесора.

Зібрана інформація та оцінка відображаються у вигляді тексту, що дозволяє користувачеві швидко оцінити стан CPU. Далі генерується стовпчаста діаграма, яка візуалізує рівень використання кожного ядра процесора, допомагаючи користувачеві побачити детальнішу картину продуктивності CPU в реальному часі. На завершення створення вкладки "CPU" логування цього етапу фіксується у системному журналі, що забезпечує додатковий контроль за коректною роботою програми. Таким чином, вкладка CPU пропонує користувачеві комплексну інформацію про продуктивність центрального процесора, включно з текстовим описом та графічною візуалізацією.

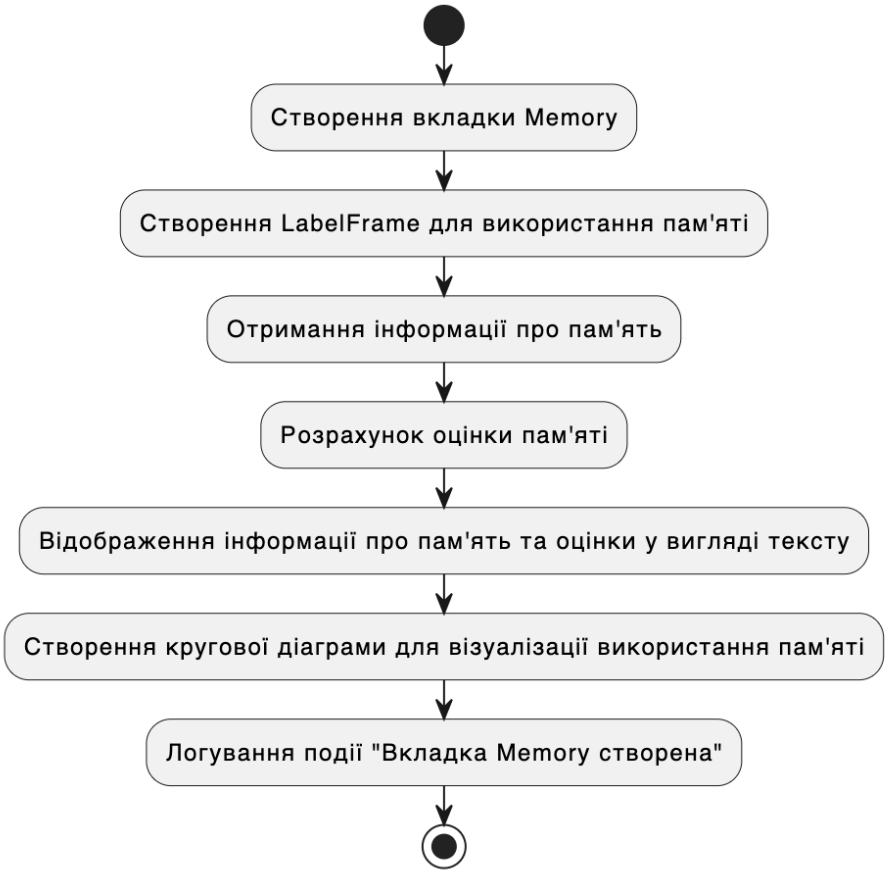


Рисунок 2.5 - Діаграма активності Memory

Діаграма ілюструє процес створення та заповнення вкладки для відображення інформації про використання пам'яті у програмі моніторингу системних ресурсів. Процес розпочинається зі створення спеціальної

вкладки "Memory", призначеної для візуалізації стану оперативної пам'яті. Першим етапом є формування контейнера (LabelFrame) для інформації про пам'ять, що задає рамку для всіх елементів, пов'язаних з використанням пам'яті. Далі програма отримує детальну інформацію про пам'ять, зокрема її загальний обсяг та поточне використання. На основі отриманих даних проводиться розрахунок оцінки пам'яті, яка надає користувачеві узагальнений індикатор рівня завантаження пам'яті. Отримані дані та оцінка відображаються текстом у вкладці, що дозволяє користувачеві швидко оцінити стан оперативної пам'яті. Додатково для зручності користувача створюється кругова діаграма, яка наочно демонструє поточне використання та доступний обсяг пам'яті. Завершальним кроком є логування події "Вкладка Memory створена" для контролю за виконанням цього етапу у системному журналі. Таким чином, вкладка "Memory" забезпечує користувача повною інформацією про оперативну пам'ять у вигляді текстового опису та графічної візуалізації.

Рисунок 2.6 - Діаграма активності Storage

Діаграма демонструє процес створення вкладки для моніторингу продуктивності сховища у системі, що дозволяє користувачу оцінювати стан та використання накопичувачів. На початку програма ініціює створення вкладки "Storage", яка буде містити всі необхідні елементи для відображення інформації про сховище. Створюється спеціальний контейнер LabelFrame, який надає структуровану рамку для подальшого заповнення інформацією про сховище. Програма збирає інформацію про наявні накопичувачі, включаючи загальний обсяг пам'яті та використання дискового простору. Потім проводиться розрахунок оцінки швидкості SSD, що забезпечує користувача інтегральним показником продуктивності накопичувача.

Зібрана інформація та оцінка відображаються текстом у вкладці, що дає змогу користувачу швидко зрозуміти загальний стан накопичувачів. Додатково створюється стовпчаста діаграма, яка візуалізує загальні розміри та використання дискового простору, що додає зручності в сприйнятті даних. Завершальним кроком є запис події "Вкладка Storage створена" у системному журналі, що допомагає відстежувати виконання кожного етапу програми. Таким чином, вкладка "Storage" надає комплексну інформацію про продуктивність накопичувачів у текстовому та графічному вигляді.

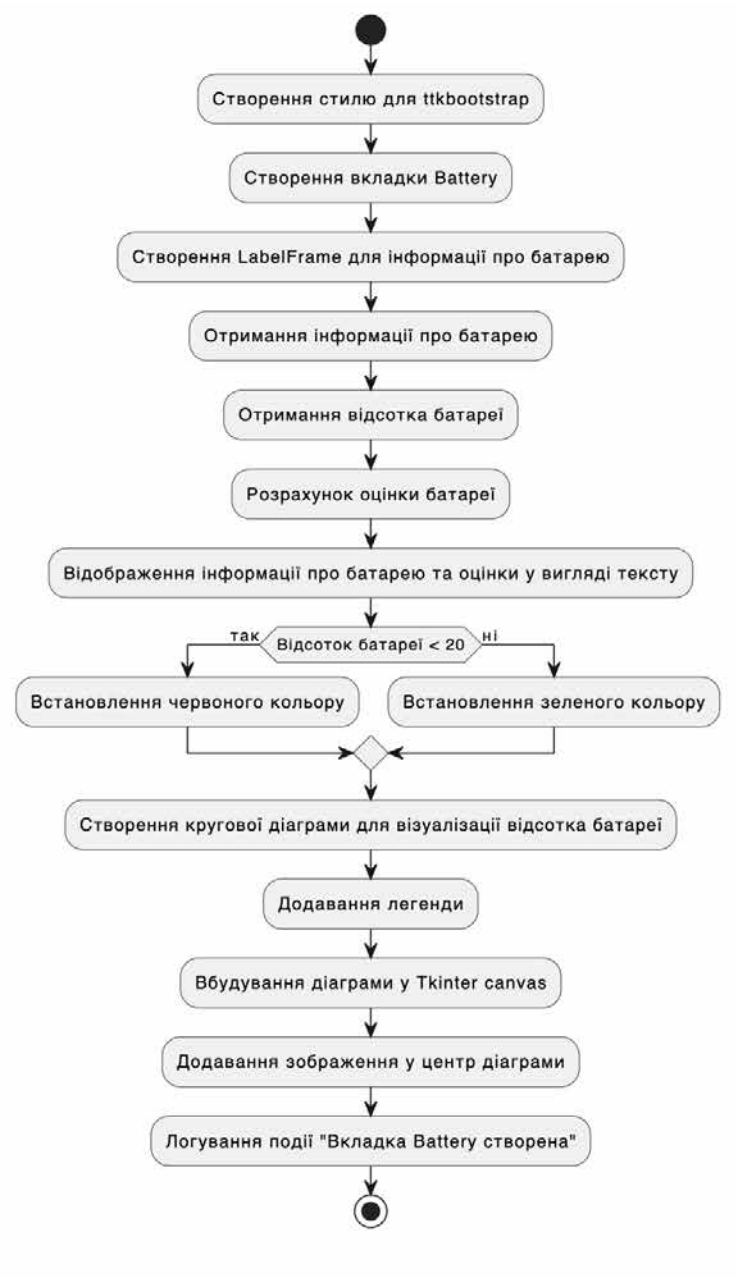


Рисунок 2.7 - Діаграма активності Battery

Діаграма описує послідовність дій при створенні вкладки "Battery" для відображення стану батареї в системі. Процес починається з налаштування стилю для використання бібліотеки ttkbootstrap, що додає естетичний вигляд елементам інтерфейсу. Далі створюється вкладка "Battery" і контейнер LabelFrame, де відображатиметься вся інформація про батарею. Програма отримує інформацію про батарею, зокрема рівень заряду, після чого виконує розрахунок інтегральної оцінки для батареї. Ця

оцінка та основна інформація виводяться у вигляді тексту, який дозволяє користувачу швидко ознайомитися з поточним станом батареї.

Залежно від рівня заряду батареї обирається колір: якщо заряд менше 20%, встановлюється червоний колір, а якщо більше – зелений. Це додає наочності і дозволяє користувачу легко оцінити стан батареї з першого погляду. Після цього програма створює кругову діаграму, що відображає рівень заряду, і додає легенду для позначення значень. Діаграма вбудовується у Tkinter canvas для інтеграції в інтерфейс програми, а в її центрі додається зображення, що підсилює візуальне сприйняття. На завершення подія "Вкладка Battery створена" записується у системний журнал для ведення обліку виконаних дій у програмі. Ця вкладка забезпечує зручне відображення інформації про батарею, використовуючи текстовий та графічний способи візуалізації для більшого комфорту користувача.

Діаграма послідовності (Sequence Diagram) є одним із видів діаграм уніфікованої мови моделювання (UML), який використовується для візуального представлення послідовності взаємодій між об'єктами або акторами в системі. [21][25] Вона демонструє, як об'єкти спілкуються між собою та взаємодіють у певному порядку для виконання певної функціональності.

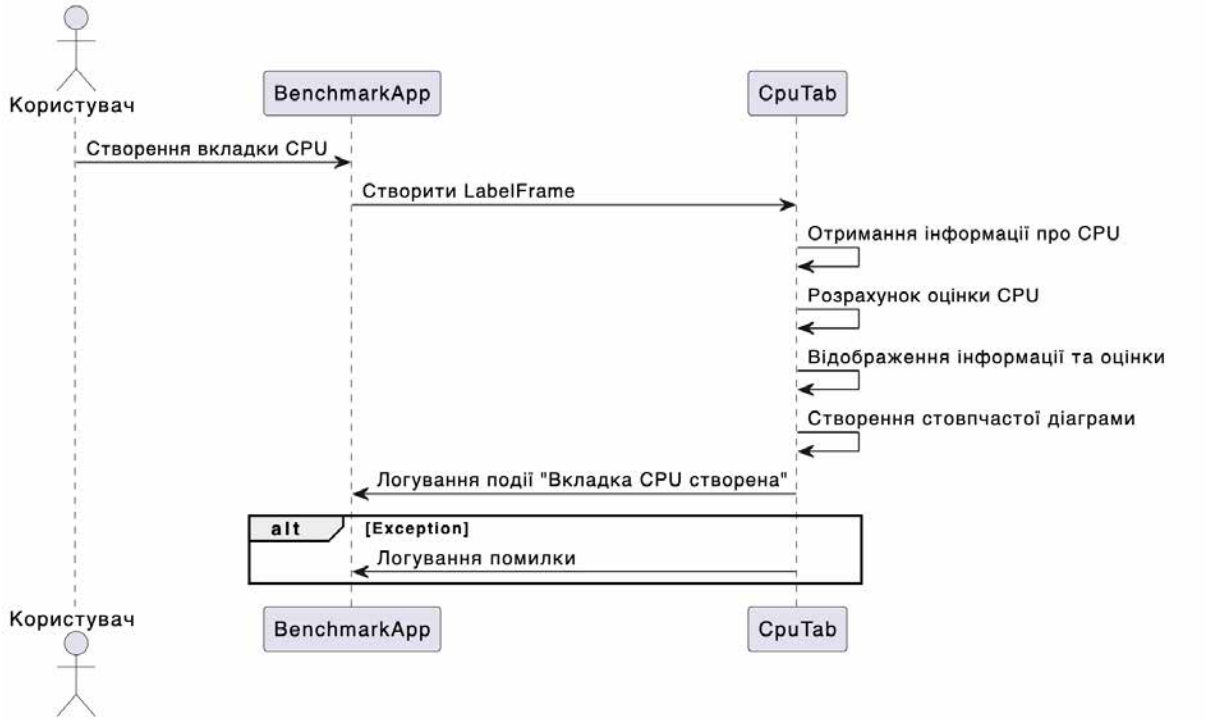


Рисунок 2.8 - Діаграма послідовності для CPU

Користувач ініціює створення вкладки CPU, надавши запит BenchmarkApp на її створення. BenchmarkApp передає завдання CpuTab, де створюється LabelFrame для відображення інформації про продуктивність CPU. CpuTab виконує отримання інформації про CPU, розраховує оцінку його продуктивності, а потім відображає як текстові дані, так і стовпчасту діаграму для візуалізації використання CPU. Після виконання завдань CpuTab передає подію BenchmarkApp для логування успішного створення вкладки "CPU". Якщо виникає виняток (Exception), CpuTab виконує логування помилки, щоб відобразити проблеми, які могли виникнути під час виконання процесу.

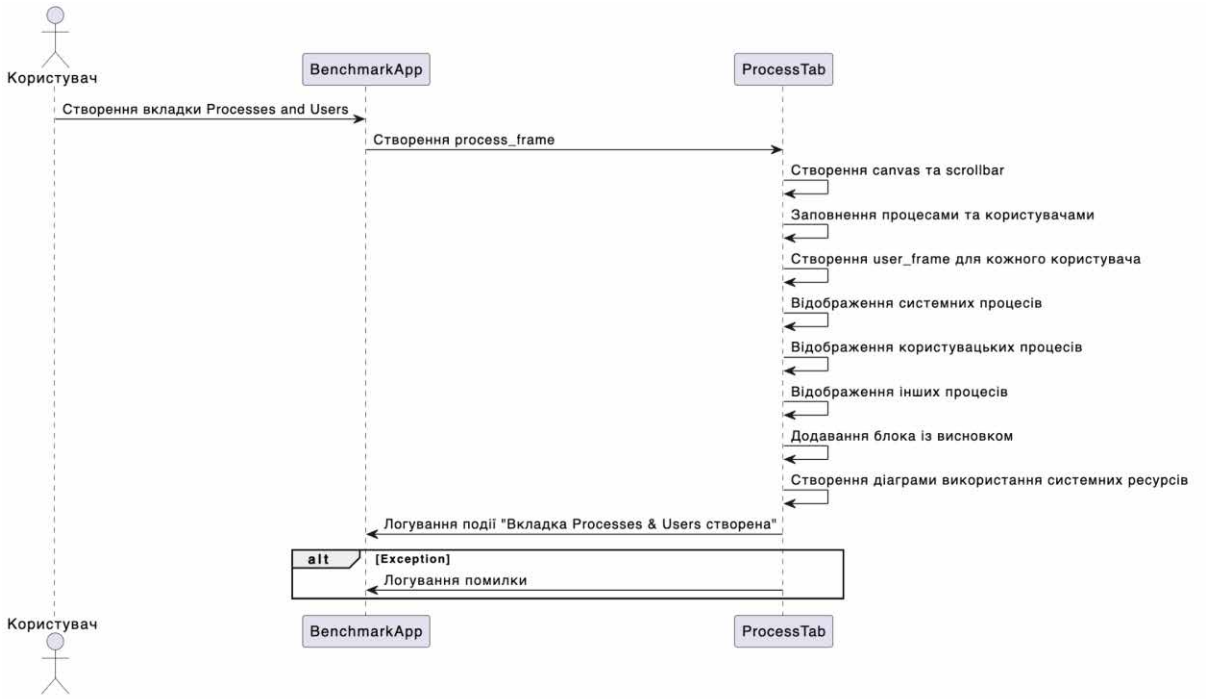


Рисунок 2.9 - Діаграма послідовності для Processes&Users

Діаграма демонструє процес створення та роботи вкладки Processes and Users в додатку BenchmarkApp, яка відображає інформацію про процеси і користувачів системи.

1. Користувач ініціює створення вкладки Processes and Users через BenchmarkApp.
2. BenchmarkApp викликає ProcessTab для створення основного компонента process_frame, де будуть відображатися процеси і користувачі.
3. ProcessTab налаштовує інтерфейс з елементами canvas та scrollbar для забезпечення зручної навігації при великій кількості процесів.
4. ProcessTab збирає та заповнює список поточних процесів та користувачів системи, додаючи кожного користувача до окремого user_frame.
5. Для кожного user_frame:
    - відображаються системні процеси, що належать користувачеві;

- відображаються користувацькі процеси, як-от веб-браузери або інші додатки;
- відображаються інші процеси, які не належать до вищезазначених категорій.

6. ProcessTab додає блок з висновками, що містить загальне навантаження на систему та рекомендації для користувача.
7. Також створюється діаграма використання системних ресурсів для візуалізації завантаженості CPU та пам’яті.
8. Після завершення створення вкладки ProcessTab логує подію "Вкладка Processes & Users створена".
9. У разі виникнення помилок, ProcessTab передає їх до BenchmarkApp для логування та подальшого аналізу.

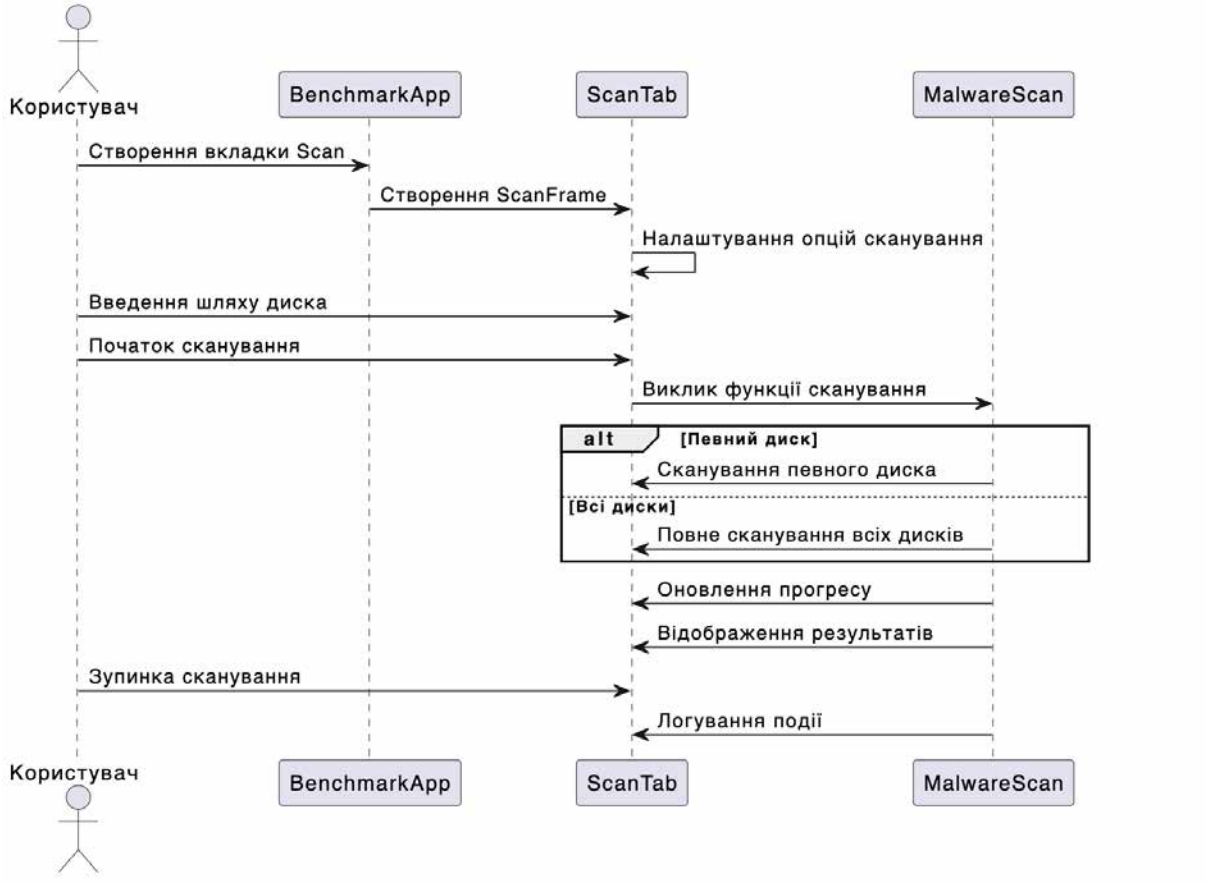


Рисунок 2.10 - Діаграма послідовності для SCAN

Діаграма демонструє процес створення та роботи вкладки Scan в додатку BenchmarkApp, яка відповідає за сканування системи на наявність шкідливих програм.

1. Користувач ініціює створення вкладки Scan через BenchmarkApp.
2. BenchmarkApp викликає ScanTab для створення основної рамки ScanFrame, де будуть доступні опції сканування.
3. ScanTab налаштовує параметри сканування, дозволяючи користувачеві обрати конкретний диск або всі диски для сканування.
4. Користувач:
    - вводить шлях до диска для вибіркового сканування;
    - починає процес сканування.
5. ScanTab викликає MalwareScan для запуску функції сканування.
6. У залежності від обраної опції сканування, MalwareScan:
    - виконує сканування певного диска, якщо було вказано конкретний шлях;
    - виконує повне сканування всіх доступних дисків, якщо не було вказано конкретного шляху.
7. Під час процесу MalwareScan оновлює прогрес сканування в ScanTab і відображає результати після завершення сканування.
8. Користувач може зупинити сканування у будь-який момент, що відображається у вкладці.
9. В кінці MalwareScan логує подію, щоб забезпечити запис усіх дій для подальшого аналізу.

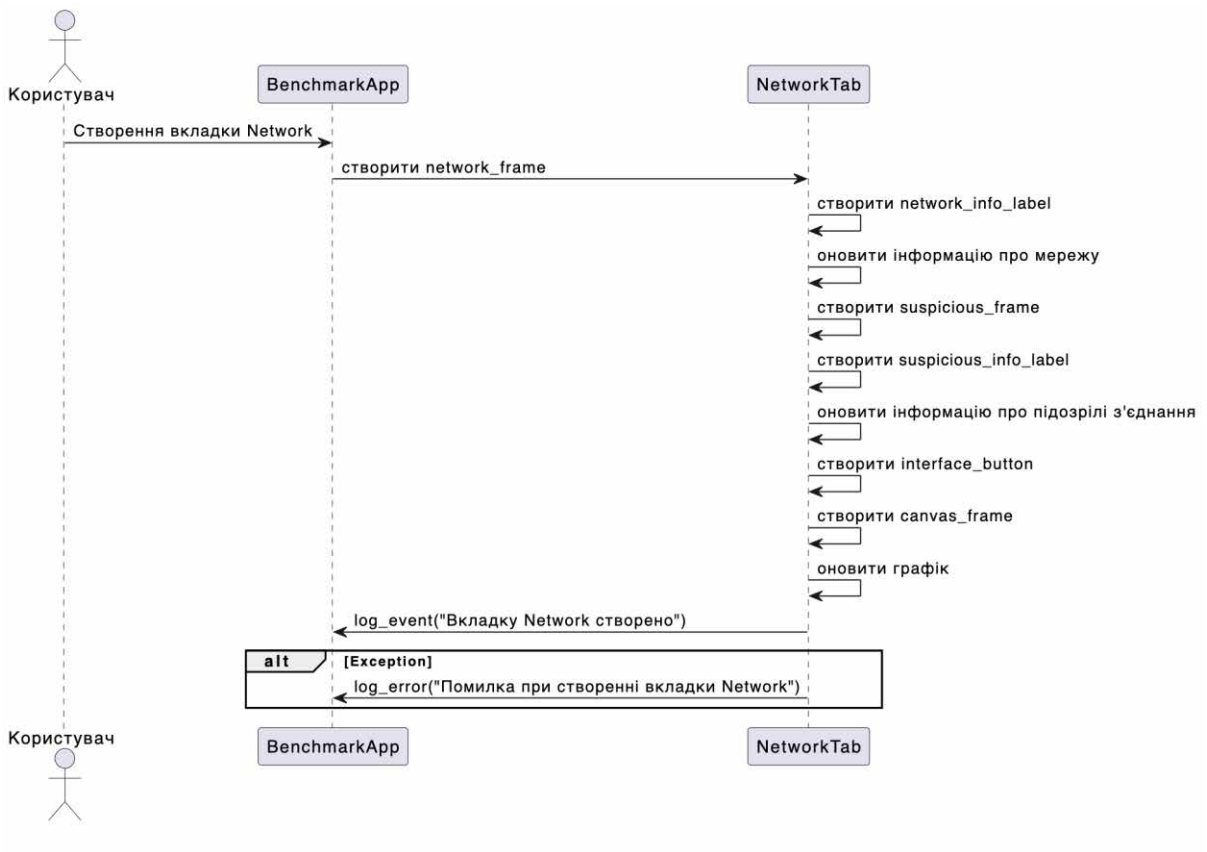


Рисунок 2.11 - Діаграма послідовності для Network

Діаграма показує процес створення вкладки Network у додатку BenchmarkApp з усіма основними компонентами та діями, необхідними для відображення інформації про мережу.

1. Користувач ініціює створення вкладки Network через BenchmarkApp.
2. BenchmarkApp викликає NetworkTab для:
    - створення network_frame — основної рамки, де буде відображено всю інформацію про мережу;
    - додавання network_info_label для показу поточного стану мережі;
    - оновлення даних про мережу, щоб користувач бачив актуальну інформацію;
    - створення suspicious_frame та suspicious_info_label для виявлення підозрілих з’єднань;

- додавання interface_button для швидкого перегляду інформації про мережеві інтерфейси;
- додавання canvas_frame для візуалізації мережевої активності у вигляді графіка;
- оновлення графіка для відображення поточного використання мережі.

3. Завершення процесу включає логування події про створення вкладки Network.
4. У випадку виникнення винятку NetworkTab логуватиме помилку через BenchmarkApp.

Діаграма розгортання (Deployment Diagram) – це тип UML-діаграми, яка відображає фізичне розташування компонентів програмної системи на апаратному обладнанні. Вона ілюструє, як програмні модулі (компоненти) розгорнуті на різних вузлах (серверах, клієнтах, пристроях), показуючи зв’язки між ними та їхнє розміщення в реальному середовищі.

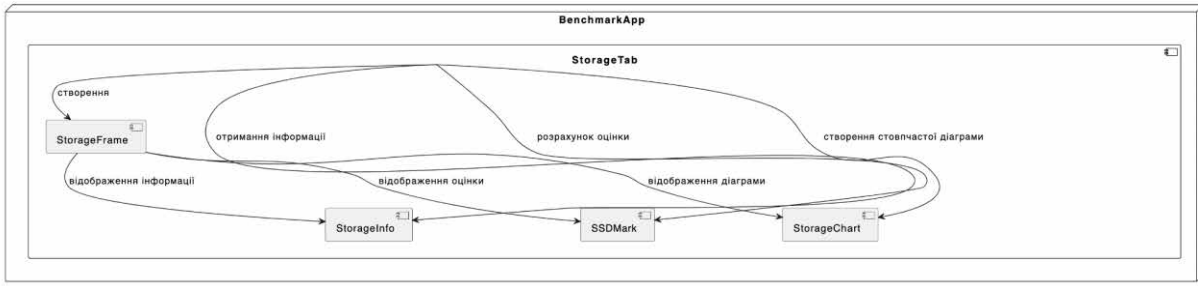


Рисунок 2.12 - Діаграма розгортання для Storage

StorageTab – основний компонент, що ініціює процес і забезпечує всі етапи для відображення інформації про сховище. StorageFrame – рамка, що містить усі відомості про вкладку та об’єднує інші елементи у вкладці Storage. Вона використовується для відображення даних у вигляді тексту і діаграм. StorageInfo – забезпечує отримання та відображення основної інформації про сховище, такої як загальний і використаний простір. SSDMark – розраховує продуктивність SSD на основі показників

швидкості, а потім відображає цю оцінку користувачеві. StorageChart – створює стовпчасту діаграму, що візуалізує загальний і використаний простір кожного розділу диска для легкого сприйняття даних.

Діаграма класів (Class Diagram) – це тип UML-діаграми, яка відображає структуру системи, показуючи класи, їх атрибути, методи, а також відносини між ними (наслідування, асоціації, агрегації, композиції). Вона ілюструє статичну структуру системи та слугує основою для моделювання об’єктно-орієнтованого дизайну.

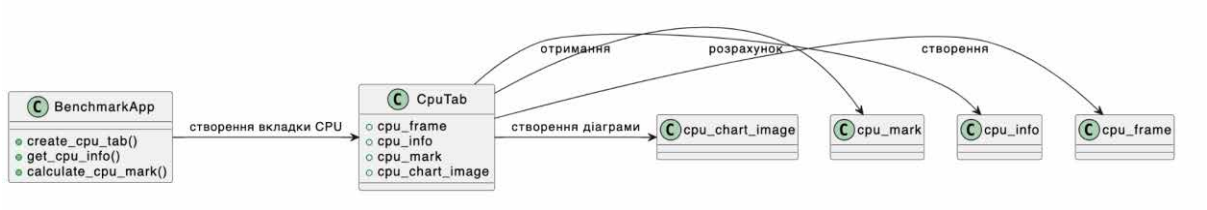


Рисунок 2.13 - Діаграма класів

Діаграма класів демонструє структуру створення та функціонування вкладки CPU в програмі BenchmarkApp, відображаючи методи й об'єкти, необхідні для цієї вкладки.

BenchmarkApp має методи create_cpu_tab, get_cpu_info, та calculate_cpu_mark. Ці методи відповідають за створення вкладки CPU, отримання інформації про процесор і розрахунок оцінки продуктивності. BenchmarkApp взаємодіє з класом CpuTab, створюючи його для відображення даних про процесор.

У класі CpuTab:

- cpu_frame представляє рамку, що містить інформацію про продуктивність CPU.
- cpu_info отримує дані про процесор, такі як назва, кількість ядер тощо.
- cpu_mark відповідає за розрахунок і відображення оцінки продуктивності CPU.

- cpu_chart_image створює стовпчасту діаграму, що візуалізує використання CPU.

Діаграма компонентів (Component Diagram) – це тип UML-діаграми, який показує структуру та взаємозв’язки компонентів системи на високому рівні абстракції. Кожен компонент представляє окремий модуль або частину системи, які можуть бути фізично реалізовані як бібліотеки, файли, модулі чи пакети. Діаграма компонентів демонструє, як різні частини системи взаємодіють через інтерфейси, відображаючи залежності між компонентами.

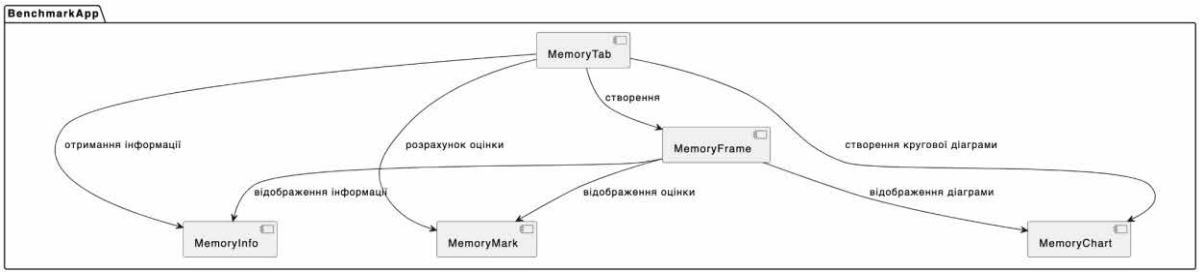


Рисунок 2.14 - Діаграма компонентів

MemoryTab є головним елементом вкладки, який взаємодіє з іншими компонентами для отримання інформації, розрахунку оцінок та візуалізації даних про використання пам'яті. MemoryFrame — рамка, що організовує відображення даних у вкладці, включаючи інформацію про пам'ять, оцінку продуктивності та діаграму. Вона містить компоненти для відображення інформації, оцінки та діаграми. MemoryInfo — модуль, що відповідає за отримання інформації про використання пам'яті, такої як загальний обсяг та використання ресурсів. MemoryTab взаємодіє з MemoryInfo для отримання необхідної інформації. MemoryMark — компонент, що виконує розрахунок оцінки використання пам'яті на основі отриманих даних, що дозволяє користувачам бачити оцінку продуктивності пам'яті. MemoryChart — компонент, який створює кругову діаграму для візуалізації використання пам'яті, допомагаючи користувачам швидко зрозуміти поточний стан ресурсів пам'яті.

Висновки до розділу 2

Основні технології, які були використані, включають мову програмування Python, фреймворк Tkinter для створення графічного інтерфейсу користувача, бібліотеку Psutil для отримання інформації про систему, бібліотеку Platform для отримання інформації про платформу, на якій працює програма, а також PyArmor для захисту вихідного коду від несанкціонованого доступу, Cryptography.hazmat для реалізації криптографічних операцій і захисту даних та YARA для виявлення і класифікації потенційно шкідливих програм за допомогою шаблонів аналізу.

В розділі також було проведено розробку вимог до програмного продукту. Функціональні вимоги визначають, які функції повинен виконувати програмний продукт, наприклад, збір інформації про систему, відображення цієї інформації у графічному інтерфейсі тощо. Нефункціональні вимоги визначають вимоги до продукту, які не стосуються його функціональності, наприклад, продукт повинен бути легким у використанні, забезпечувати швидку відповідь тощо. Опис методів включає детальний опис алгоритмів та процедур, що використовуються у програмному продукті для досягнення поставлених функціональних вимог. Архітектура програмного продукту була представлена за допомогою діаграми використання, діаграми активності та діаграми послідовності, розгортання, компонентів та класів.

РОЗДІЛ 3. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Методика тестування

Тестування програмного забезпечення є важливим етапом у розробці, що дозволяє переконатися в правильності роботи програми та відповідності її функцій заявленим вимогам. Для програми BenchmarkApp було розроблено тестовий план, спрямований на перевірку кожної з її основних функціональних частин, таких як моніторинг системних ресурсів, аналіз активності процесів, сканування на шкідливе ПЗ, перевірка безпеки та аналіз журналів подій. Метою тестування є виявлення можливих помилок у функціонуванні компонентів, оцінка стабільності роботи додатку, а також забезпечення належної якості кінцевого продукту. За допомогою даного тестування можна також впевнитись у відповідності програми очікуванням користувача та вимогам до функціональності. Таблиця 3.1 детально показує кожен із проведених тестів із зазначенням очікуваних та фактичних результатів, що дозволяє оцінити якість та ефективність розробленого програмного забезпечення.

Таблиця 3.1 - План тестування

ID тесту	**Опис тесту**	**Очікуваний результат**	**Фактичний результат**
1	Запуск програми	Програма відкривається, основне вікно і всі вкладки завантажуються коректно	Програма запустилась успішно, всі вкладки присутні
2	Вкладка "Battery"	Відображається інформація про батарею та діаграма у вигляді кругової діаграми, якщо статус батареї доступний	Інформація про батарею відображена коректно з діаграмою

3	Вкладка "CPU Performance"	Відображаються деталі CPU, використання та стовпчаста діаграма для кожного ядра	Відображено інформацію про CPU та діаграму використання
4	Вкладка "Memory Usage"	Відображається використання пам'яті, деталі та кругова діаграма з використаною та доступною пам'яттю	Відображено деталі пам'яті з правильною діаграмою
5	Вкладка "Storage Performance"	Відображається використання сховища, швидкість SSD та стовпчаста діаграма для використання сховища	Інформація про сховище та діаграма використання відображені
6	Вкладка "Security" - Генерація RSA ключів	Приватний і публічний ключі створюються та зберігаються у файли	Ключі успішно створені та збережені
7	Вкладка "Security" - Підпис файлу	Обраний файл підписується, виводиться результат успішного підпису	Файл успішно підписано
8	Вкладка "Security" - Перевірка підпису	Перевірка підпису файлу проходить успішно	Підпис перевірено коректно
9	Вкладка "Security" - Обфускація коду	Обраний файл коду обфускується без помилок	Код обфусковано успішно
10	Вкладка "Network Monitoring" - Відображення інформації про мережеві інтерфейси	Відображаються активні мережеві інтерфейси та IP-адреси разом із підозрілими з'єднаннями	Коректно відображено деталі мережевих інтерфейсів
11	Вкладка "Logs" - Перегляд логів	Відкривається вікно, що відображає всі останні логи	Логи відображені успішно
12	Вкладка "Logs" - Перегляд аномалій	Відображаються лише записи з попередженнями та помилками	Аномалії відображені коректно
13	Вкладка "Logs" - Очищення логів	Усі записи з файлу логів видаляються	Логи успішно очищені

14	Вкладка "Scan" - Початок сканування на шкідливе ПЗ для певного диска	Сканування розпочинається для вибраного диска; оновлення прогресу та результати відображаються наприкінці	Сканування розпочато, прогрес відображається
15	Вкладка "Scan" - Повне сканування	Розпочинається повне сканування всіх доступних дисків; періодично оновлюється прогрес, результати відображаються наприкінці	Повне сканування виконано, результати відображено
16	Вкладка "Processes & Users" - Відображення користувацьких та системних процесів	Відображаються всі користувацькі та системні процеси, розподілені за користувачем та типом процесу, разом із зведенням	Всі процеси та зведення відображено
17	Вкладка "Processes & Users" - Діаграма використання ресурсів	Відображається стовпчаста діаграма для відсотків використання CPU та пам'яті	Діаграма використання ресурсів відображена коректно
18	Обробка помилок - Спроба доступу до неіснуючого диска	Система відображає відповідне повідомлення про помилку та не завершується аварійно	Відображено повідомлення про помилку без завершення роботи
19	Обробка помилок - Неправильний файл з правилами YARA	Помилка записується в лог, система припиняє виконання з відповідним повідомленням	Помилка записана в лог і виконання припинено
20	Вкладка "Security" - Генерація файлів ключів без належних прав доступу	Відображається повідомлення про відсутність прав і записується помилка	Повідомлення та лог про помилку доступу відображено коректно

3.2 Проведення тестування

Видимі всі вкладки програми: Battery, CPU, Memory, Storage, Scan, Processes & Users, Security, Network і Logs. Це дозволяє користувачеві швидко переходити між основними функціями програми.

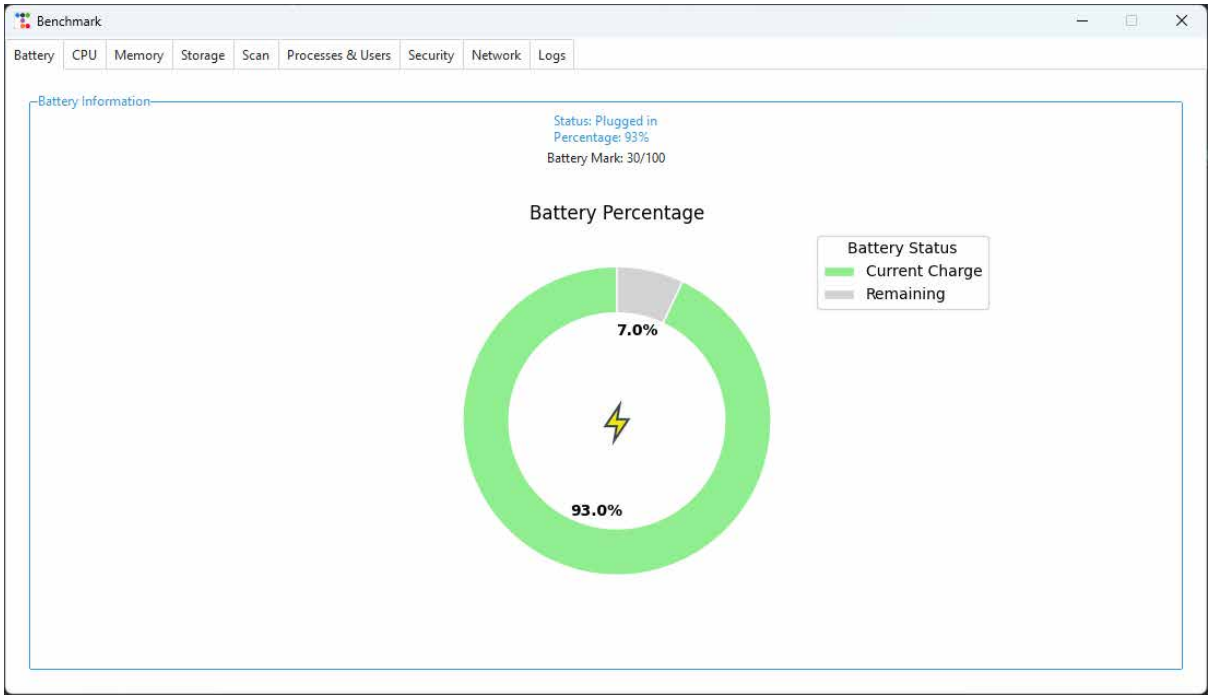


Рисунок 3.1 - Вкладка "Battery"

Рис.3.1 демонструє інформацію про батарею пристрою. Показано рівень заряду батареї у відсотках, оцінка батареї, а також кругова діаграма, що візуалізує рівень заряду, що допомагає користувачеві швидко оцінити поточний стан батареї.

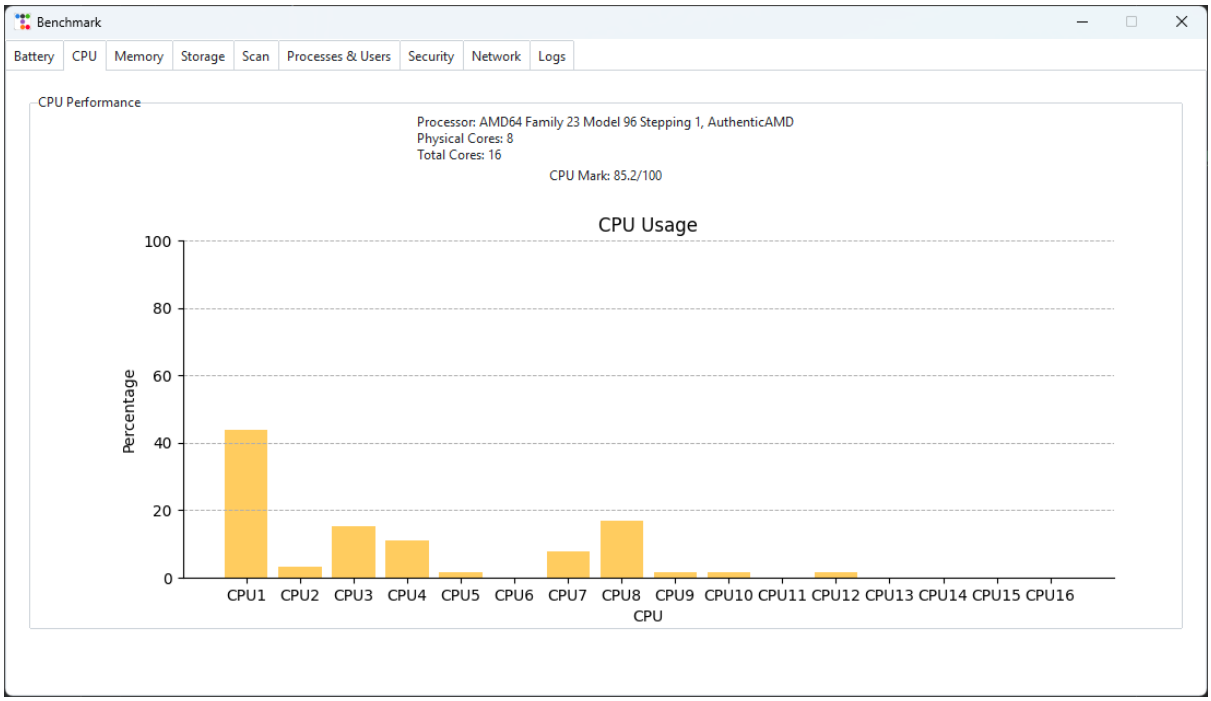


Рисунок 3.2 - Вкладка "CPU"

Рис.3.2 показує вкладку CPU з текстовою інформацією про процесор (наприклад, кількість ядер та модель CPU), оцінку продуктивності та стовпчасту діаграму, яка візуалізує використання кожного ядра процесора. Дана інформація дає змогу оцінити навантаження на процесор.

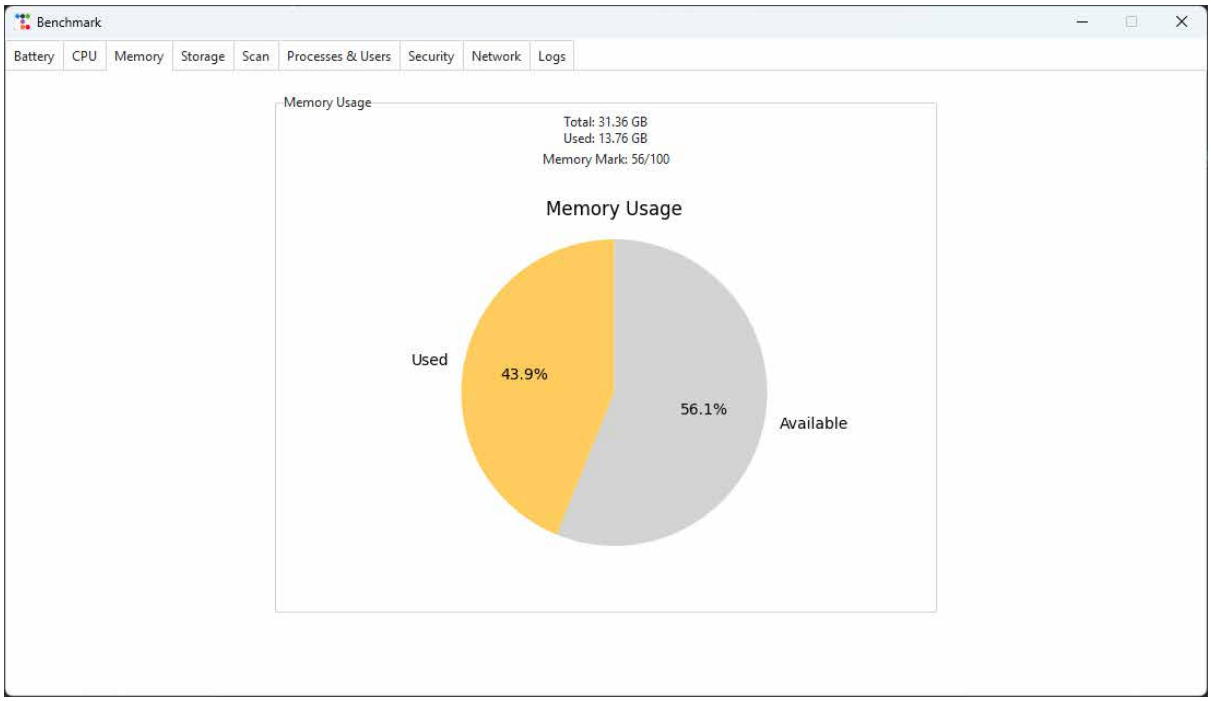


Рисунок 3.3 - Вкладка "Memory"

На рис.3.3 зображено інформацію про використання пам’яті: загальний обсяг пам’яті, її поточне використання та оцінку пам'яті. Додана кругова діаграма, що показує співвідношення використаної та доступної пам’яті, дозволяє швидко оцінити стан пам'яті.

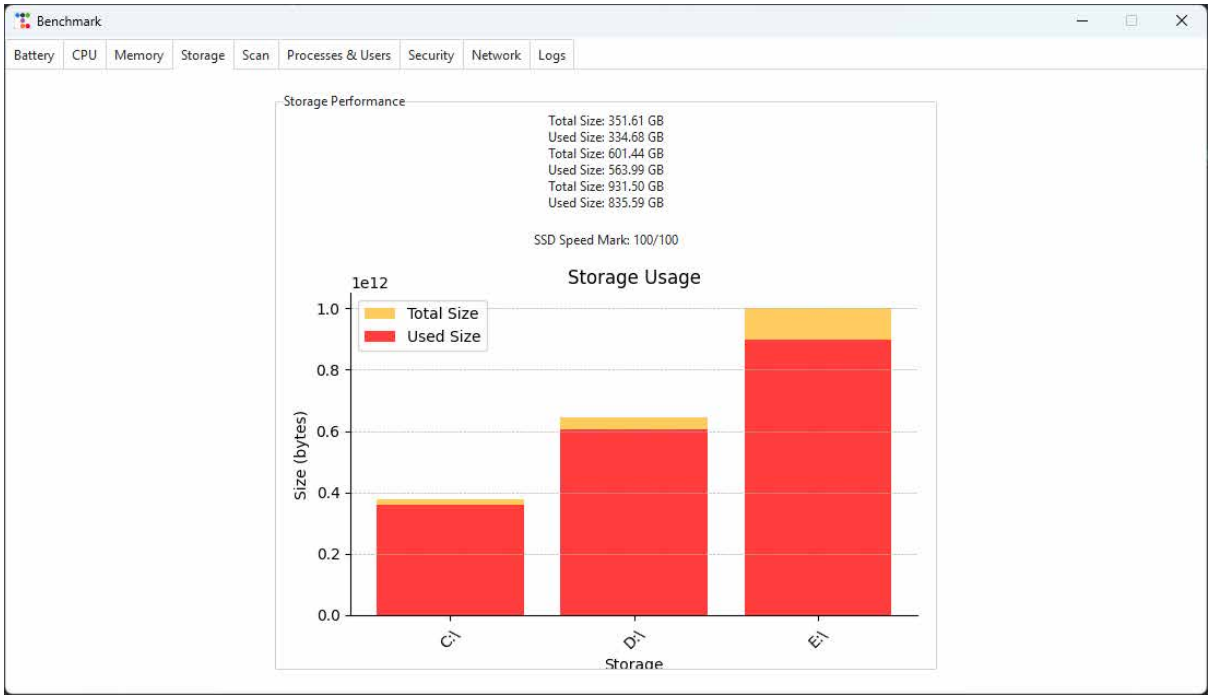


Рисунок 3.4 - Вкладка "Storage"

Рис.3.4 відображає інформацію про дисковий простір, включно із загальним розміром і поточним використанням кожного розділу. Додано стовпчасту діаграму для візуалізації обсягу використаного простору, що дозволяє оцінити залишковий простір на пристрої.

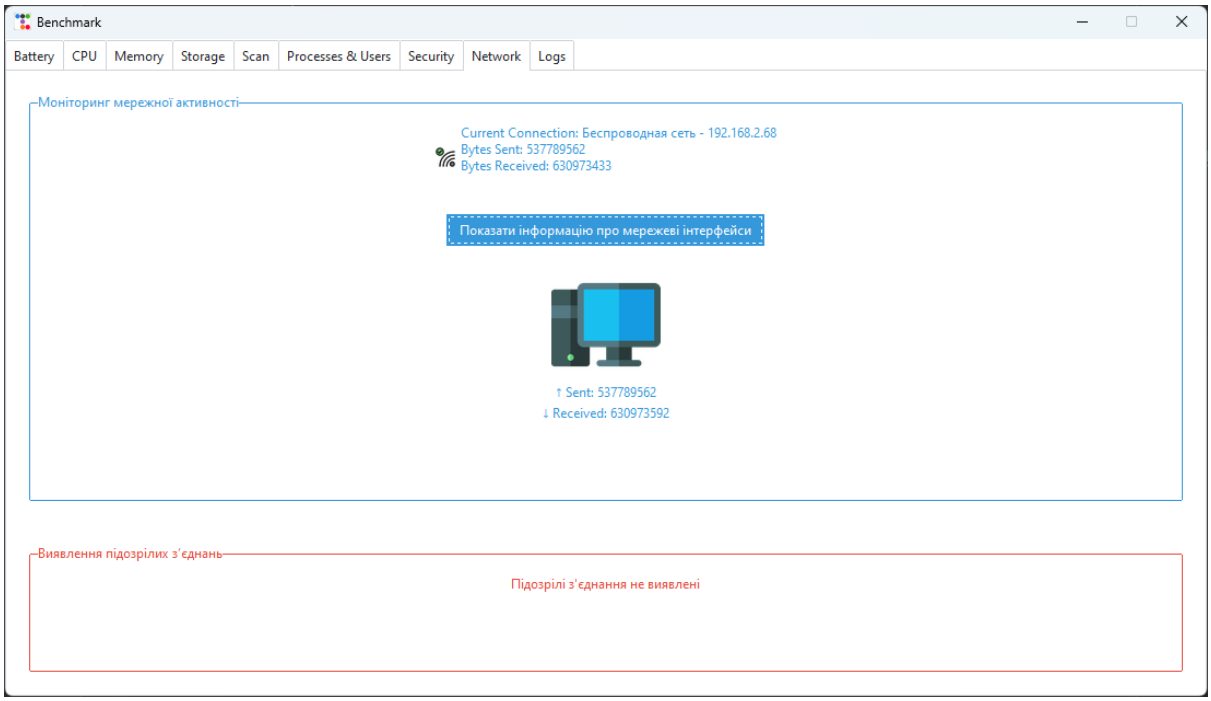


Рисунок 3.5 - Вкладка "Network"

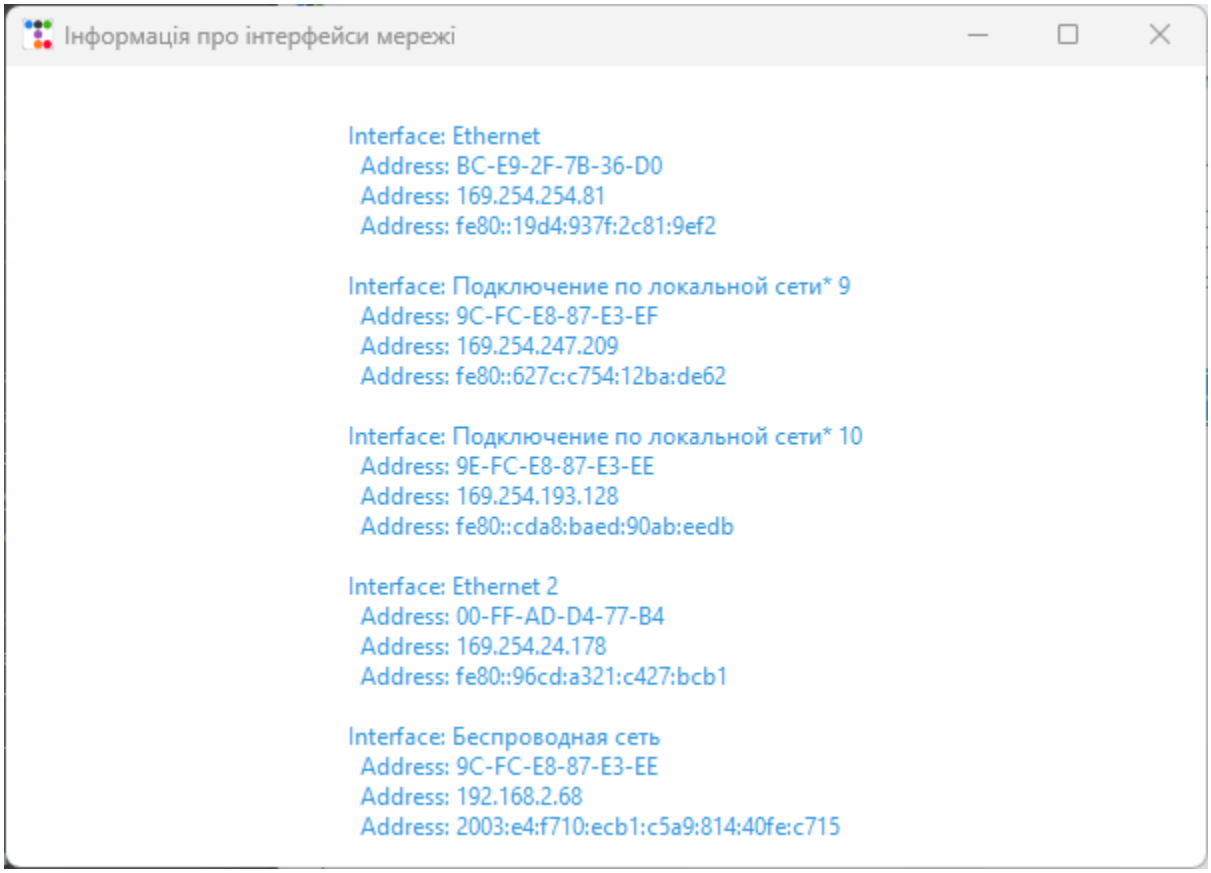


Рисунок 3.6 - Деталі "Network"

Рис.3.5 та рис.3.6 показує інформацію про мережеве підключення та підозрілих з’єднань. Інформація про поточне підключення, тип з’єднання (Ethernet або Wi-Fi) і кількість надісланих/отриманих байтів допомагає користувачеві стежити за активністю мережі.

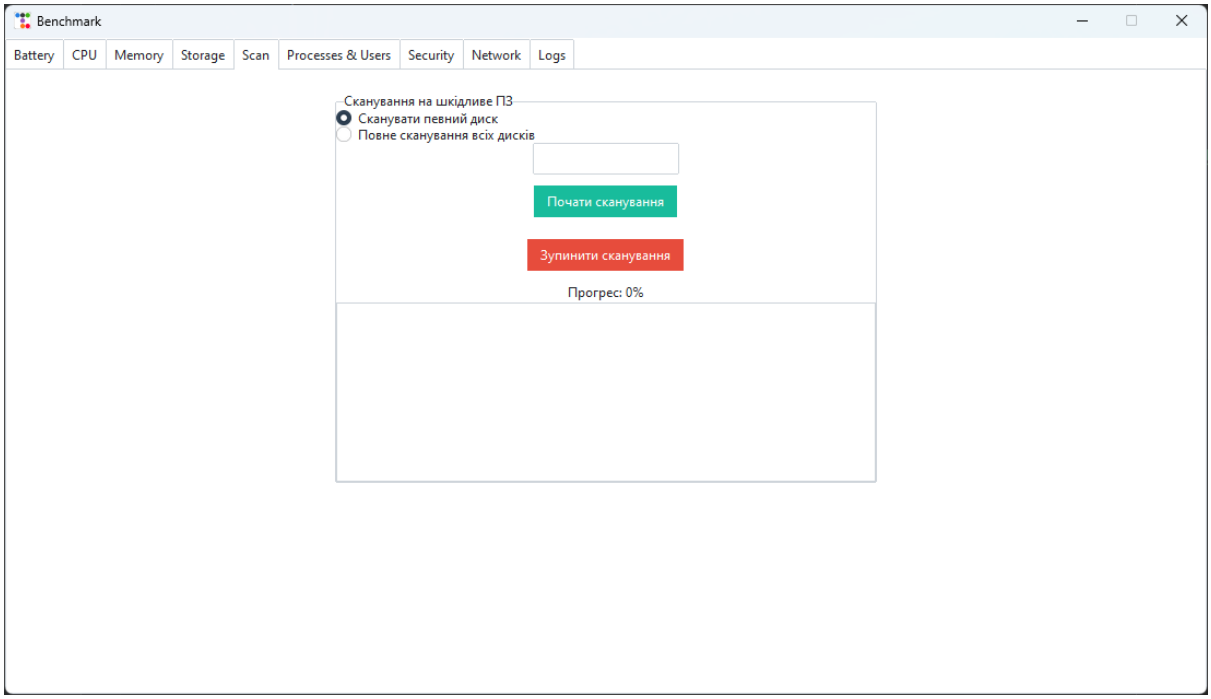


Рисунок 3.7 - Вкладка "Scan"

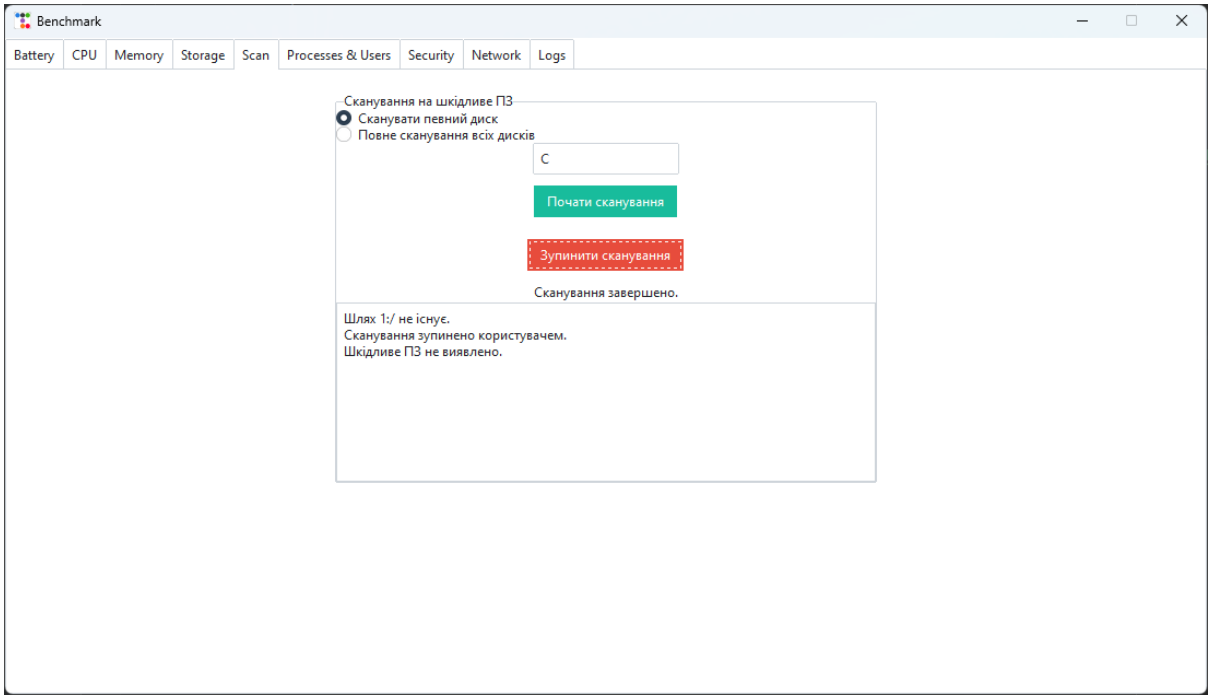


Рисунок 3.8 - Результати "Scan"

Рис.3.7 та рис.3.8 демонструє процес сканування на шкідливе ПЗ, зокрема опції вибору типу сканування, прогрес виконання, та результати сканування. Інформація відображає, чи були виявлені шкідливі файли, та загальну ефективність функції.

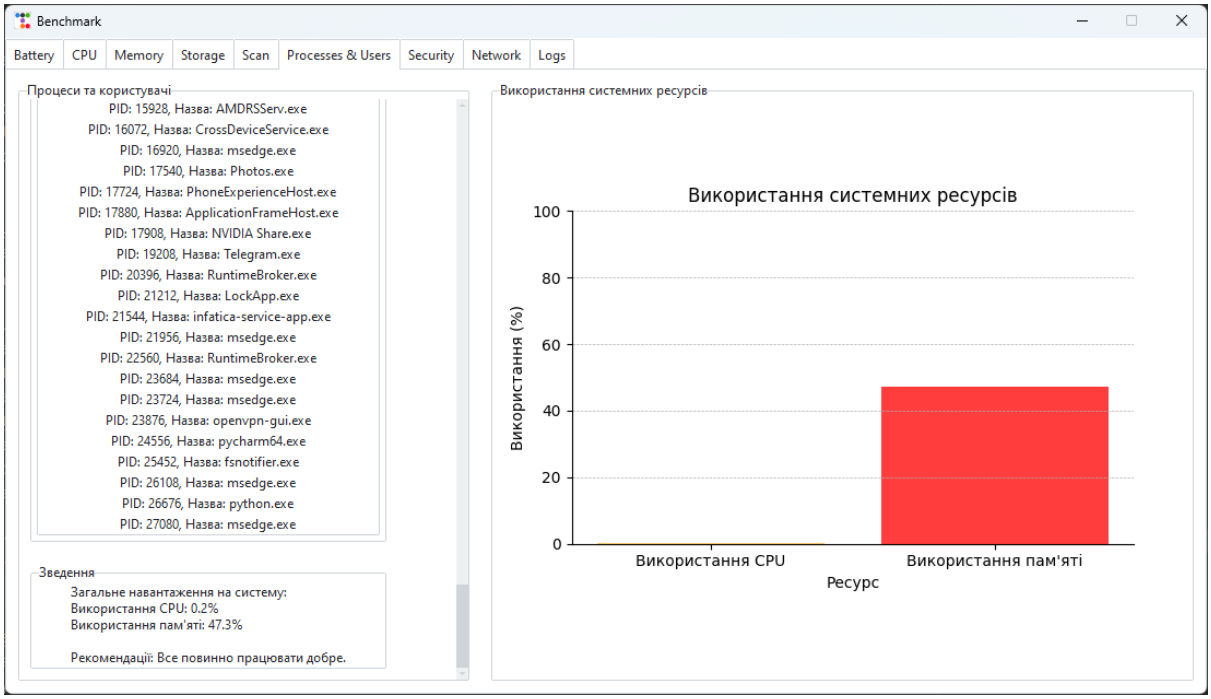


Рисунок 3.9 - Вкладка "Processes & Users"

На рис.3.9 відображено список активних процесів, розділених на системні, користувацькі та інші. Інформація включає кількість процесів, використання CPU та пам'яті кожним користувачем, а також підсумкові дані про навантаження на систему. Це дозволяє користувачеві стежити за активними процесами.

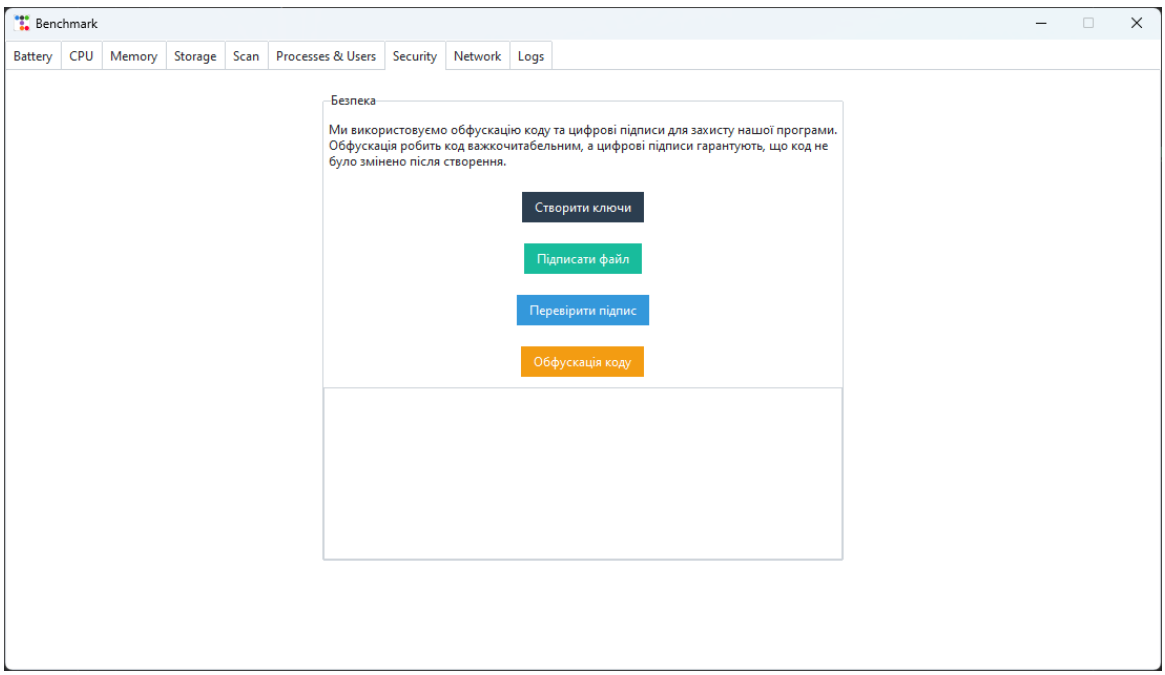


Рисунок 3.10 - Вкладка "Security"

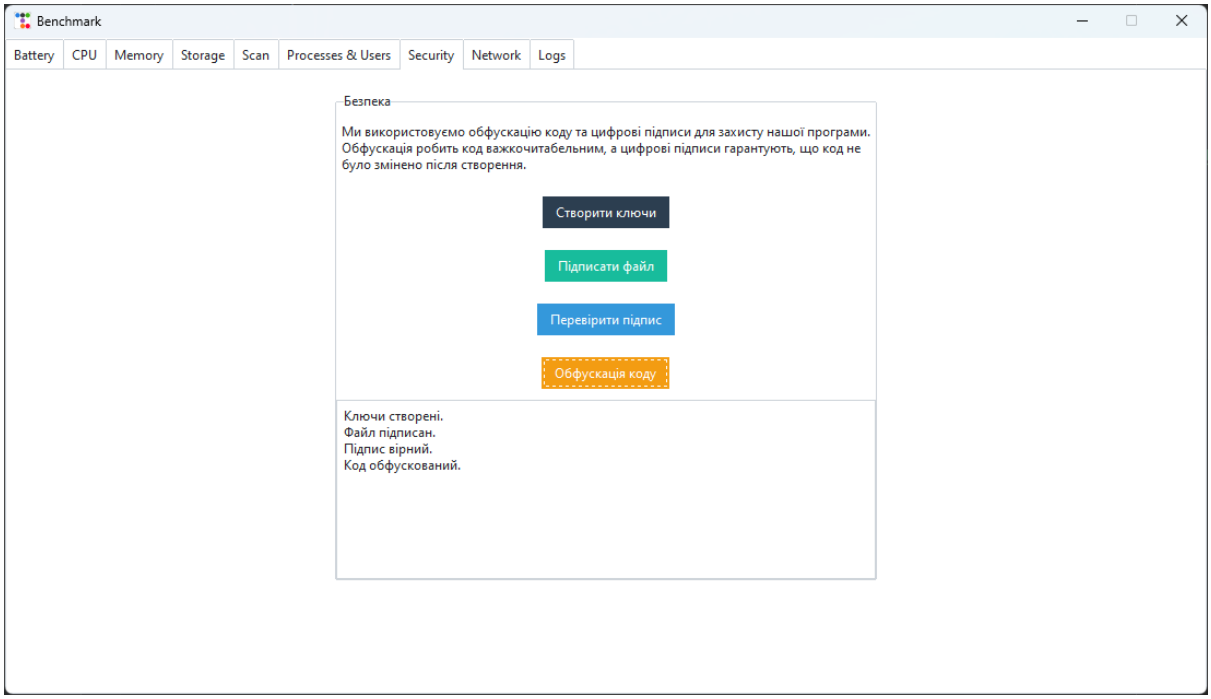


Рисунок 3.11 - Результати аналізу "Security"

Рис.3.10 та рис.3.11 демонструє функціонал вкладки Security, зокрема генерацію ключів, підписування файлів, перевірку підпису та обфускацію коду. Це важливі функції для захисту програми від несанкціонованого доступу та маніпуляцій з кодом.

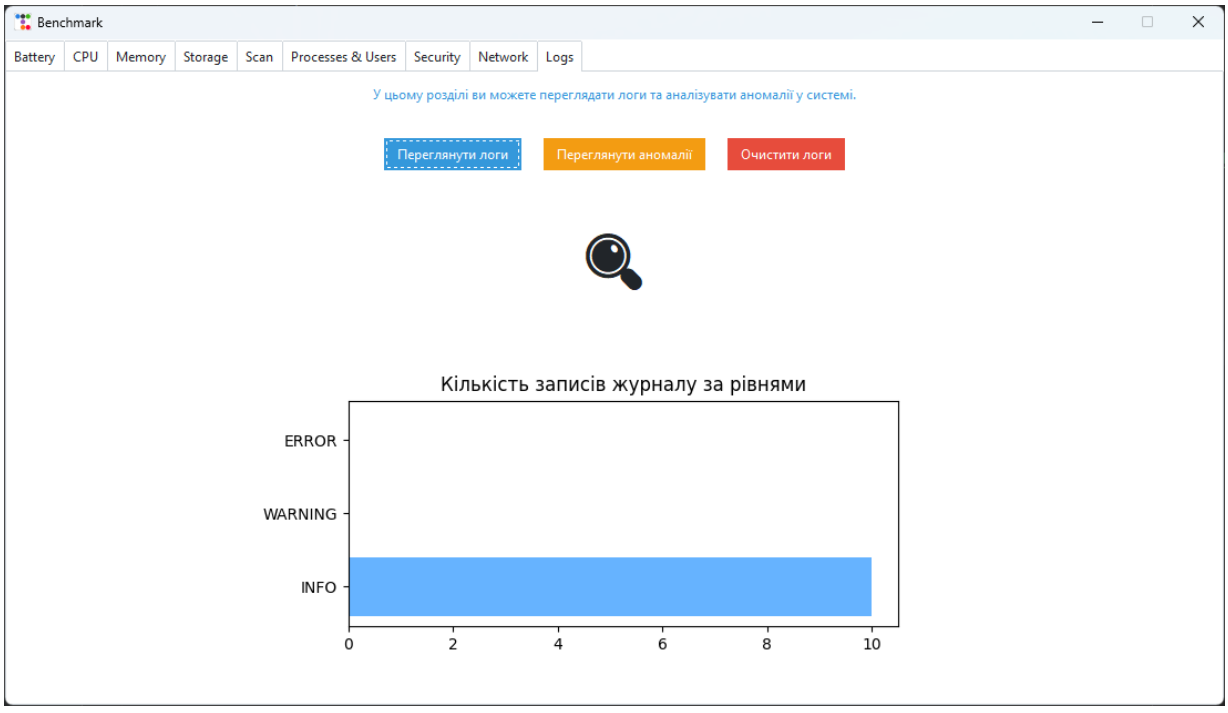


Рисунок 3.12 - Вкладка "Logs"

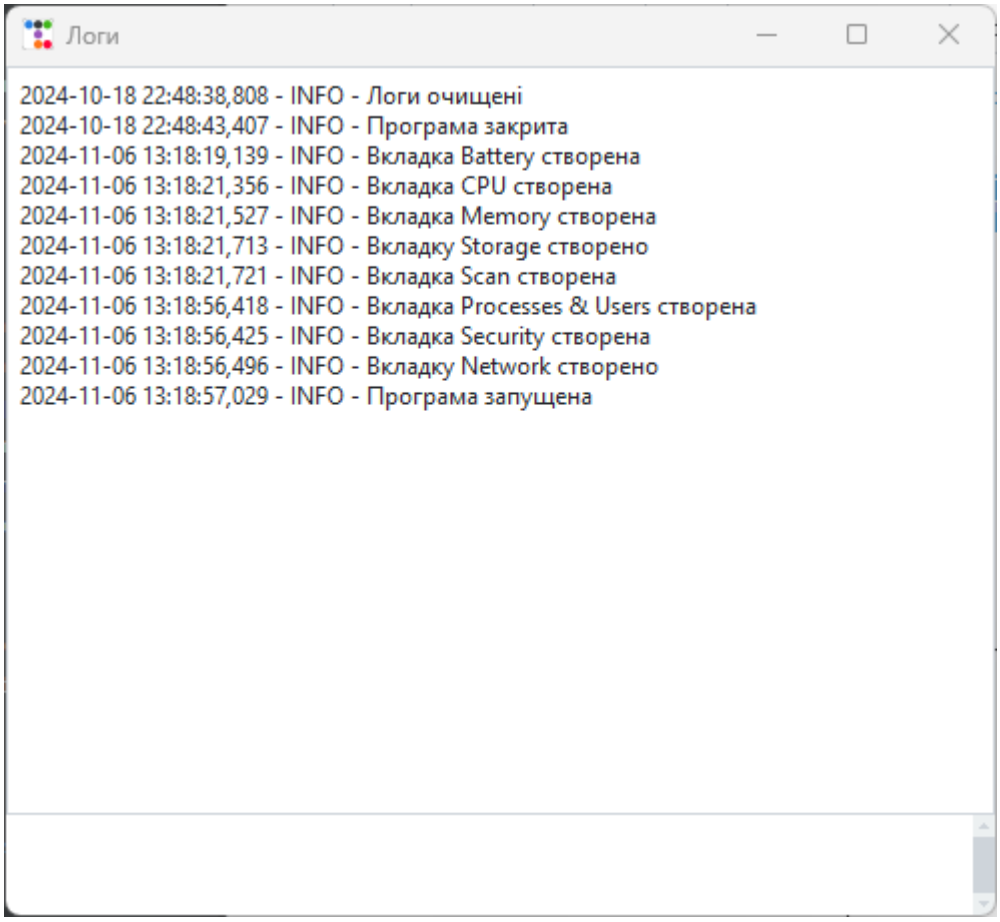


Рисунок 3.13 - Результати "Logs"

Рис.3.12 та рис.3.13 показує вкладку Logs, яка дозволяє переглядати логи за рівнями INFO, WARNING, ERROR, а також проводити аналіз аномалій у системі. Це важливий інструмент для моніторингу роботи програми та виявлення проблем, що виникають під час її роботи.

3.2 Обґрунтування переваги розробленого ПЗ

Розроблене ПЗ, BenchmarkApp, має низку важливих переваг, які роблять його цінним інструментом для користувачів, що прагнуть отримати докладну інформацію про роботу свого комп'ютера, а також для адміністраторів, які слідкують за стабільністю та безпекою систем. Основні переваги програми включають зручний інтерфейс, багато функцій для моніторингу та аналітики, інтегровані інструменти безпеки, та можливість

логування і аналізу аномалій. Нижче наведено детальні описи основних переваг.

Таблиця 3.2 - Основні переваги та їх обґрунтування

Перевага	**Опис**
Зручний інтерфейс	Програма має інтуїтивно зрозумілий інтерфейс, який дозволяє швидко знаходити необхідну інформацію у вкладках. Користувач може легко перемикатися між функціями, такими як моніторинг CPU, пам'яті, мережі та сканування на шкідливе ПЗ.
Комплексний моніторинг системи	Програма забезпечує доступ до всіх основних параметрів системи, включаючи заряд батареї, використання CPU, пам'яті та дискового простору, а також відображає активність мережі. Це дозволяє отримати повну картину роботи комп'ютера.
Аналіз безпеки	Вкладка безпеки забезпечує можливість генерування ключів, підписання файлів та обфускації коду, що дозволяє захистити програму від несанкціонованих змін та доступу, підвищуючи загальний рівень захищеності системи.
Сканування на шкідливе ПЗ	Програма дозволяє виконувати сканування окремих дисків або всіх дисків на наявність шкідливих файлів, використовуючи YARA-правила. Це забезпечує швидке виявлення потенційних загроз і допомагає підтримувати безпечне середовище.
Відображення активних процесів	Функція моніторингу процесів та користувачів відображає інформацію про всі активні процеси системи, їх використання CPU та пам'яті. Це дозволяє оперативно виявляти ресурсоємні або підозрілі процеси.
Логування та аналіз аномалій	Програма зберігає логи роботи із можливістю аналізу попереджень та помилок, що дозволяє відслідковувати та усувати потенційні проблеми, зокрема за допомогою вкладки для перегляду аномалій у логах.
Гнучкі налаштування	Користувачі можуть вибирати параметри для сканування, перегляду логів або оновлення графічної інформації, що дозволяє підлаштувати програму під конкретні потреби.

Таблиця 3.3 - Порівняння із подібними програмами

Критерій	**BenchmarkApp**	**Аналоги**
Інтерфейс	Інтуїтивний, багатовкладковий	Може бути менш зручним та складним для користувача
Моніторинг системи	Комплексний: CPU, пам'ять, мережа, процеси	Обмежений на певних параметрах
Інструменти безпеки	Генерація ключів, обфускація, аналіз підозрілих з'єднань	Часто відсутні або мають тільки базові функції
Логування	Є з можливістю аналізу аномалій	Не завжди передбачена функція аналізу логів та аномалій
Гнучкість налаштувань	Висока: індивідуальні налаштування для кожної вкладки	Обмежені можливості для налаштування параметрів

Загалом, BenchmarkApp забезпечує всебічний моніторинг, контроль безпеки та гнучкі налаштування для широкого спектра користувачів, що робить його універсальним інструментом для щоденного використання та ефективним рішенням для підтримки стабільності комп’ютера.

Висновки до розділу 3

У розділі 3 проведено тестування програмного забезпечення з метою перевірки його працездатності та виявлення можливих помилок. Було виконано низку перевірок, що дозволили оцінити відповідність програми встановленим вимогам і специфікаціям. Результати тестування підтвердили, що програмне забезпечення функціонує задовільно та стабільно. Ключові функції програми, включаючи моніторинг продуктивності системи, аналіз мережевої активності та контроль безпеки, були успішно перевірені. Контрольний приклад, описаний у розділі, підтвердив коректну роботу основних функцій та можливостей програми.

ВИСНОВКИ

У процесі роботи було розроблено комплексне програмне забезпечення для моніторингу продуктивності, безпеки та стабільності комп'ютерної системи. Програма забезпечує глибокий аналіз використання основних системних ресурсів, таких як процесор, оперативна пам'ять та накопичувачі, а також здійснює моніторинг мережевої активності та надає інструменти для виявлення потенційних загроз. Такі можливості є актуальними в умовах зростаючих вимог до надійності систем у сучасних умовах кіберзагроз, коли значення продуктивності та стабільності роботи комп'ютерів набуває особливої ваги. У розробленій програмі також передбачено інструменти для перевірки коду на шкідливе ПЗ, відстеження підозрілих з'єднань та аналізу поведінки користувачів і процесів у системі.

Розроблена система дозволяє не тільки виявляти поточні проблеми, але й забезпечує інформативну основу для подальшого аналізу. Наприклад, вона може виявити, як ресурси комп'ютера використовуються протягом робочого дня, який обсяг пам'яті залучається в пікові години навантаження або які мережеві з'єднання є підозрілими. Таке програмне забезпечення особливо актуальне для підприємств та організацій, де продуктивність і захист даних є критично важливими для виконання повсякденних задач і збереження інформаційної безпеки.

Досягнуті результати

1. Розроблено програмне забезпечення для комплексного моніторингу продуктивності системи, що включає такі компоненти, як вкладки для аналізу CPU, оперативної пам'яті, накопичувачів, моніторинг мережевої активності та логування. Кожна з цих вкладок надає окремий набір інструментів для збору й візуалізації відповідних даних. Наприклад, для CPU додана можливість спостерігати за

завантаженням кожного ядра, а для пам'яті – аналізувати її використання у вигляді діаграм. Подібний підхід дозволяє зосередити увагу на ключових ресурсах, що відповідають за продуктивність, і забезпечує можливість оперативно виявляти можливі вузькі місця у функціонуванні системи.

2. Забезпечено можливість виявлення загроз у мережевому середовищі. Інструменти для моніторингу мережевої активності в програмі реалізовані з можливістю відображення підозрілих з'єднань та аналізу з'єднань із зовнішніми IP-адресами. Це дозволяє вчасно ідентифікувати потенційні загрози та мінімізувати ризики для інформаційної безпеки. Наприклад, додано функцію перевірки на наявність підозрілих IP-адрес у мережевих з'єднаннях, що може виявити можливу несанкціоновану діяльність. Така функція є важливою для зменшення ризиків, пов'язаних із витоком даних або несанкціонованим доступом.
3. Здійснено комплексне тестування програмного забезпечення, яке підтвердило стабільність його роботи та відповідність заявленим вимогам і специфікаціям. Під час тестування було проведено детальну перевірку кожної з функцій програми: від моніторингу системних ресурсів до виявлення шкідливих процесів. Результати тестування засвідчили, що програмне забезпечення є стабільним у роботі та здатне до виконання завдань у різних робочих умовах. Тестування включало контрольні приклади для кожної функції, які підтвердили коректність реалізації основних можливостей програми.
4. Забезпечено зручний інтерфейс користувача з використанням компонентів бібліотеки ttkbootstrap, що забезпечує простий доступ до різних функцій програми. Структура інтерфейсу дозволяє швидко переходити між вкладками, легко спостерігати за інформацією про системні ресурси та працювати з результатами моніторингу. Така

організація спрощує взаємодію з програмою навіть для недосвідчених користувачів, роблячи її зручною та ефективною у повсякденному використанні. Додаткові елементи, такі як кольорові індикатори стану та візуалізація у вигляді діаграм, сприяють швидкому сприйняттю інформації.

На основі проведеного дослідження та розробки програмного забезпечення можна зробити висновок, що отримане рішення є ефективним інструментом для задач моніторингу продуктивності та безпеки комп'ютерних систем. Програма демонструє високу стабільність, забезпечуючи точність аналізу системних показників і оперативне виявлення потенційних загроз. Завдяки поєднанню функцій моніторингу, безпеки та зручного інтерфейсу, розроблене програмне забезпечення може успішно використовуватися в широкому спектрі середовищ, де необхідні стабільність, безпека та продуктивність. Програма є новим підходом до інтегрованого забезпечення стабільності комп'ютерних систем, де об'єднано моніторинг і безпеку, що сприяє ефективному контролю за роботою системи та її захистом.

ДЖЕРЕЛА

1. «Antutu,» [Електронний ресурс]. URL: https://ru.wikipedia.org/wiki/Antutu.
2. What Does AnTuTu Benchmark Actually Measure? [Електронний ресурс] – Режим доступу до ресурсу: https://www.makeuseof.com/tag/antutu-benchmark-measure/.
3. «Geekbench,» [Онлайновий]. Available: https://www.geekbench.com/.
4. Novabench [Електронний ресурс]. URL: https://novabench.com/.
5. Geekbench [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Geekbench.
6. «3DMark,» [Електронний ресурс]. URL: https://benchmarks.ul.com/3dmark.
7. «Cinebench benchmark,» [Електронний ресурс]. URL: http://www.wincore.ru/programs/8088-cinebench-proverennyy-vremenem-benchmark.html.
8. «PassMark Software,» [Електронний ресурс]. URL: https://www.passmark.com/.
9. «Top of Benchmarks,» [Електронний ресурс]. URL: https://aws.futuremark.com/en/resources/what-is-a-good-pcmark-10-score.
10. The best benchmarking software for your PC [Електронний ресурс] – Режим доступу до ресурсу: https://www.pcworld.com/article/394927/best-benchmarking-software-for-pc.html.
11. 12 Best PC Benchmark Software In 2023 [Електронний ресурс] – Режим доступу до ресурсу: https://www.softwaretestinghelp.com/best-pc-benchmark-software/.

12. «Documentation for Python,» [Електронний ресурс]. URL: https://www.python.org/.
13. Matthes E. Python Crash Course / Eric Matthes., 2016.
14. Barry P. Head-First Python, 2nd edition / Paul Barry., 2016.
15. «Python GUI Programming With Tkinter,» [Електронний ресурс]. URL: https://realpython.com/python-gui-tkinter/.
16. «Psutil documentation,» [Електронний ресурс]. URL: https://psutil.readthedocs.io/en/latest/.
17. «Що таке функціональні вимоги: приклади, визначення, повний посібник,» [Електронний ресурс]. URL: https://visuresolutions.com/uk/blog/functional-requirements/.
18. Функціональні і нефункціональні вимоги [Електронний ресурс] – Режим доступу до ресурсу: http://ni.biz.ua/7/7_5/7_50635_funktsionalnie-i-nefunktsionalnie-trebovaniya.html.
19. М. Олександр, «UML для бізнес-моделювання: для чого потрібні діаграми процесів,» [Електронний ресурс]. URL: https://evergreens.com.ua/ua/articles/uml-diagrams.html.
20. UML Class Diagram Tutorial [Електронний ресурс] – Режим доступу до ресурсу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/.
21. The Unified Modeling Language [Електронний ресурс] – Режим доступу до ресурсу: https://www.uml-diagrams.org/.
22. An Introduction to Tkinter [Електронний ресурс] – Режим доступу до ресурсу: https://www.cs.mcgill.ca/~hv/classes/MS/TkinterPres/.
23. Python GUI Guide: Introduction to Tkinter [Електронний ресурс] – Режим доступу до ресурсу: https://learn.sparkfun.com/tutorials/python-gui-guide-introduction-to-tkinter/tkinter-overview.

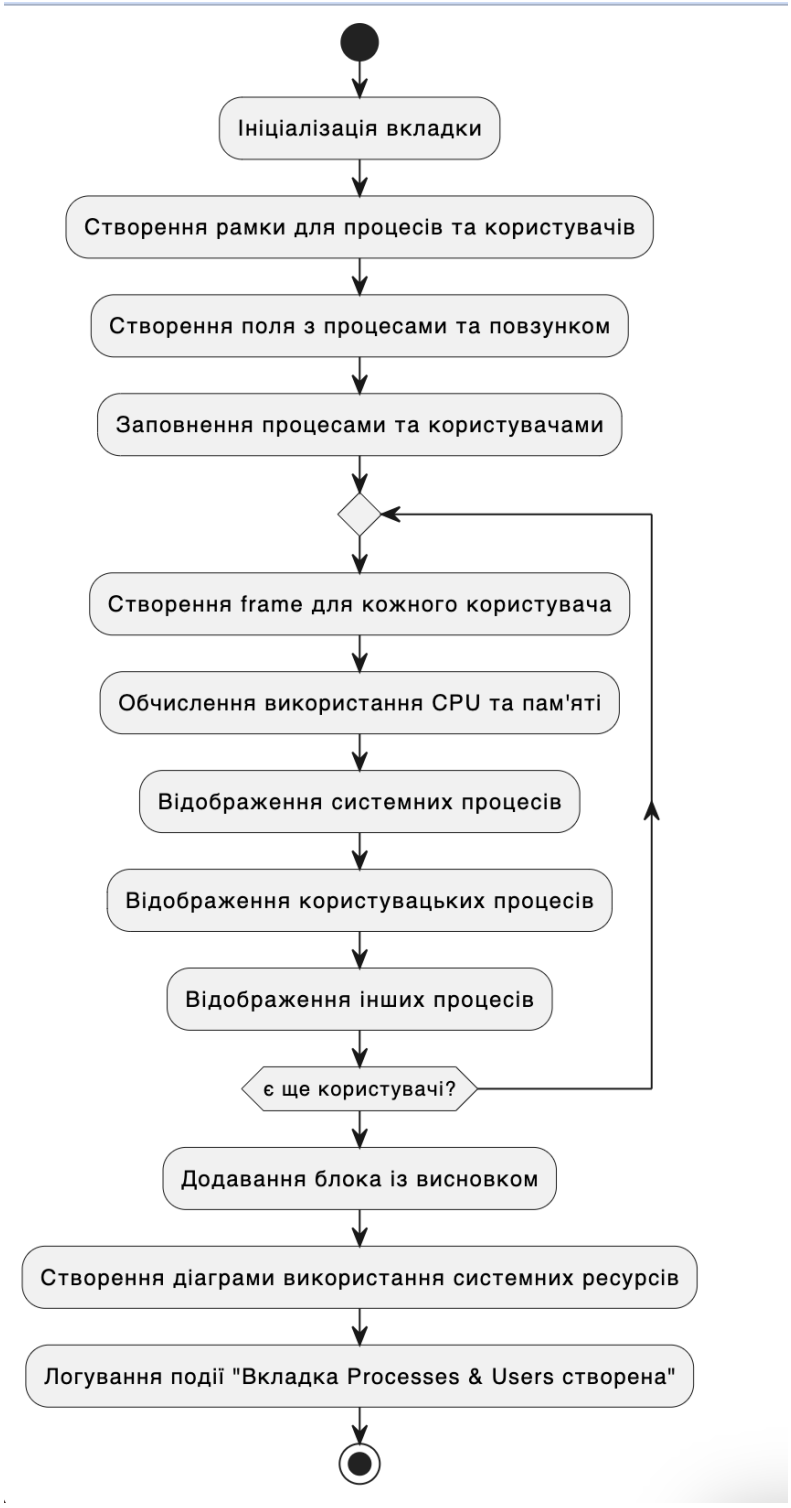
24. Top 10 Python GUI Frameworks Compared [Електронний ресурс] – Режим доступу до ресурсу: https://www.activestate.com/blog/top-10-python-gui-frameworks-compared/.
25. Архітектура програмного забезпечення: визначення та роль в проектуванні [Електронний ресурс] – Режим доступу до ресурсу: https://www.kursak.com/arkhitektura-prohramnoho-zabezpechennia/.
26. Волков А.М. Аналіз продуктивності та безпеки комп'ютерної системи за допомогою бенчмарку. Науково-практична конференція студентів і аспірантів «Теоретичні та прикладні аспекти розробки комп'ютерних систем '2024», факультет інформаційних технологій НУБІП України, 25-26 кітня 2024 р. С.225-226 URL: https://drive.google.com/file/d/1F-kDaVvyc46dGPBc_S9XLVZVB3tWahYm/view.
27. Волков А.М. Виявлення та оцінка активних процесів у бенчмарках як інструмент виявлення потенційно небажаних програм. XIV міжнародна науково-практична конференція молодих вчених «Інформаційні технології: економіка, техніка, освіта». факультет інформаційних технологій НУБІП України, 26-27 жовтня 2024 р. С.105-106 URL: https://drive.google.com/file/d/1w6p_Bll7NY1VCQ8zxBYOBsrGNfLM5hgz/view.

ДОДАТОК 1

Діаграма класів

ДОДАТОК 2

Принципова схема



ДОДАТОК 3

Лістинг коду

```
import platform
import subprocess
import psutil
import wmi
import socket
import matplotlib
matplotlib.use("TkAgg")
import matplotlib.pyplot as plt
import yara
import os
import tkinter as tk
from tkinter import ttk
from tkinter import ttk as ttkb
from threading import Thread
import sys
import logging
import datetime
import ttkbootstrap as ttkb
from ttkbootstrap.constants import *
from tkinter import messagebox
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from ttkbootstrap import Style
from PIL import Image, ImageTk
from cryptography.hazmat.primitives.asymmetric import rsa,
padding
from cryptography.hazmat.primitives import serialization,
hashes

# Налаштування логування
logging.basicConfig(filename='app.log', level=logging.INFO,
                    format='%(asctime)s - %(levelname)s -
%(message)s')


def log_event(event):
    logging.info(event)


def log_warning(event):
    logging.warning(event)
```

```
def log_error(event):
    logging.error(event)


def analyze_logs():
    with open('app.log', 'r') as log_file:
        logs = log_file.readlines()

    anomalies = []
    for log in logs:
        if "ERROR" in log or "WARNING" in log:
            anomalies.append(log)

    return anomalies


def count_log_levels():
    with open('app.log', 'r') as log_file:
        logs = log_file.readlines()

    info_count = 0
    warning_count = 0
    error_count = 0

    for log in logs:
        if "INFO" in log:
            info_count += 1
        elif "WARNING" in log:
            warning_count += 1
        elif "ERROR" in log:
            error_count += 1

    return info_count, warning_count, error_count


def clear_logs():
    with open('app.log', 'w'):
        pass

class BenchmarkApp(ttkb.Window):
    def __init__(self):
        super().__init__(themename="flatly")
        self.title("Benchmark")
        self.geometry("1100x600")
```

```
        self.resizable(False, False)

        self.notebook = ttkb.Notebook(self)
        self.notebook.pack(fill=tk.BOTH, expand=True)

        self.battery_tab = ttkb.Frame(self.notebook)
        self.cpu_tab = ttkb.Frame(self.notebook)
        self.memory_tab = ttkb.Frame(self.notebook)
        self.storage_tab = ttkb.Frame(self.notebook)
        self.scan_tab = ttkb.Frame(self.notebook)
        self.process_tab = ttkb.Frame(self.notebook)
        self.security_tab = ttkb.Frame(self.notebook)
        self.network_tab = ttkb.Frame(self.notebook)  # вкладка
моніторингу мережі
        self.logs_tab  =  ttkb.Frame(self.notebook)   #  вкладка
логування

        self.notebook.add(self.battery_tab, text="Battery")
        self.notebook.add(self.cpu_tab, text="CPU")
        self.notebook.add(self.memory_tab, text="Memory")
        self.notebook.add(self.storage_tab, text="Storage")
        self.notebook.add(self.scan_tab, text="Scan")
        self.notebook.add(self.process_tab,  text="Processes  &
Users")
        self.notebook.add(self.security_tab, text="Security")
        self.notebook.add(self.network_tab, text="Network")   #
вкладка моніторингу мережі
        self.notebook.add(self.logs_tab,   text="Logs")       #
вкладка логування

        self.create_battery_tab()
        self.create_cpu_tab()
        self.create_memory_tab()
        self.create_storage_tab()
        self.create_scan_tab()
        self.create_process_tab()
        self.create_security_tab()
        self.create_network_tab()  # вкладка моніторингу мережі
        self.create_logs_tab()  # вкладка логування

        self.scanning = False

        # Логування запуску програми
        log_event("Програма запущена")
```

```
        # Логування закриття програми
        self.protocol("WM_DELETE_WINDOW", self.on_closing)

    def create_logs_tab(self):
        try:
            # Супровідний текст
            info_label = ttkb.Label(self.logs_tab,
                                    text="У цьому розділі ви
можете переглядати логи та аналізувати аномалії у системі.",
                                    bootstyle="info")
            info_label.pack(pady=10)

            # Рамка для кнопок
            button_frame = ttkb.Frame(self.logs_tab)
            button_frame.pack(pady=20)

            log_button       =       ttkb.Button(button_frame,
text="Переглянути      логи",       command=self.display_logs,
bootstyle="info")
            log_button.pack(side="left", padx=10)

            anomaly_button     =     ttkb.Button(button_frame,
text="Переглянути аномалії", command=self.display_anomalies,
                                       bootstyle="warning")
            anomaly_button.pack(side="left", padx=10)

            clear_button      =      ttkb.Button(button_frame,
text="Очистити логи", command=self.clear_logs_action,
                                      bootstyle="danger")
            clear_button.pack(side="left", padx=10)

            # Графічний елемент (іконка)
            icon_label  =  ttkb.Label(self.logs_tab,  text="□",
font=("Helvetica", 48))
            icon_label.pack(pady=20)

            # Додавання горизонтальної стовпчастої діаграми для
відображення кількості записів журналу за рівнями журналу
            fig, ax = plt.subplots()

            info_count,    warning_count,    error_count    =
count_log_levels()
```

```
            levels = ['INFO', 'WARNING', 'ERROR']
            counts = [info_count, warning_count, error_count]

            ax.barh(levels,      counts,      color=['#66b3ff',
'#ffcc99', '#ff9999'])

            ax.set_title('Кількість    записів    журналу    за
рівнями')
            ax.set_xlabel('Кількість записів')
            ax.set_ylabel('Рівень журналу')

            canvas           =           FigureCanvasTkAgg(fig,
master=self.logs_tab)

            canvas.draw()

            canvas.get_tk_widget().pack(pady=40)   # Піднімаємо
діаграму трохи вище

        except Exception as e:
            log_error(f"Помилка під час створення вкладки Logs:
{e}")

    def display_logs(self):
        try:
            log_window = tk.Toplevel(self)
            log_window.title("Логи")

            log_text = tk.Text(log_window, wrap="word")
            log_text.pack(expand=True, fill="both")

            scrollbar       =       ttkb.Scrollbar(log_window,
orient="vertical", command=log_text.yview)
            log_text.configure(yscrollcommand=scrollbar.set)
            scrollbar.pack(side="right", fill="y")

            with open('app.log', 'r') as log_file:
                log_text.insert(tk.END, log_file.read())
        except Exception as e:
            log_error(f"Помилка відображення логів: {e}")

    def display_anomalies(self):
        try:
            anomalies = analyze_logs()
```

```
            anomaly_window = tk.Toplevel(self)
            anomaly_window.title("Аномалії у логах")

            anomaly_text = tk.Text(anomaly_window, wrap="word")
            anomaly_text.pack(expand=True, fill="both")

            scrollbar     =     ttkb.Scrollbar(anomaly_window,
orient="vertical", command=anomaly_text.yview)

anomaly_text.configure(yscrollcommand=scrollbar.set)
            scrollbar.pack(side="right", fill="y")

            if anomalies:
                for anomaly in anomalies:
                    anomaly_text.insert(tk.END, anomaly + "\n")
            else:
                anomaly_text.insert(tk.END,    "Аномалії    не
виявлено. Усі системи працюють в штатному режимі.")
        except Exception as e:
            log_error(f"Помилка   при   відображенні   аномалій:
{e}")

    def clear_logs_action(self):
        try:
            clear_logs()
            log_event("Логи очищені")
        except Exception as e:
            log_error(f"Помилка при очищенні логів: {e}")

    def on_closing(self):
        log_event("Програма закрита")
        self.destroy()

#Вкладка Мережа
    def create_network_tab(self):
        try:
            network_frame  =  ttkb.LabelFrame(self.network_tab,
text="Моніторинг мережної активності", bootstyle="info")
            network_frame.pack(padx=20,  pady=20,  fill=tk.BOTH,
expand=True)

            self.network_info_label = ttkb.Label(network_frame,
text="", bootstyle="info")
```

```
            self.network_info_label.pack(pady=10)

            self.update_network_info()

            suspicious_frame                                    =
ttkb.LabelFrame(self.network_tab,  text="Виявлення  підозрілих
з’єднань",

bootstyle="danger")
            suspicious_frame.pack(padx=20,            pady=20,
fill=tk.BOTH, expand=True)

            self.suspicious_info_label                          =
ttkb.Label(suspicious_frame, text="", bootstyle="danger")
            self.suspicious_info_label.pack(pady=10)

            self.update_suspicious_info()

            # Кнопка для відкриття вікна з інформацією про
мережні інтерфейси
            self.interface_button = ttkb.Button(network_frame,
text="Показати інформацію про мережеві інтерфейси",

command=self.show_interface_info, bootstyle="info")
            self.interface_button.pack(pady=10)

            # Графічна візуалізація
            self.canvas_frame = ttkb.Frame(network_frame)
            self.canvas_frame.pack(pady=10)
            self.update_graph()
            log_event("Вкладку Network створено")
        except Exception as e:
            log_error(f"Помилка  під  час  створення  вкладки
Network: {e}")

    def get_current_connection(self):
        wifi_interfaces = ["wlan", "wi-fi", "wireless", "wifi",
"беспроводная", "бездротова"]
        stats = psutil.net_if_stats()
        addrs = psutil.net_if_addrs()

        for interface in stats:
            if  stats[interface].isup  and  interface.lower()  !=
'loopback pseudo-interface 1':  # Виключаємо loopback
```

```
                for addr in addrs[interface]:
                    if addr.family == socket.AF_INET:     #
Перевірка на IPv4
                        #print(f"Перевірка інтерфейса:
{interface}, Адрес: {addr.address}")     # Відлагоджувальна
інформація
                        # Тут виконується перевірка імені
інтерфейсу
                        interface_lower = interface.lower()
                        if any(wifi in interface_lower for wifi
in wifi_interfaces):
                            return f"{interface} -
{addr.address}", "Wi-Fi"
                        else:
                            return f"{interface} -
{addr.address}", "Ethernet"

        return "Немає активного підключення", "Невідомо"

    def update_network_info(self):
        net_io = psutil.net_io_counters()
        current_connection, connection_type =
self.get_current_connection()

        #print(f"Поточні дані: {current_connection}, Тип
з'єднання: {connection_type}")  # Відлагоджувальна інформація

        net_info = f"Current Connection:
{current_connection}\nBytes Sent: {net_io.bytes_sent}\nBytes
Received: {net_io.bytes_recv}\n"

        # Перевірка типу з'єднання та вибір картинки
        if connection_type == "Wi-Fi":
            img_path = "wifi.png"
        elif connection_type == "Ethernet":
            img_path = "ethernet.png"
        else:
            img_path = "unknown.png"

        # Завантажуємо зображення
        try:
            img = Image.open(img_path)
            img = img.resize((20, 20), Image.LANCZOS)
            img = ImageTk.PhotoImage(img)
```

```
            # Оновлюємо віджет з інформацією
            self.network_info_label.config(text=net_info,
image=img, compound=tk.LEFT)
            self.network_info_label.image = img  # Зберігаємо
посилання на зображення

        except Exception as e:
            print(f"Помилка завантаження зображення: {e}")  #
Відладка помилок

    def update_suspicious_info(self):
        suspicious_connections                                =
self.detect_suspicious_connections()
        if suspicious_connections:
            suspicious_info = "Виявлені підозрілі з'єднання:\n"
+ "\n".join(suspicious_connections)
        else:
            suspicious_info = "Підозрілі з'єднання не виявлені"

self.suspicious_info_label.config(text=suspicious_info)

    def detect_suspicious_connections(self):
        suspicious_connections = []
        connections = psutil.net_connections()
        for conn in connections:
            if conn.status == 'ESTABLISHED' and conn.raddr:
                ip, port = conn.raddr
                if self.is_suspicious(ip):
                    suspicious_connections.append(f"Підозрілі
з'єднання: {ip}:{port}")
        return suspicious_connections

    def is_suspicious(self, ip):
        # Список підозрілих IP-адрес
        suspicious_ips = [
            "192.168.1.1", "10.0.0.1",
            "192.168.0.105", "172.16.0.5",
            # Добавим больше IP для проверки
            "192.168.2.1", "192.168.2.2",
            "192.168.3.1", "192.168.3.2"
        ]

        # Перевірка на збіг IP у списку підозрілих
```

```
        return ip in suspicious_ips

    def show_interface_info(self):
        interface_window = tk.Toplevel(self)
        interface_window.title("Інформація про інтерфейси
мережі")
        interface_window.geometry("600x400")

        addrs = psutil.net_if_addrs()
        net_info = ""
        for interface, addr_list in addrs.items():
            net_info += f"\nInterface: {interface}\n"
            for addr in addr_list:
                net_info += f"  Address: {addr.address}\n"

        interface_label = ttkb.Label(interface_window,
text=net_info, bootstyle="info")
        interface_label.pack(pady=10, padx=10)

    def update_graph(self):
        for widget in self.canvas_frame.winfo_children():
            widget.destroy()

        net_io = psutil.net_io_counters()
        sent = net_io.bytes_sent
        recv = net_io.bytes_recv

        # Завантаження зображення комп'ютера
        img = Image.open("computer.png")
        img = img.resize((100, 100), Image.LANCZOS)
        img = ImageTk.PhotoImage(img)

        # Створення мітки для відображення зображення комп'ютера
        img_label = ttkb.Label(self.canvas_frame, image=img)
        img_label.image = img  # Збереження посилання на
зображення
        img_label.pack()

        # Відображення стрілок для надсилання та отримання даних
    sent_arrow = ttkb.Label(self.canvas_frame, text=f'↑ Sent: {sent}', bootstyle="info")
        sent_arrow.pack()

    recv_arrow = ttkb.Label(self.canvas_frame, text=f'↓ Received: {recv}', bootstyle="info")
        recv_arrow.pack()
```

```
    # Вкладка Безпека
    def create_security_tab(self):
        try:
            security_frame = ttkb.LabelFrame(self.security_tab,
text="Безпека")
            security_frame.pack(padx=20, pady=20)

            explanation_label = ttkb.Label(security_frame,
                                           text="Ми
використовуємо обфускацію коду та цифрові підписи для захисту
нашої програми. Обфускація робить код важкочитабельним, а
цифрові підписи гарантують, що код не було змінено після
створення.",
                                           wraplength=500)
            explanation_label.pack(pady=10)

            generate_keys_button = ttkb.Button(security_frame,
text="Створити ключи", command=self.generate_keys,

bootstyle="primary")
            generate_keys_button.pack(pady=10)

            sign_file_button   =   ttkb.Button(security_frame,
text="Підписати файл", command=self.sign_file,

bootstyle="success")
            sign_file_button.pack(pady=10)

            verify_signature_button                          =
ttkb.Button(security_frame, text="Перевірити підпис",

command=self.verify_signature, bootstyle="info")
            verify_signature_button.pack(pady=10)

            obfuscate_code_button = ttkb.Button(security_frame,
text="Обфускація коду", command=self.obfuscate_code,

bootstyle="warning")
            obfuscate_code_button.pack(pady=10)

            self.result_text     =     tk.Text(security_frame,
height=10, width=80)
            self.result_text.pack()
```

```
            log_event("Вкладка Security створена")
        except Exception as e:
            log_error(f"Помилка під час створення вкладки
Security: {e}")

    def generate_keys(self):
        private_key = rsa.generate_private_key(
            public_exponent=65537,
            key_size=2048,
        )

        public_key = private_key.public_key()

        with open("private_key.pem", "wb") as f:
            f.write(private_key.private_bytes(
                encoding=serialization.Encoding.PEM,

format=serialization.PrivateFormat.TraditionalOpenSSL,

encryption_algorithm=serialization.NoEncryption()
            ))

        with open("public_key.pem", "wb") as f:
            f.write(public_key.public_bytes(
                encoding=serialization.Encoding.PEM,

format=serialization.PublicFormat.SubjectPublicKeyInfo
            ))

        self.result_text.insert(tk.END, "Ключи створені.\n")

    def sign_file(self):
        with open("private_key.pem", "rb") as f:
            private_key = serialization.load_pem_private_key(
                f.read(),
                password=None,
            )

        with open("main.py", "rb") as f:
            file_data = f.read()

        signature = private_key.sign(
            file_data,
            padding.PSS(
```

```
                mgf=padding.MGF1(hashes.SHA256()),
                salt_length=padding.PSS.MAX_LENGTH
            ),
            hashes.SHA256()
        )

        with open("signature.sig", "wb") as f:
            f.write(signature)

        self.result_text.insert(tk.END, "Файл підписан.\n")

    def verify_signature(self):
        with open("public_key.pem", "rb") as f:
            public_key                                          =
serialization.load_pem_public_key(f.read())

        with open("signature.sig", "rb") as f:
            signature = f.read()

        with open("main.py", "rb") as f:
            file_data = f.read()

        try:
            public_key.verify(
                signature,
                file_data,
                padding.PSS(
                    mgf=padding.MGF1(hashes.SHA256()),
                    salt_length=padding.PSS.MAX_LENGTH
                ),
                hashes.SHA256()
            )
            self.result_text.insert(tk.END, "Підпис вірний.\n")
        except Exception as e:
            self.result_text.insert(tk.END, f"Підпис невірний:
{e}\n")

    def obfuscate_code(self):
        os.system('pyarmor                                      gen
"C:\\Users\\foxik\\OneDrive\\Рабочий
стол\\ДИПЛОМ_МАГІСТР\\CODE\\main.py"                             --
output="C:\\Users\\foxik\\OneDrive\\Рабочий
стол\\ДИПЛОМ_МАГІСТР\\CODE\\dist"')
        self.result_text.insert(tk.END, "Код обфускований.\n")
```

```
#Вкладка Батарея
    def create_battery_tab(self):
        try:
            # Створюємо стиль для ttkbootstrap
            style = Style(theme='flatly')

            # код для створення вкладки Battery
            battery_frame  =  ttkb.LabelFrame(self.battery_tab,
text="Battery Information", bootstyle="info", width=900,
                                              height=700)
            battery_frame.pack(padx=20,  pady=20,  expand=True,
fill="both")

            battery_info = self.get_battery_info()
            battery_percent = self.get_battery_percent()

            # Calculate battery mark
            battery_mark                                          =
self.calculate_battery_mark(battery_percent)

            # Display battery information and mark as text
            battery_info_label    =    ttkb.Label(battery_frame,
text=battery_info, bootstyle="info")
            battery_info_label.pack()
            battery_mark_label    =    ttkb.Label(battery_frame,
text=f"Battery Mark: {battery_mark}/100")
            battery_mark_label.pack()

            # Set color based on battery percentage
            if battery_percent < 20:
                battery_color = "red"
            else:
                battery_color = "lightgreen"

            # Create a pie chart to visualize battery percentage
            fig,    ax    =    plt.subplots(figsize=(8,    6),
subplot_kw=dict(aspect="equal"))  # Increasing the size of the
figure
            labels = ["Battery"]
            sizes = [battery_percent, 100 - battery_percent]
            colors = [battery_color, "lightgray"]
```

```
            wedges,    texts,    autotexts    =    ax.pie(sizes,
colors=colors, autopct="%1.1f%%", startangle=90,

wedgeprops=dict(width=0.3, edgecolor='w'))

            # Make the text more readable
            plt.setp(autotexts,     size=10,     weight="bold",
color="black")
            ax.set_title("Battery Percentage")

            # Add Legend
            ax.legend(wedges, ["Current Charge", "Remaining"],
title="Battery Status", loc="upper left",
                      bbox_to_anchor=(1,  1))    #  Moving  the
legend

            # Embed the plot in the Tkinter canvas
            canvas           =           FigureCanvasTkAgg(fig,
master=battery_frame)
            canvas.draw()
            canvas.get_tk_widget().pack()

            # Add image in the center with Tkinter
            img = Image.open("battery.png")
            img = img.resize((50, 50), Image.LANCZOS)
            img = ImageTk.PhotoImage(img)

            # Calculate center coordinates
            img_label = ttkb.Label(battery_frame, image=img)
            img_label.image = img  # Saving a link to an image
            img_label.place(relx=0.5,                rely=0.5,
anchor="center", x=10, y=30)   # Set coordinates to center of
chart

            log_event("Вкладка Battery створена")
        except Exception as e:
            log_error(f"Помилка  під  час  створення  вкладки
Battery: {e}")

    def get_battery_info(self):
        battery = psutil.sensors_battery()
        if battery.power_plugged:
            status = "Plugged in"
        else:
```

```
            status = "Not plugged in"
        info      =      f"Status:      {status}\nPercentage:
{battery.percent}%"
        return info

    def get_battery_percent(self):
        battery = psutil.sensors_battery()
        return battery.percent

    def calculate_battery_mark(self, battery_percent):
        if battery_percent < 75:
            return 15
        else:
            return 30

    def create_cpu_tab(self):
        try:
            # Код створення вкладки CPU
            cpu_frame = ttkb.LabelFrame(self.cpu_tab, text="CPU
Performance")
            cpu_frame.pack(padx=20, pady=20)

            cpu_info = self.get_cpu_info()

            # Calculate CPU mark
            cpu_mark = self.calculate_cpu_mark()

            # Display CPU information and mark as text
            cpu_info_label       =       ttkb.Label(cpu_frame,
text=cpu_info)
            cpu_info_label.pack()
            cpu_mark_label  =  ttkb.Label(cpu_frame,  text=f"CPU
Mark: {cpu_mark}/100")
            cpu_mark_label.pack()

            # Create a bar chart to visualize CPU usage
            cpu_percentages  =  psutil.cpu_percent(interval=1,
percpu=True)
            cores    =    [f"CPU{i    +    1}"    for    i    in
range(len(cpu_percentages))]
            plt.figure(figsize=(11, 4))

            plt.bar(cores, cpu_percentages, color="#ffcc5f")
            plt.ylim(0, 100)
```

```
            plt.xlabel("CPU")
            plt.ylabel("Percentage")
            plt.title("CPU Usage")
            plt.gca().spines["right"].set_visible(False)
            plt.gca().spines["top"].set_visible(False)
            plt.gca().yaxis.set_ticks_position("left")
            plt.gca().xaxis.set_ticks_position("bottom")
            plt.grid(axis="y", linestyle="--", linewidth=0.7)
            plt.savefig("cpu_chart.png")

            # Display the CPU chart image
            cpu_chart_image                                    =
tk.PhotoImage(file="cpu_chart.png")
            cpu_chart_label       =       ttkb.Label(cpu_frame,
image=cpu_chart_image)
            cpu_chart_label.image = cpu_chart_image
            cpu_chart_label.pack()
            log_event("Вкладка CPU створена")
        except Exception as e:
            log_error(f"Помилка під час створення вкладки CPU:
{e}")

    def create_memory_tab(self):
        try:
            # код для створення вкладки Memory
            memory_frame   =   ttkb.LabelFrame(self.memory_tab,
text="Memory Usage")
            memory_frame.pack(padx=20, pady=20)

            memory_info = self.get_memory_info()

            # Calculate memory mark
            memory_mark = self.calculate_memory_mark()

            # Display memory information and mark as text
            memory_info_label    =    ttkb.Label(memory_frame,
text=memory_info)
            memory_info_label.pack()
            memory_mark_label    =    ttkb.Label(memory_frame,
text=f"Memory Mark: {memory_mark}/100")
            memory_mark_label.pack()

            # Create a pie chart to visualize memory usage
            memory_usage = psutil.virtual_memory()
```

```
            plt.figure(figsize=(6, 4))
            labels = ["Used", "Available"]
            sizes = [memory_usage.used, memory_usage.available]
            colors = ["#ffcc5f", "lightgray"]
            plt.pie(sizes,    labels=labels,    colors=colors,
autopct="%1.1f%%", startangle=90)
            plt.axis("equal")
            plt.title("Memory Usage")
            plt.savefig("memory_chart.png")

            # Display the memory chart image
            memory_chart_image                                  =
tk.PhotoImage(file="memory_chart.png")
            memory_chart_label    =    ttkb.Label(memory_frame,
image=memory_chart_image)
            memory_chart_label.image = memory_chart_image
            memory_chart_label.pack()
            log_event("Вкладка Memory створена")
        except Exception as e:
            log_error(f"Помилка  під  час  створення  вкладки
Memory: {e}")

    def get_memory_info(self):
        memory_usage = psutil.virtual_memory()
        total_size = self.format_size(memory_usage.total)
        used_size = self.format_size(memory_usage.used)
        memory_info = f"Total: {total_size}\nUsed: {used_size}"
        return memory_info

    def calculate_memory_mark(self):
        memory_usage = psutil.virtual_memory()
        memory_mark      =      (memory_usage.available      /
memory_usage.total) * 100
        return int(memory_mark)

    def create_storage_tab(self):
        try:
            # Код для створення вкладки Storage
            storage_frame  =  ttkb.LabelFrame(self.storage_tab,
text="Storage Performance")
            storage_frame.pack(padx=20, pady=20)

            storage_info = self.get_storage_info()
```

```
            # Calculate SSD speed mark
            ssd_speed = self.get_ssd_speed()
            ssd_mark = self.calculate_ssd_mark(ssd_speed)

            # Display storage information and SSD mark as text
            storage_info_label    =    ttkb.Label(storage_frame,
text=storage_info)
            storage_info_label.pack()
            ssd_mark_label     =     ttkb.Label(storage_frame,
text=f"SSD Speed Mark: {ssd_mark}/100")
            ssd_mark_label.pack()

            # Create a bar chart to visualize total and used
sizes of storage
            partitions = psutil.disk_partitions()
            total_sizes = []
            used_sizes = []
            for partition in partitions:
                if platform.system() == "Windows":
                    if    "cdrom"    in    partition.opts    or
partition.fstype == "":
                        continue
                elif platform.system() == "Darwin":
                    if "/Volumes" in partition.mountpoint:
                        continue
                else:
                    if "/media" in partition.mountpoint:
                        continue

                usage = psutil.disk_usage(partition.mountpoint)
                total_sizes.append(usage.total)
                used_sizes.append(usage.used)

            plt.figure(figsize=(6, 4))
            plt.bar(range(len(total_sizes)),       total_sizes,
color="#ffcc5f", label="Total Size")
            plt.bar(range(len(used_sizes)),         used_sizes,
color="#ff3d3d", label="Used Size")
            plt.xticks(range(len(partitions)),
[partition.device for partition in partitions], rotation=45)
            plt.xlabel("Storage")
            plt.ylabel("Size (bytes)")
            plt.title("Storage Usage")
            plt.legend()
```

```
            plt.gca().spines["right"].set_visible(False)
            plt.gca().spines["top"].set_visible(False)
            plt.gca().yaxis.set_ticks_position("left")
            plt.gca().xaxis.set_ticks_position("bottom")
            plt.grid(axis="y", linestyle="--", linewidth=0.5)
            plt.tight_layout()
            plt.savefig("storage_chart.png")

            # Display the storage chart image
            storage_chart_image                                    =
tk.PhotoImage(file="storage_chart.png")
            storage_chart_label   =   ttkb.Label(storage_frame,
image=storage_chart_image)
            storage_chart_label.image = storage_chart_image
            storage_chart_label.pack()
            log_event("Вкладку Storage створено")
        except Exception as e:
            log_error(f"Помилка  під  час  створення  вкладки
Storage: {e}")

    def get_cpu_info(self):
        cpu_info = f"Processor: {platform.processor()}\n"
        cpu_info          +=          f"Physical          Cores:
{psutil.cpu_count(logical=False)}\n"
        cpu_info           +=           f"Total           Cores:
{psutil.cpu_count(logical=True)}"
        return cpu_info

    def calculate_cpu_mark(self):
        cpu_percent = psutil.cpu_percent(interval=1)
        cpu_mark = 100 - cpu_percent
        return cpu_mark

    def get_memory_info(self):
        memory_usage = psutil.virtual_memory()
        total_size = self.format_size(memory_usage.total)
        used_size = self.format_size(memory_usage.used)
        memory_info = f"Total: {total_size}\nUsed: {used_size}"
        return memory_info

    def calculate_memory_mark(self):
        memory_usage = psutil.virtual_memory()
        memory_mark      =      (memory_usage.available      /
memory_usage.total) * 100
```

```
        return int(memory_mark)

    def get_storage_info(self):
        partitions = psutil.disk_partitions()
        storage_info = ""
        for partition in partitions:
            if platform.system() == "Windows":
                if     "cdrom"     in     partition.opts     or
partition.fstype == "":
                    continue
            elif platform.system() == "Darwin":
                if "/Volumes" in partition.mountpoint:
                    continue
            else:
                if "/media" in partition.mountpoint:
                    continue

            usage = psutil.disk_usage(partition.mountpoint)
            total_size = self.format_size(usage.total)
            used_size = self.format_size(usage.used)

            storage_info += f"Total Size: {total_size}\n"
            storage_info += f"Used Size: {used_size}\n"

        return storage_info

    def get_ssd_speed(self):
        c = wmi.WMI()
        ssd_speed = {}
        for disk in c.Win32_DiskDrive(MediaType='SSD'):
            drive = disk.Name
            interface_type = disk.InterfaceType
            cmd  =  f"wmic  diskdrive  where  Name='{drive}'  get
BytesPerSector /format:list"
            output        =        subprocess.check_output(cmd,
shell=True).decode().strip()
            lines = output.split('\n')
            for line in lines:
                if line.startswith('BytesPerSector='):
                    bytes_per_sector = int(line.split('=')[1])
                    break
            else:
                bytes_per_sector = None
```

```
            if bytes_per_sector is not None and interface_type
== 'Solid State':
                cmd = f"winsat disk -drive {drive}"
                speed_output   =   subprocess.check_output(cmd,
shell=True).decode().strip()
                speed_lines = speed_output.split('\n')
                for speed_line in speed_lines:
                    if              speed_line.startswith('Disk
Sequential'):
                        speed       =       speed_line.split(':
')[1].strip().split()[0]
                        ssd_speed[drive] = int(speed)
        return ssd_speed

    def calculate_ssd_mark(self, ssd_speed):
        ssd_mark = 100
        for drive, speed in ssd_speed.items():
            if speed < 200:
                ssd_mark -= 10
        return ssd_mark

#Scan Block
    def collect_yara_files(self, directory):
        yara_files = []
        for root, dirs, files in os.walk(directory):
            for file in files:
                if          file.endswith('.yara')          or
file.endswith('.yar'):
                    yara_files.append(os.path.join(root,
file))
        return yara_files

    def create_scan_tab(self):
        try:
            scan_frame     =     ttkb.LabelFrame(self.scan_tab,
text="Сканування на шкідливе ПЗ")
            scan_frame.pack(padx=20, pady=20)

            self.scan_option = tk.StringVar()
            self.scan_option.set("1")

            ttkb.Radiobutton(scan_frame, text="Сканувати певний
диск", variable=self.scan_option, value="1").pack(
                anchor=tk.W)
```

```
            ttkb.Radiobutton(scan_frame, text="Повне сканування
всіх дисків", variable=self.scan_option,
                             value="2").pack(anchor=tk.W)

            self.disk_entry = ttkb.Entry(scan_frame)
            self.disk_entry.pack()

            self.scan_button     =     ttkb.Button(scan_frame,
text="Почати сканування", command=self.start_scan,

bootstyle="success")
            self.scan_button.pack(pady=10)

            self.stop_button     =     ttkb.Button(scan_frame,
text="Зупинити сканування", command=self.stop_scan,
                                          bootstyle="danger")
            self.stop_button.pack(pady=10)

            self.progress_label    =    ttkb.Label(scan_frame,
text="Прогрес: 0%")
            self.progress_label.pack()

            self.scan_result_text     =     tk.Text(scan_frame,
height=10, width=80)
            self.scan_result_text.pack()

            log_event("Вкладка Scan створена")
        except Exception as e:
            log_error(f"Помилка  при  створенні  вкладки  Scan:
{e}")

    def update_scan_tab(self, scan_info):
        # Оновлюємо інформацію у віджетах вкладки "Scan"
        self.scan_result_text.insert(tk.END, scan_info)

    def start_scan(self):
        self.result_text.delete(1.0, tk.END)
        self.progress_label.config(text="Прогрес: 0%")
        scan_option = self.scan_option.get()
        disk = self.disk_entry.get().upper()
        self.scanning = True

        if scan_option == "1" and disk:
            scan_path = f"{disk}:/"
```

```
            if not os.path.exists(scan_path):
                scan_info = f"Шлях {scan_path} не існує.\n"
                self.update_scan_tab(scan_info)
                return
            thread    =    Thread(target=self.scan_for_malware,
args=(scan_path,))
            thread.start()
        elif scan_option == "2":
            thread = Thread(target=self.scan_all_disks)
            thread.start()
        else:
            scan_info = "Невірний вибір.\n"
            self.update_scan_tab(scan_info)

    def stop_scan(self):
        self.scanning = False
        scan_info = "Сканування зупинено користувачем.\n"   #
Визначаємо змінну scan_info
        self.update_scan_tab(scan_info)

    def scan_directory(self, directory, rules):
        matches = []
        total_files  =  sum([len(files)  for  r,  d,  files  in
os.walk(directory)])
        scanned_files = 0

        with open("scan_errors.txt", "w") as error_file:
            for root, dirs, files in os.walk(directory):
                if not self.scanning:
                    break
                for file in files:
                    if not self.scanning:
                        break
                    file_path = os.path.join(root, file)
                    try:
                        match = rules.match(file_path)
                        if match:
                            matches.append((file_path, match))
                    except Exception as e:
                        error_file.write(f"Помилка          при
скануванні файлу {file_path}: {e}\n")

                    scanned_files += 1
```

```
                    progress = (scanned_files / total_files) *
100
                    self.progress_label.config(text=f"Прогрес:
{progress:.2f}%")
                    self.update_idletasks()

        self.progress_label.config(text="Сканування
завершено.")
        return matches

    def scan_for_malware(self, scan_path):
        rules_path     =     r'C:/Users/foxik/OneDrive/Рабочий
стол/ДИПЛОМ_МАГІСТР/malware-yara-main'

        if not os.path.exists(scan_path):
            scan_info = f"Шлях {scan_path} не існує.\n"
            self.update_scan_tab(scan_info)
            return []

        rules_files = self.collect_yara_files(rules_path)

        if not rules_files:
            scan_info = "Немає файлів .yara для компіляції.\n"
            self.update_scan_tab(scan_info)
            return []

        try:
            rules    =    yara.compile(filepaths={f'rule_{i}':
rules_files[i] for i in range(len(rules_files))})
        except yara.YaraSyntaxError as e:
            scan_info = f"Помилка компіляції правил Yara: {e}\n"
            self.update_scan_tab(scan_info)
            return []

        matches = self.scan_directory(scan_path, rules)
        self.display_results(matches)

    def scan_all_disks(self):
        rules_path     =     r'C:/Users/foxik/OneDrive/Рабочий
стол/ДИПЛОМ_МАГІСТР/malware-yara-main'

        rules_files = self.collect_yara_files(rules_path)

        if not rules_files:
```

```
            scan_info = "Немає файлів .yara для компіляції.\n"
            self.update_scan_tab(scan_info)
            return []

        try:
            rules    =    yara.compile(filepaths={f'rule_{i}':
rules_files[i] for i in range(len(rules_files))})
        except yara.Error as e:
            scan_info = f"Помилка компіляції правил Yara: {e}\n"
            self.update_scan_tab(scan_info)
            return []

        malware_matches = []

        for disk in 'ABCDEFGHIJKLMNOPQRSTUVWXYZ':
            if not self.scanning:
                break
            scan_path = f"{disk}:/"
            if os.path.exists(scan_path):
                matches = self.scan_directory(scan_path, rules)
                malware_matches.extend(matches)

        self.display_results(malware_matches)

    def display_results(self, matches):
        if matches:
            scan_info = "Шкідливе ПЗ виявлено!\n"
            for match in matches:
                short_path = os.path.basename(match[0])
                scan_info  +=  f"Файл:  {short_path},  Збіги:
{match[1]}\n"
        else:
            scan_info = "Шкідливе ПЗ не виявлено.\n"

        self.update_scan_tab(scan_info)

    #Processes and users
    def create_process_tab(self):
        try:
            # Створюємо рамку для процесів та користувачів
            process_frame  =  ttkb.LabelFrame(self.process_tab,
text="Процеси та користувачі")
            process_frame.pack(side="left",  padx=10,  pady=10,
fill="y", expand=False)
```

```
            # Поле з процесами та повзунком
            canvas = tk.Canvas(process_frame, width=400)    #
ширина поля з ползунками
            scrollbar      =      ttkb.Scrollbar(process_frame,
orient="vertical", command=canvas.yview)
            scrollable_frame = ttkb.Frame(canvas)

            scrollable_frame.bind(
                "<Configure>",
                lambda e: canvas.configure(
                    scrollregion=canvas.bbox("all")
                )
            )

            canvas.create_window((0,                          0),
window=scrollable_frame, anchor="nw")
            canvas.configure(yscrollcommand=scrollbar.set)

            scrollbar.pack(side="right", fill="y")
            canvas.pack(side="left", fill="both", expand=True)

            # Заповнення процесами та користувачами
            user_processes = {}
            for proc in psutil.process_iter(['pid', 'name',
'username']):
                user = proc.info['username']
                if user not in user_processes:
                    user_processes[user] = []
                user_processes[user].append(proc.info)

            for user, processes in user_processes.items():
                user_frame = ttkb.LabelFrame(scrollable_frame,
text=f"Користувач: {user}")
                user_frame.pack(padx=10, pady=10, fill="x")

                process_count = len(processes)
                cpu_usage = 0
                memory_usage = 0

                for proc in processes:
                    try:
                        p = psutil.Process(proc['pid'])
```

```
                        cpu_usage                                   +=
p.cpu_percent(interval=0.1) / psutil.cpu_count()
                        memory_usage += p.memory_info().rss
                    except                (psutil.NoSuchProcess,
psutil.AccessDenied, psutil.ZombieProcess):
                        continue

                user_info_label = ttkb.Label(user_frame,
                                             text=f"Процеси:
{process_count}\nВикористання                                  CPU:
{cpu_usage:.2f}%\nВикористання                               пам'яті:
{self.format_size(memory_usage)}")
                user_info_label.pack()

                # Розділяємо процеси за типами
                system_processes = [proc for proc in processes
if "system" in proc['name'].lower()]
                user_processes = [proc for proc in processes if
                                  "chrome"                      in
proc['name'].lower() or "firefox" in proc['name'].lower()]
                other_processes = [proc for proc in processes
if
                                   proc not in system_processes
and proc not in user_processes]

                # Відображаємо системні процеси
                system_frame   =   ttkb.LabelFrame(user_frame,
text="Системні процеси")
                system_frame.pack(padx=5, pady=5, fill="x")
                for proc in system_processes:
                    proc_label    =    ttkb.Label(system_frame,
text=f"PID: {proc['pid']}, Назва: {proc['name']}")
                    proc_label.pack()

                # Відображаємо процеси користувача
                user_proc_frame  =  ttkb.LabelFrame(user_frame,
text="Користувацькі процеси")
                user_proc_frame.pack(padx=5, pady=5, fill="x")
                for proc in user_processes:
                    proc_label  =  ttkb.Label(user_proc_frame,
text=f"PID: {proc['pid']}, Назва: {proc['name']}")
                    proc_label.pack()

                # Відображаємо інші процеси
```

```
                other_frame    =    ttkb.LabelFrame(user_frame,
text="Інші процеси")
                other_frame.pack(padx=5, pady=5, fill="x")
                for proc in other_processes:
                    proc_label    =    ttkb.Label(other_frame,
text=f"PID: {proc['pid']}, Назва: {proc['name']}")
                    proc_label.pack()

            # Додаємо блок із висновком
            summary_frame  =  ttkb.LabelFrame(scrollable_frame,
text="Зведення")
            summary_frame.pack(padx=10, pady=10, fill="x")
            summary_label      =      ttkb.Label(summary_frame,
text=self.get_summary())
            summary_label.pack()

            # Рамка для діаграми
            chart_frame   =   ttkb.LabelFrame(self.process_tab,
text="Використання системних ресурсів")
            chart_frame.pack(side="left",   padx=10,   pady=10,
fill="both", expand=True)  # Поміщаємо діаграму праворуч

            # Створення діаграми
            self.create_summary_chart()

            summary_chart_image                                =
tk.PhotoImage(file="summary_chart.png")
            summary_chart_label    =    ttkb.Label(chart_frame,
image=summary_chart_image)
            summary_chart_label.image = summary_chart_image
            summary_chart_label.pack(side="top",   fill="both",
expand=True)

            # Звужуємо блок із повзунком
            canvas.config(width=400)

            log_event("Вкладка Processes & Users створена")
        except Exception as e:
            log_error(f"Помилка  під  час  створення  вкладки
Processes & Users: {e}")

    def get_summary(self):
        cpu_usage = psutil.cpu_percent(interval=1)
        memory_usage = psutil.virtual_memory().percent
```

```
        if cpu_usage < 30 and memory_usage < 30:
            recommendation = "Ваш комп'ютер практично не
завантажений."
        elif 30 <= cpu_usage < 50 and 30 <= memory_usage < 60:
            recommendation = "Все повинно працювати добре."
        elif 60 <= cpu_usage < 80 or 60 <= memory_usage < 80:
            recommendation = "Ваш комп'ютер досить
завантажений."
        else:
            recommendation = "Можуть бути лаги, зависання,
комп'ютер практично повністю завантажений."

        summary = f"Загальне навантаження на
систему:\nВикористання CPU: {cpu_usage}%\nВикористання пам'яті:
{memory_usage}%\n\nРекомендації: {recommendation}"
        return summary

    def create_summary_chart(self):
        cpu_usage = psutil.cpu_percent(interval=1)
        memory_usage = psutil.virtual_memory().percent

        labels = ["Використання CPU", "Використання пам'яті"]
        sizes = [cpu_usage, memory_usage]
        colors = ["#ffcc5f", "#ff3d3d"]

        plt.figure(figsize=(6, 4))
        plt.bar(labels, sizes, color=colors)
        plt.ylim(0, 100)
        plt.xlabel("Ресурс")
        plt.ylabel("Використання (%)")
        plt.title("Використання системних ресурсів")
        plt.gca().spines["right"].set_visible(False)
        plt.gca().spines["top"].set_visible(False)
        plt.gca().yaxis.set_ticks_position("left")
        plt.gca().xaxis.set_ticks_position("bottom")
        plt.grid(axis="y", linestyle="--", linewidth=0.5)
        plt.tight_layout()
        plt.savefig("summary_chart.png")
    @staticmethod
    def format_size(size):
        power = 2 ** 10
        n = 0
        power_labels = {0: "", 1: "K", 2: "M", 3: "G", 4: "T"}
        while size > power:
```

```
            size /= power
            n += 1
        return f"{size:.2f} {power_labels[n]}B"

if __name__ == "__main__":
    app = BenchmarkApp()
    app.mainloop()
```