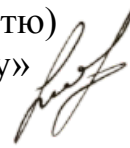


Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Зав. кафедри ПЗ, проф., д.т.н.

_____ О. Н. Романюк
(підпис)
«___» _____ 20__ р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу з дисципліни «Операційні системи»
студенту групи 2ПІ-22б ПІБ (повністю)
«Марисіку Даніїлу Миколайовичу»



Вихідні дані:

Розробити програмний додаток для ОС Windows, використовуючи мову програмування C++, IDE Microsoft Visual Studio

- 1 Провести аналіз сучасного стану питання та обґрунтування задачі:
 - 1.1 провести аналіз аналогів і їх основних характеристик;
 - 1.2 обґрунтувати вибір методу та засобу розробки.
- 2 Розробити програмний застосунок:
 - 2.1 розробити блок-схему і структурну схему застосунку;
 - 2.2 розробити структурну схему інтерфейсу;
 - 2.3 розробити графічний інтерфейс.
- 3 Розробити моделі програмного застосунку:
 - 3.1 розробити функціональний клас;
 - 3.2 розробити алгоритм роботи програмного застосунку і детальний алгоритм функціонального методу класу;
 - 3.3 розробити діаграму функціонального класу.
- 4 Провести тестування програмного застосунку:
 - 4.1 описати методики тестування;
 - 4.2 розробити інструкцію тестування програмного застосунку;
 - 4.3 розробити інструкцію користувача програмного застосунку.

Дата видачі «___» _____ 20__ р.

Керівник _____

Завдання отримав Марисик Д. М.

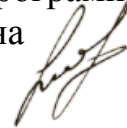
ЗАТВЕРДЖЕНО
Наказ Міністерства освіти і науки
України
29.03.2012 N 384

Форма N Н-6.01

Міністерство освіти і науки України
Вінницький національний технічний університет
Кафедра програмного забезпечення

КУРСОВА РОБОТА
з дисципліни «Операційні системи»
на тему: «Розробка програмного додатку для ОС Windows»

Студента 3-го курсу групи 2ПІ-22б
спеціальності 121 «Інженерія програмного забезпечення»
Марисика Даніїла Миколайовича



Керівник доцент кафедри ПЗ, Рейда О. М.
Кількість балів: _____ Оцінка: ECTS _____

Члени комісії

(підпис)

(прізвище та ініціали)

(підпис)

(прізвище та ініціали)

м. Вінниця – 2025 рік

ЗМІСТ

ВСТУП.....	4
1 АНАЛІЗ СУЧАСНОГО СТАНУ ПИТАННЯ ТА ОБҐРУНТУВАННЯ ЗАДАЧІ.....	5
1.1 Аналоги і їх основні характеристики.....	5
1.2 Обґрунтування вибору мови розробки застосунку.....	9
1.3 Обґрунтування середовища розробки застосунку.....	12
1.4 Висновки.....	16
2 РОЗРОБКА ПРОГРАМНОГО ЗАСТОСУНКУ.....	17
2.1 Обґрунтування вибору інтерфейсу.....	17
2.2 Блок-схеми застосунку і базових модулів.....	17
2.3 Графічна схема інтерфейсу.....	19
2.4 Висновки.....	22
3 РОЗРОБКА МОДЕЛІ ПРОГРАМНОГО ЗАСТОСУНКУ.....	23
3.1 Функціональний клас застосунку.....	23
3.2 Алгоритми роботи програмного застосунку і базових модулів.....	24
3.3 Висновки.....	25
4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСТОСУНКУ.....	26
4.1 Методики тестування.....	26
4.2 Методика тестування програмного застосунку.....	27
4.3 Інструкція користувача програмного застосунку.....	28
4.4 Висновки.....	34
ВИСНОВКИ.....	35
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ.....	36
Додаток А БЛОК-СХЕМИ АЛГОРИТМІВ.....	37
Додаток Б ЛІСТИНГ ПРОГРАМНОГО КОДУ.....	39

ВСТУП

Актуальність теми зумовлена потребою підвищення ефективності роботи операційної системи шляхом моніторингу та аналізу завантаження ЦП, ОЗП та мережевих ресурсів. Оптимальне використання цих ресурсів дозволяє запобігти перевантаженню, підвищити продуктивність системи та забезпечити стабільну роботу ПК.

Мета роботи полягає у розробці програмного застосунку аналізу даних і структур реєстру для підвищення ефективності роботи операційної системи, оптимізації ресурсів.

Об'єкт дослідження – процес розробки програмного застосунку аналізу завантаження ЦП, ОЗП й мережі.

Предмет дослідження – розробка програмного застосунку для аналізу завантаження ЦП, ОЗП й мережі.

В процесі розробки потрібно розв'язати такі задачі:

- обґрунтувати вибір методу та засобу розробки.
- розробити блок-схему і структурну схему застосунку;
- розробити структурну схему інтерфейсу;
- розробити графічний інтерфейс;
- розробити функціональний клас.

Потрібно досягти таких цілей:

- мова програмування і засіб розробки;
- блок-схема і структурна схема застосунку;
- структурна схема інтерфейсу;
- графічний інтерфейс;
- функціональний клас, що виконує основну задачу програмного застосунку.

Робота подана на 35 сторінках, складається з 4 розділів і містить 18 рисунків та 5 таблиць.

1 АНАЛІЗ СУЧАСНОГО СТАНУ ПИТАННЯ ТА ОБҐРУНТУВАННЯ ЗАДАЧІ

1.1 Аналоги і їх основні характеристики

В наш час тестування комплектуючих ПК є одним з ключових етапів перевірки, воно допомагає перевірити роботу всього обладнання від процесора (ЦП) до охолодження комп'ютера. Є багато аналогів, але всі вони мають або тяжкий для новачків інтерфейс, або не мають моніторингу основних ресурсів, тому було вирішено створити свій застосунок, який складе велику конкуренцію цим аналогам, та навіть буде кращим за них.

Аналоги програмного забезпечення

CPU-Z [1] — розробник CPUID, поточна версія 2.15 оновлена 17.03.2025 року. Утиліта видає докладні відомості про ЦП, а також деяку інформацію про материнську плату, оперативну пам'ять, встановлену в системі (див. рисунок 1.1).

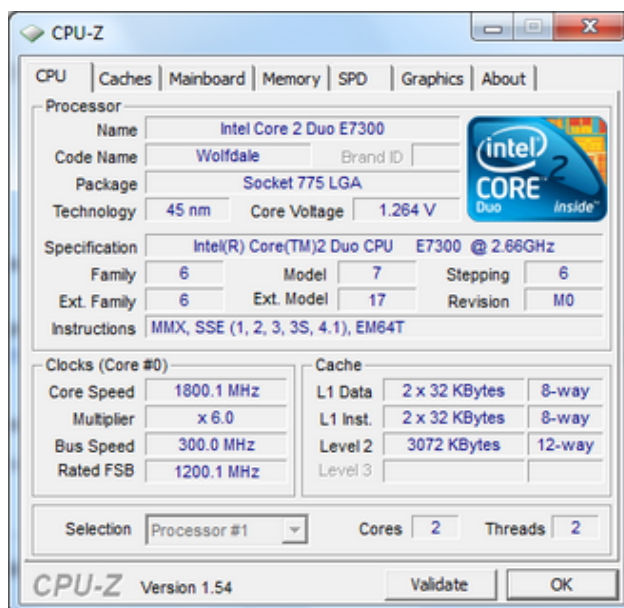


Рисунок 1.1 - Зображення головного вікна програми CPU-Z

В ній є вбудований тест процесора, який можна використовувати як для оцінки його продуктивності в порівнянні з іншими моделями, так і для стресового навантаження.

HWiNFO [2] — випущена компанією Diagnostic Software, поточна версія 8.22 випущена 26.02.2025 року. Також може розповісти деяку інформацію про ваше

«залізо», а ще може створювати звіти навідміну від попередньої програми, але ключовою фішкою програми є не це, а окреме вікно з датчиками. Дозволяє спостерігати за показниками температури, напруги та навантаження всіх компонентів комп'ютера — причому з фіксацією мінімальних, середніх та максимальних значень (див. рисунок 1.2).

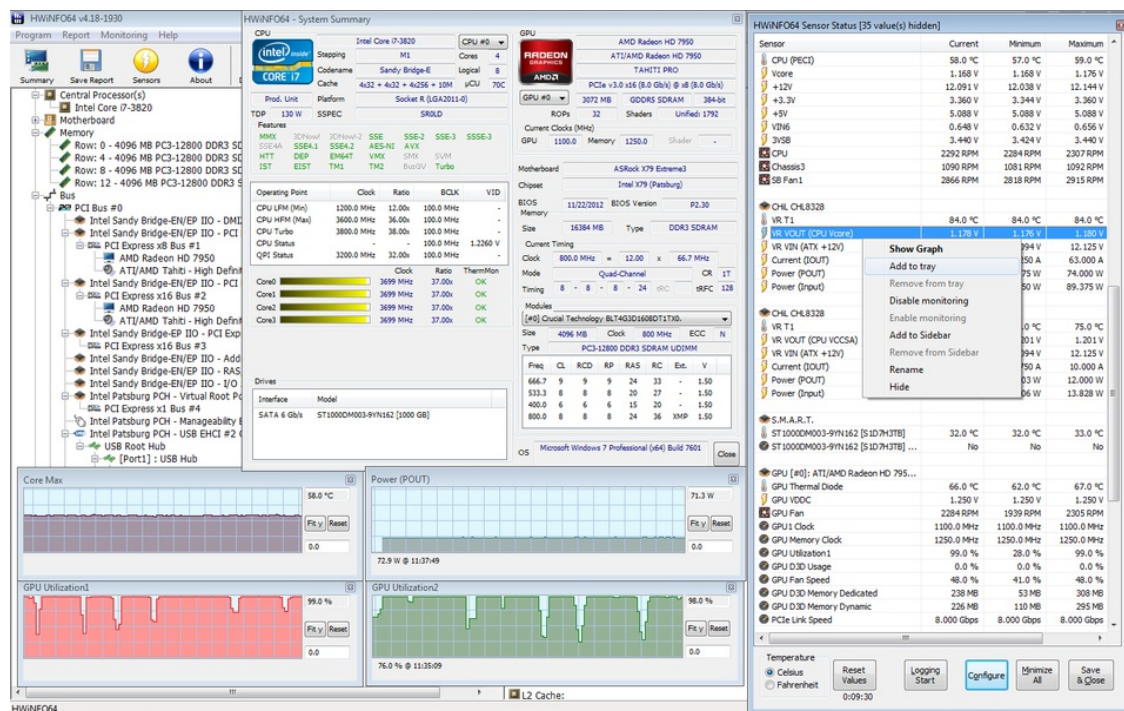


Рисунок 1.2 - Головне вікно програми HWiNFO

За допомогою інструмента можна стежити за цими значеннями в процесі робочого навантаження для виявлення відхилень температури та напруги від гранично допустимих.

SiSoftware Sandra [3] — системний аналізатор для 32- і 64-розрядних версій Windows, що включає в себе тестові та інформаційні модулі. Sandra об'єднує можливості для порівняння продуктивності як на високому, так і на низькому рівні випущена компанією SiSoftware, поточна версія 21, останнє оновлення 13.06.2023 року (див. рисунок 1.3).

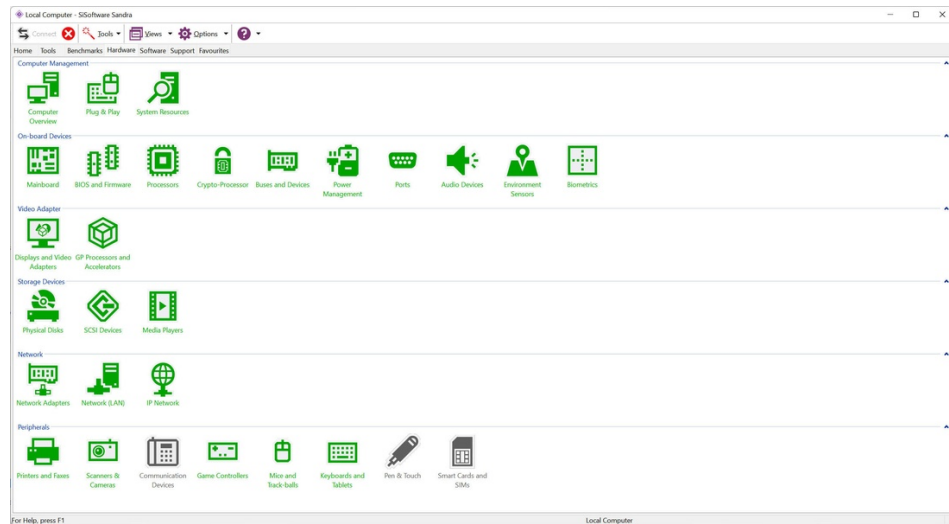


Рисунок 1.3 - Головне вікно програми SiSoftware Sandra

Здатна надати величезну кількість інформації, та інтерфейс не викликає незручності, є навіть категорії: обслуговування ПК, вбудовані пристрої тощо. Як можна наглядати, Sandra надає інформацію про температуру кожної лінії, скільки обертів вентилятора, температуру ЦП та різну іншу інформацію, як дату та час проведення тесту, напругу тощо.

TestPC (розроблюваний застосунок) — виробником являється студент ВНТУ Марисик Данііл Миколайович, групи 2ПІ-22б, поточна версія 1.0, випущена в 2025 році (див. рисунок 1.4).

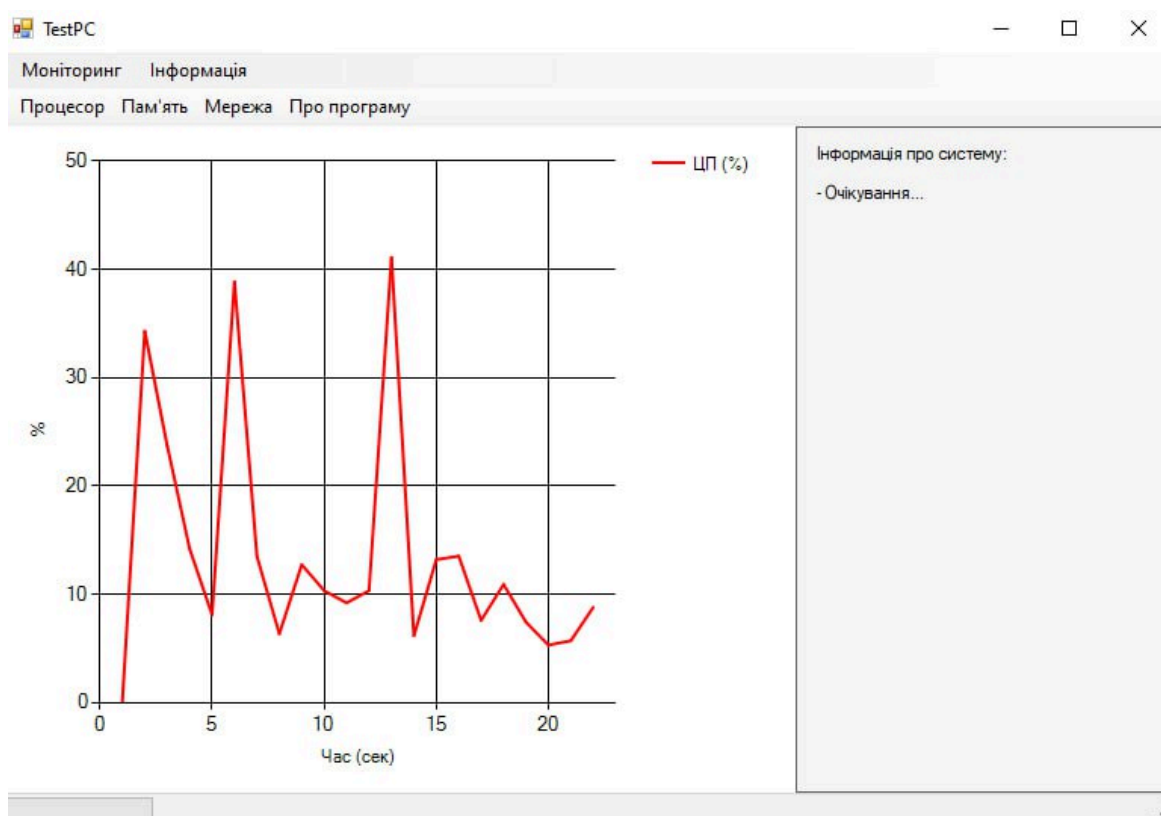


Рисунок 1.4 - Головне вікно програми TestPC

В даній програмі реалізована інформація про навантаження на ЦП, ОЗУ та мережу, показана вона у вигляді progress bar'a, відстежує ресурси, які завантажують ПК, та передають цю інформацію у процентах, також реалізована детальна інформація та моніторинг навантаження комп'ютера.

Давайте порівняєм аналоги з TestPC та вирішимо чи може новий застосунок скласти конкуренцію аналогам, для цього створимо таблицю (див. таблиця 1.1).

Таблиця 1.1 — Порівняння характеристик функціоналу аналогів

Характеристика	CPU-Z	HWiNFO	Sandra	TestPC
1	2	3	4	5
Графічна візуалізація (графік лінійний)	0	0	1	1
Моніторинг у реальному часі (постійний)	0	1	0	1

Продовження таблиці 1.1

1	2	3	4	5
Моніторинг завантаження пам'яті ОПУ	0	1	1	1
Стрес-тести (тестування продуктивності, навантаження)	1	0	1	0
Моніторинг мережі (візуалізація, завантаження, в реальному часі)	0	0	0	1
Сумарний коефіцієнт	1	2	3	4

На основі даної таблиці можемо сказати, що розроблюваний застосунок TestPC не тільки складає конкуренцію аналогам, а й навіть краще за них, і він може стати прекрасним інструментом для користувачів, які займаються моніторингом ресурсів.

1.2 Обґрунтування вибору мови розробки застосунку

При розробці програм на Windows можна використовувати багато різних мов програмування, але одні із основних це: Java, C# та C++. Розглянемо їх детальніше, а потім порівняємо їх між собою, щоб впевнитись яка з них є оптимальним варіантом для нашого застосунку.

Java [4] — об'єктно-орієнтована мова програмування, випущена 1995 року компанією «Sun Microsystems» як основний компонент платформи Java. З 2009 року мовою займається компанія «Oracle», яка того року придбала «Sun Microsystems». Поточна версія: Java 21 (останній випуск — вересень 2023). В офіційній реалізації Java-програми компілюються у байт-код, який при виконанні інтерпретується віртуальною машиною для конкретної платформи.

Переваги:

- Крос-платформенність: Код працює на будь-якій системі з JVM (Windows, Linux, macOS, Android).
- Безпека: Модель пісочниці, мінімізація ручного керування пам'яттю.
- Багата екосистема: Фреймворки (Spring, Hibernate), інструменти (Maven, Gradle), бібліотеки (Apache Commons).
- Підтримка багатопоточності: Вбудовані механізми для паралельних обчислень.

Недоліки:

- Витрати пам'яті: JVM та GC потребують додаткових ресурсів.
- Нижча продуктивність порівняно з C++/Rust через віртуалізацію.
- Версійні конфлікти: Зворотна сумісність іноді породжує проблеми.
- Складність для початківців: Синтаксис з перевантаженням методів, шаблонами.

C# [5] — це кросплатформна мова загального призначення, яка робить розробників продуктивними при написанні високопродуктивного коду. З мільйонами розробників C# є найпопулярнішою мовою .NET. C# має широку підтримку в екосистемі та всіх робочих навантаженнях .NET. На основі об'єктно-орієнтованих принципів він включає безліч функцій з інших парадигм, а не найменш функціонального програмування. Низькорівневі функції підтримують сценарії високої ефективності без написання небезпечного коду. Велика частина середовища виконання та бібліотек .NET написана на C#, і прогрес в C# часто використовується для всіх розробників .NET. Поточна версія: C# 12 (останній випуск — листопад 2023, разом з .NET 8). Розробник/Власник: Microsoft (розробляється як частина платформи .NET).

Переваги:

- Висока продуктивність: Оптимізований JIT-комп., можливість низькорівневого програмування.
- Багата екосистема: Доступ до бібліотек .NET (ASP.NET, Entity Framework, ML.NET), інтеграція з Visual Studio.

Недоліки:

- Залежність від екосистеми .NET: Поза межами .NET інструменти обмежені.

– Крива навчання: Складність для новачків через множину парадигм.

C++ [6] — компільована, статично типізована мова програмування загального призначення. Підтримує такі парадигми програмування, як процедурне програмування, об'єктно-орієнтоване програмування, узагальнене програмування. Мова має багату стандартну бібліотеку, яка включає в себе поширені контейнери та алгоритмами, введення-виведення, регулярні вирази, підтримку багатопотоковості та інші можливості. Поточна версія: C++23 (стандарт офіційно затверджено у 2023 році) Розробник/Власник: Стандартизується ISO (комітет ISO/IEC JTC1 SC22/WG21). Первісний автор — Б'ярне Страуструп (1985 р.).

Переваги:

- Прямий доступ до пам'яті, оптимізація під CPU.
- Відсутність накладних витрат (немає віртуальної машини, GC).
- Вручне керування пам'яттю, inline-асемблер, SIMD-інструкції.
- Контейнери (vector, map), алгоритми (sort, find).

Недоліки:

- Крута крива навчання (вказівники, копіювання/переміщення об'єктів).
- Переповнення буфера, витоки пам'яті, dangling pointers.
- Залежності часто збираються вручну (на відміну від Python/Pip [7] або C#/NuGet).

Тепер перейдемо до таблиці основних характеристик, вибору оптимальної мови для написання застосунку (див. таблиця 1.2).

Таблиця 1.2 — Порівняння характеристик мов програмування

	Java	C#	C++
1	2	3	4
Статична типізація	1	1	1
Використання оперативної пам'яті	0	1	1
Множинне наслідування	0	0	1

Продовження таблиці 1.2

1	2	3	4
Перевантаження функцій	1	1	1
Значення параметрів за замовчуванням	0	1	1
Сумарний коефіцієнт	2	4	5

Отже, порівняння показало, що C++ це оптимальний варіант для написання програмного застосунку TestPC.

1.3 Обґрунтування середовища розробки застосунку

Для розробки застосунку TestPC на C++ потрібно обрати ще середовище в якому будемо писати код, та створювати нашу програму, воно має включати в себе підтримку графічного редактору, серед таких можемо обрати: Microsoft Visual Studio, CLion та Microsoft Visual Studio Code (VS Code).

Microsoft Visual Studio [8] — це творчий стартовий майданчик, який можна використовувати для редагування, налагодження та складання коду, а також для публікації програми (див. рисунок 1.5).

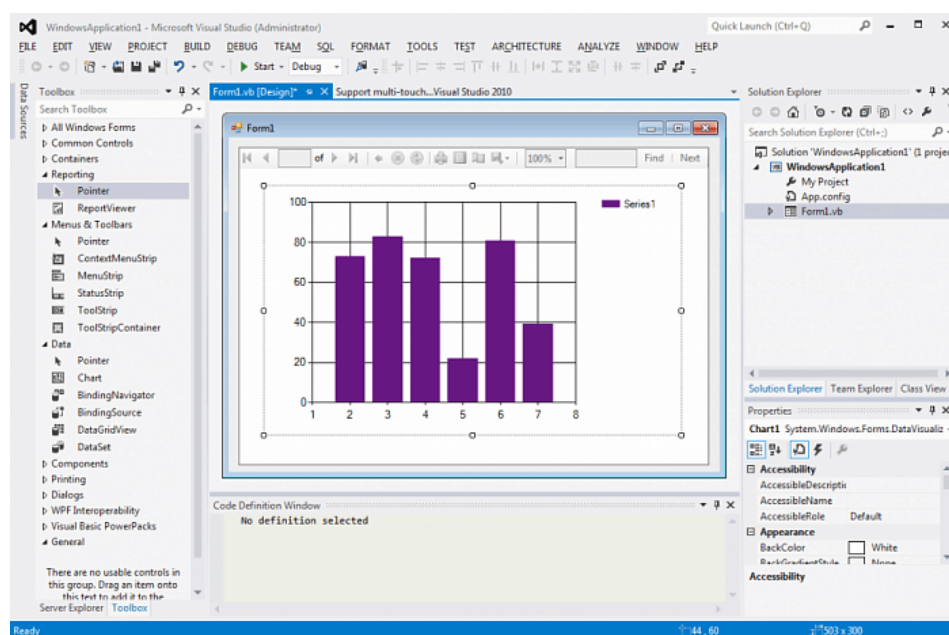


Рисунок 1.5 - Середовище розробки Visual Studio

На додаток до стандартного редактора та відладчика, що надаються більшістю інтегрованих середовищ розробки, Visual Studio включає компілятори, засоби завершення коду, графічні конструктори та багато інших функцій для покращення процесу розробки програмного забезпечення.

Переваги:

- є безкоштовна версія Visual Studio Community;
- вбудований інтерфейс командного рядка;
- API для підключення додаткових інструментів налаштування;
- повний набір інструментів розробника для створення та клонування Git репозиторіїв, управління гілками та вирішення конфліктів злиття прямо в інтегрованому середовищі розробки C++;
- великий набір доповнень розширення базової функціональності.

Недоліки:

- висока вартість платних версій Professional та Enterprise (від 45 доларів на місяць);
- високі вимоги до «заліза»;

CLion [9]— це більше, ніж просто редактор. IDE надає потужний відладчик і інструменти динамічного аналізу коду, що дозволяють швидко знаходити і усувати проблеми, підтримує Google Test, Boost.Test, Doctest і Catch для модульного тестування, інтегрується з популярними системами контролю версій та іншими інструментами (див. рисунок 1.6).

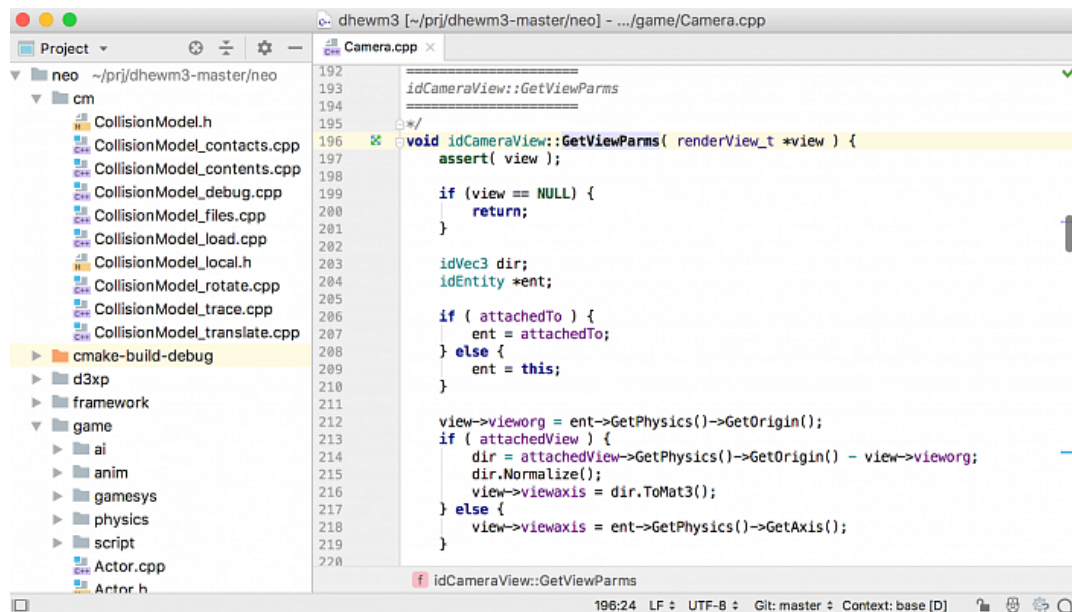


Рисунок 1.6 - Середовище розробки CLion

На відміну від Visual Studio, CLion не безкоштовне задоволення, але він набагато зручніший, надає безліч інструментів, які покращують та полегшують рутинні завдання

Переваги:

- зручні механізми налаштування програм;
- автозавершення коду ;
- підтримка VIM.

Недоліки:

- немає безкоштовної версії – лише демо на 30 днів;
- немає вбудованого компілятора;
- виникають проблеми зі встановленням компілятора.

VS Code [10] — це спрощений, але потужний редактор вихідного коду, який працює на комп'ютері і доступний для Windows, macOS і Linux (див. рисунок 1.7).

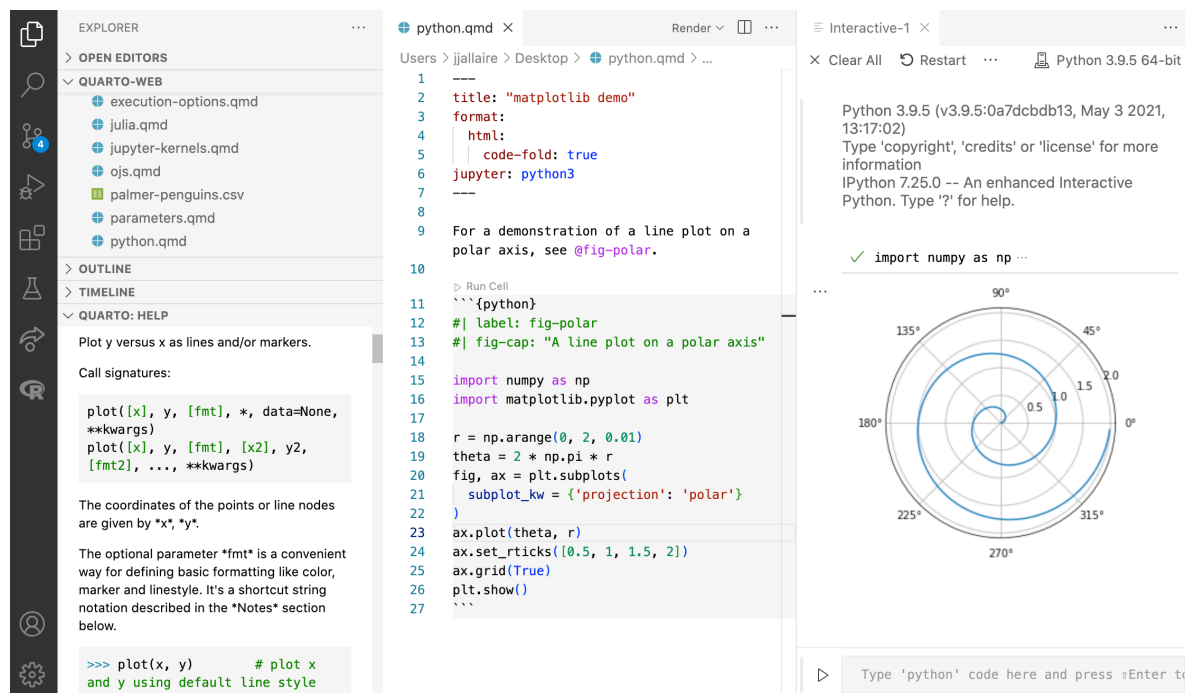


Рисунок 1.7 - Середовище розробки VS Code

Має вбудовану підтримку JavaScript, TypeScript і Node.js, а також велику екосистему розширень для інших мов і середовищ виконання (наприклад, C++, C#, Java, Python, PHP, Go).

Переваги:

- Легкий та швидкий – низьке навантаження на систему;
- Автозавершення коду (IntelliSense) для багатьох мов;
- Вбудований термінал – CLI без перемикання між вікнами.

Недоліки:

- Не повноцінна IDE – для складних проектів може знадобитися додаткове налаштування;
- Немає вбудованого компілятора;
- Обмежена підтримка дебагінгу – деякі мови потребують додаткових плагінів.

Створимо таблицю для порівняння середовищ та оберем найкращий варіант (див. таблиця 1.3).

Таблиця 1.3 — Порівняльний аналіз середовищ розробки

Критерії	Microsoft Visual Studio	CLion	VS Code
1	2	3	4
Debugger	1	1	0
Конструктор GUI	1	0	0
Інтегрований ланцюжок інструментів	1	1	0
Автозаповнення	1	1	1
Дизайн на основі GUI	1	0	0
Підтримка C++ за замовчуванням	1	1	0
Сумарний коефіцієнт	7	4	1

Отже, для нашої програми на мові програмування C++ найкраще середовище це — Microsoft Visual Studio, так як воно має все що нам потрібно, а саме головне має підтримку графічних застосунків на C++.

1.4 Висновки

Отже, було проведено аналіз та обґрунтовано вибір середовища для розробки застосунку TestPC це — Microsoft Visual Studio. Була обрана мова програмування C++, яка підтримує роботу з Windows API, та має все для вирішення поставлених задач. Це все дозволить нам створити ефективний та зручний для користувачів застосунок для моніторингу навантаження комп'ютера.

2 РОЗРОБКА ПРОГРАМНОГО ЗАСТОСУНКУ

2.1 Обґрунтування вибору інтерфейсу

Є чотири типи інтерфейсів: графічний (GUI), командний рядок (CLI), голосовий (VUI) та сенсорний. Так як у нас застосунок для операційної системи Windows мій вибір упав на графічний інтерфейс (GUI). Графічний інтерфейс користувача (GUI, Graphical User Interface) – це тип інтерфейсу, який дозволяє взаємодіяти з електронними пристроями або програмним забезпеченням за допомогою візуальних елементів, таких як вікна, кнопки, іконки, меню та інші графічні об'єкти. Основні елементи GUI: вікна, кнопки, меню, іконки, панелі інструментів, поля введення, прокрутка (Scrollbars), діалогові вікна.

Переваги такого інтерфейсу в його зручності, наприклад не потрібно запам'ятовувати команди, як у CLI, в ньому інтуїтивно зрозумілі елементи керування, а також його доступність, він підходить для широкого кола користувачів, включаючи новачків. Звісно є і недоліки наприклад як: GUI споживає більше оперативної пам'яті та потужності процесору.

2.2 Блок-схеми застосунку і базових модулів

Після вибору інтерфейсу, нам потрібно розробити блок-схеми нашого застосунку та його модулів, можемо для цього використати структурні схеми.

Розглянемо структурну схему нашого застосунку, на якій зображено всі модулі, які використовуються при роботі програмного застосунку (див. рисунок 2.1).

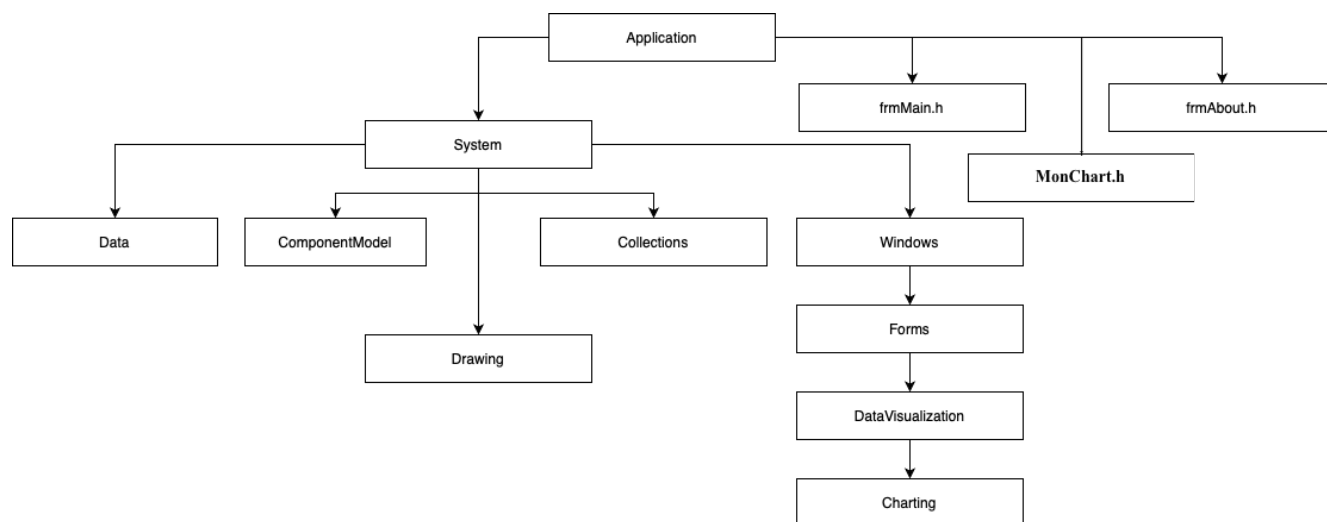


Рисунок 2.1 - Структурна діаграма застосунку

Далі розглянемо структурну схему нашого головного вікна `frmMain.h` на ній зображені всі елементи які будуть знаходитись в нашому інтерфейсі, серед них знаходяться випадаючі списки, кнопки, іконки, місце під графік навантаження, який буде відображати у вигляді лінійного графіку в областях від 0% до 100%, текстові поля в який буде відображатись додаткова інформація. Дана схема наглядно демонструє зв'язок між компонентами (див. рисунок 2.2).

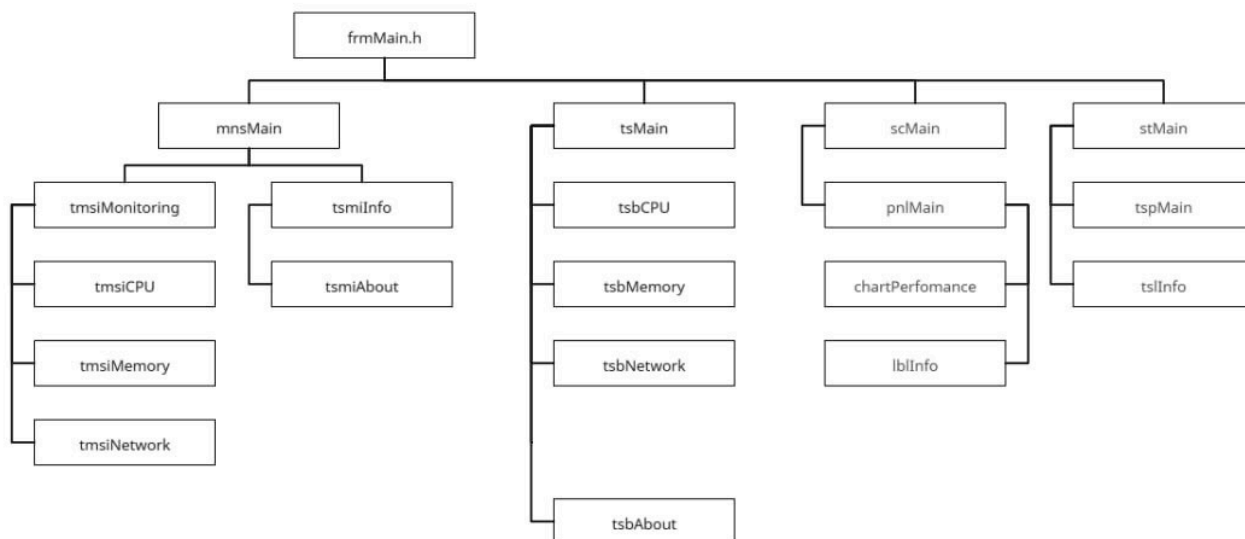


Рисунок 2.2 - Структурна діаграма головного вікна

Остання структурна схема це схема діалогового вікна «Про застосунок». На ній також показані всі елементи, які елементи мають знаходитись на цьому вікні (див. рисунок 2.3).

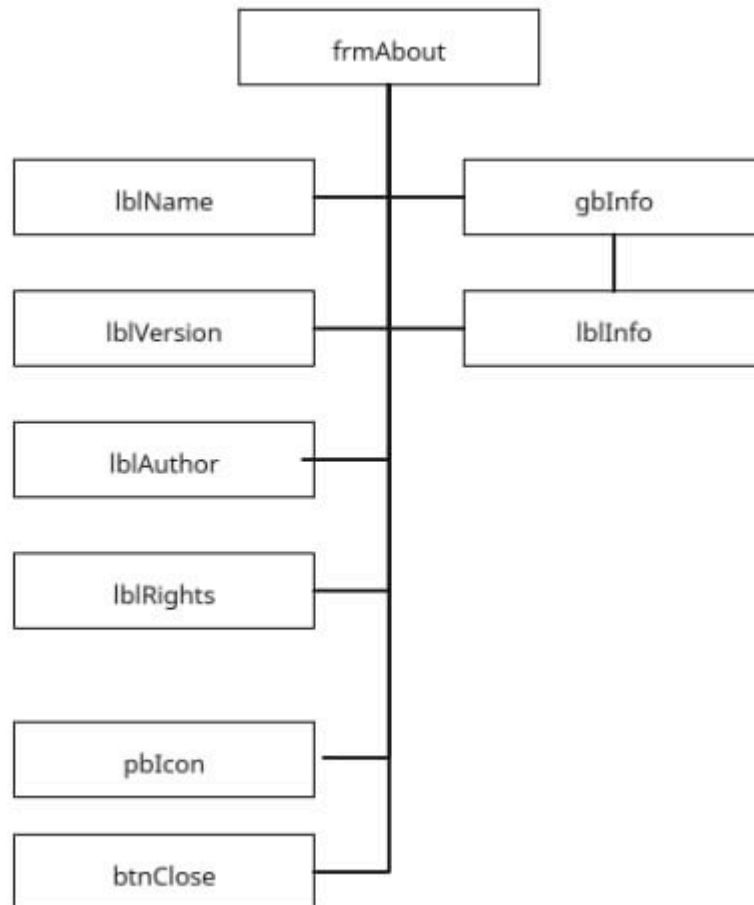


Рисунок 2.3 - Структурна схема форми “про застосунок”

В ньому є така інформація як: версія продукту, автор, права на цей застосунок, логотип та текстове поле в якому описані можливості застосунку та які функції він виконує

2.3 Графічна схема інтерфейсу

У нас є структурні схеми нашого застосунку, тепер потрібно зробити графічну схему того, як буде виглядати головний інтерфейс нашого застосунку, а також зробити схему нашого діалогового вікна «Про застосунок».

На даній схемі зображений графічний інтерфейс нашого програмного продукту, на ньому можна розглянути всі елементи, які він містить в собі (див. рисунок 2.4).

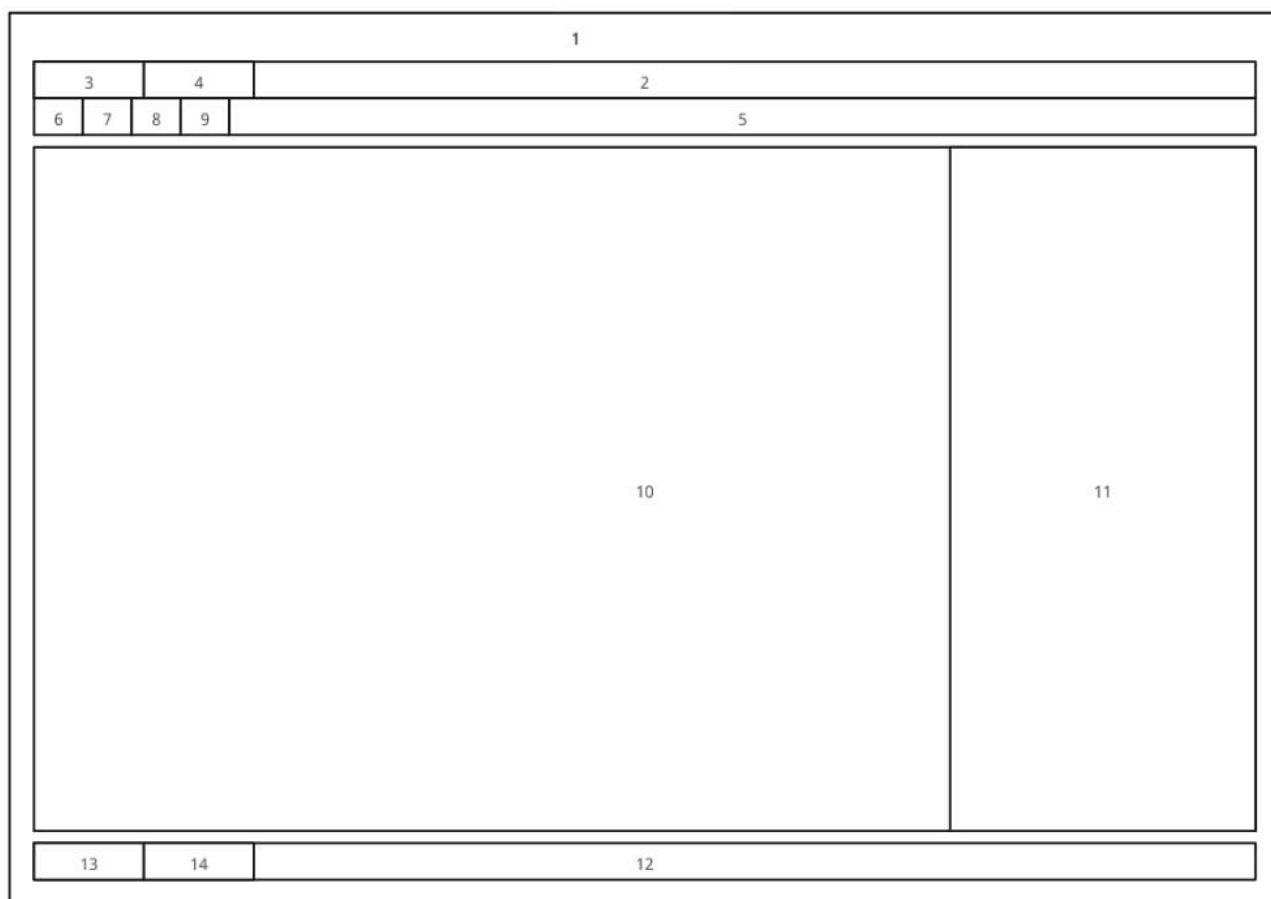


Рисунок 2.4 - Графічна схема головного вікна

Елементи головного вікна:

1. frmMain - головна форма на якій розташовується елемент управління і візуалізації
2. mnsMain - головне меню в якому містяться елементи управління роботою застосунку
3. tsmiMonitoring - випадаюче меню управління процесом моніторингу
4. tsmiInfo - випадаюче меню відображення пункту інформації і про застосунок
5. tsMain - панель швидкого доступу
6. tsbCPU - кнопка для відображення моніторингу CPU
7. tsbMemory - кнопка для відображення моніторингу Memory
8. tsbNetwork - кнопка для відображення моніторингу Network
9. tsbAbout - кнопка для відображення інформації «про застосунок»
10. chartPerfomance - графік завантаження

11. lblInfo - мітка що містить текстову інформацію
12. scMain - панель статусу що розташована внизу форми
13. tspMain - панель прогресу
14. tslInfo - текстове поле що виводить додаткову інформацію про компоненти управління

Діалогове вікно «Про застосунок» містить інформацію про застосунок та розробника, його графічну схему також можна розглянути, та побачити, які елементи, знаходяться там (див. рисунок 2.5).

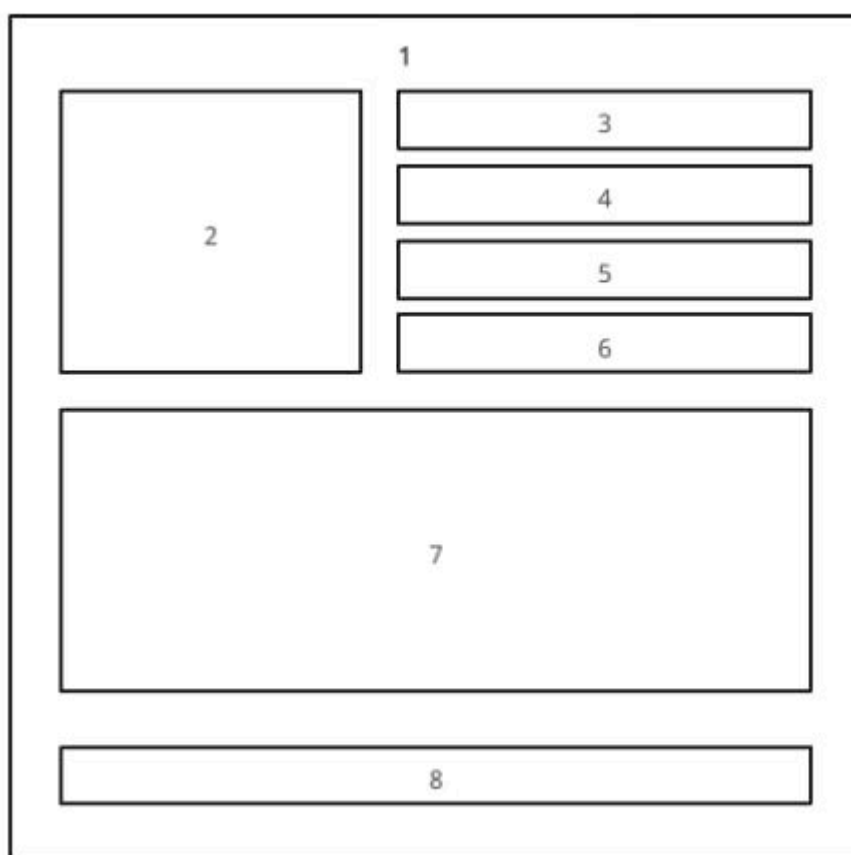


Рисунок 2.5 - Графічна схема вікна “про застосунок”

Елементи вікна «Про застосунок»:

1. frmAbout – форма «Про застосунок»
2. pbIcon – компонент із зображенням логотипу
3. lblName – текстовий компонент відображення назви застосунку
4. lblVersion - текстовий компонент відображення версії застосунку

5. lblAuthor - текстовий компонент відображення автора застосунку
6. lblRights - текстовий компонент відображення прав на застосунок
7. lblInfo - текстовий компонент відображення інформації про застосунок
8. btnClose – кнопка для закриття вікна

2.4 Висновки

В даному розділі було обґрунтовано вибір інтерфейсу, обрано графічний інтерфейс (GUI), також розглянули його переваги та недоліки. Створено п'ять схем, три з яких структурні інші дві графічні, на структурних схемах зображені наш застосунок, головне вікно та діалогове вікно «Про застосунок». Щодо графічних, на них зображений шаблон нашого головного інтерфейсу, та діалогового вікна. На них можна розглянути елементи, які вони містять та їх розташування.

3 РОЗРОБКА МОДЕЛІ ПРОГРАМНОГО ЗАСТОСУНКУ

3.1 Функціональний клас застосунку

Основним функціональним класом у застосунку TestPC є MonChart.h. Цей клас відповідає за моніторинг окремих ресурсів у реальному часі, а саме: ЦП, ОЗУ та мережу.

Клас забезпечує моніторинг цих ресурсів в реальному часі, з оновленням даних кожену секунду, що дозволяє користувачу отримувати актуальні показники використання ресурсів системи.

Основні методи класу:

- MonChart(Char[^] chart) – конструктор, який ініціалізує графік та лічильники продуктивності для ЦП і ОЗУ.
- SetMode(CharMode newMode) – встановлює режим моніторингу (ЦП, ОЗУ або мережа), оновлює графік та налаштовує нові параметри відображення.
- UpdateData(Object[^] sender, EventArgs[^] e) – метод для оновлення даних про ресурси системи в реальному часі.
- GetNetworkBytes(bool incoming) – отримує кількість байт, що надійшли або надіслані по мережі.

Діаграму функціонального класу можна розглянути на рисунку 3.1.

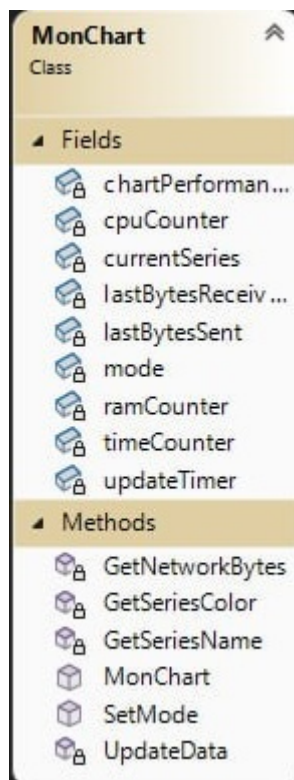


Рисунок 3.1 — Діаграма функціонального класу

На даній діаграмі функціонального класу MonChart можна побачити всі методи та поля, які реалізовані в цьому класі.

Методи класу включають основні операції для моніторингу ресурсів, такі як ініціалізація компонентів, оновлення даних в реальному часі та зміна режиму моніторингу (ЦП, ОЗУ, мережа).

Поля містять об'єкти для взаємодії з Windows API (наприклад, PerformanceCounter для моніторингу ЦП та ОЗУ, і Timer для оновлення даних), а також змінні для збереження останніх значень даних, таких як lastBytesReceived і lastBytesSent, які використовуються для обчислення мережевого трафіку.

3.2 Алгоритми роботи програмного застосунку і базових модулів

Тепер розглянемо основні алгоритми роботи програмного застосунку TestPC та його базових модулів, що відповідають за моніторинг системних ресурсів у реальному часі.

Почнемо з алгоритму роботи нашого застосунку. Його мета полягає в постійному відстеженні та оновленні показників використання ресурсів

комп'ютера, таких як ЦП, ОЗУ та мережа, з затримкою в одну секунду для кожного з ресурсів. Блок-схему можна розглянути на рисунку А.1.

На ній можемо розглянути послідовність дій, та які компоненти що відтворюють, наприклад: якщо ми натиснем на кнопку `tsmiCPU` або `tsbCPU`, то наша програма буде виводити графік навантаження нашого центрального процесору, те саме буде відбуватись якщо натиснем на `tsmiNetwork` або `tsmiMemory`.

Після блок-схеми роботи застосунку розглянемо й блок-схему алгоритму нашого функціонального класу (див. рисунок А.2).

Сама блок-схема описує алгоритм роботи `MonChart`. Основну логіку застосунку `TestPC`, на ній зображено кілька основних етапів для роботи всього алгоритму: ініціалізація, вибір ресурсу моніторингу, оновлення даних, оновлення графіка та повторення процесу .

3.3 Висновки

У цьому розділі ми розглянули основні аспекти роботи програмного застосунку `TestPC`, зокрема його алгоритм роботи. Також розглянули діаграму нашого функціонального класу та алгоритм який забезпечує моніторинг системних ресурсів у реальному часі та його основні етапи.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСТОСУНКУ

4.1 Методики тестування

На етапі розробки програмного забезпечення одним з ключових етапів є тестування, яке дозволяє виявити помилки у функціонуванні застосунку, оцінити стабільність його роботи та відповідність вимогам, визначеним на етапі проєктування.

Для розробленого програмного застосунку моніторингу центрального процесора, оперативної пам'яті та мережевого трафіку було застосовано кілька підходів до тестування з метою забезпечення його коректної роботи в реальних умовах експлуатації.

До основних методик тестування, які застосовувалися у межах даної роботи, належать:

- Модульне тестування, яке дозволяє перевірити окремі компоненти застосунку, зокрема класи та методи, що відповідають за збір системної інформації та її візуалізацію.
- Функціональне тестування, спрямоване на перевірку відповідності поведінки застосунку очікуваним результатам при виконанні основних дій користувача.
- Інтерфейсне тестування, що забезпечує перевірку зручності та коректності взаємодії користувача з елементами інтерфейсу.
- Ручне тестування — через спостереження за реакцією системи під час виконання ключових функцій (відображення графіків, перемикання режимів, обробка подій).

Застосування модульного підходу дозволяє комплексно оцінити якість роботи застосунку та виявити потенційні недоліки ще до передачі його кінцевим користувачам.

4.2 Методика тестування програмного застосунку

Для перевірки працездатності програмного застосунку було розроблено та реалізовано методику тестування, яка дозволяє виявити логічні помилки, збої у відображенні інформації та порушення у роботі графічного інтерфейсу.

Основною метою тестування є підтвердження коректної роботи:

- збору системних даних із використанням лічильників продуктивності Windows;
- відображення цих даних у вигляді графіків у реальному часі;
- перемикання між режимами моніторингу (ЦП, ОЗУ, мережа);
- взаємодії користувача з інтерфейсом програми.

У межах тестування було складено сім тест-кейсів, кожен з яких описує окрему дію користувача та очікувану реакцію застосунку. Тестування проводиться в ручну на ОС Windows 10 з використанням Task Manager для порівняння даних (див. таблицю 4.1).

Таблиця 4.1 - Перелік тест-кейсів

Ідентифікатор	Назва	Методика проведення тестування	Очікуваний результат	Результат
1	2	3	4	5
ТВ-1	Перевірка відображення головного вікна	1. Запустити застосунок, відкривши файл TestPC.exe.	1. Відображається головне вікно з логотипом, назвою програми та панеллю керування.	Виконано
ТВ-2	Перевірка графіків завантаження ЦП	1. У головному вікні перейти на вкладку «процесор».	1. Відображається графік реального часу із навантаженням на ЦП.	Виконано
ТВ-3	Перевірка графіків використання оперативної пам'яті (ОЗУ)	1. У головному вікні перейти на вкладку «пам'ять».	1. Відображається графік реального часу із використанням ОЗП.	Виконано

Продовження таблиці 4.1

1	2	3	4	5
ТВ-4	Перевірка відображення мережевої активності	1. У головному вікні перейти на вкладку «Мережа».	1. Відображаються графіки прийнятих та переданих байтів у реальному часі.	Виконано
ТВ-5	Перевірка оновлення показників у реальному часі	1. Залишити застосунок відкритим на 1 хвилину.	1. Дані на графіках оновлюються кожну секунду.	Виконано
ТВ-6	Перевірка роботи кнопки «Про застосунок»	1. Натиснути кнопку « Про застосунок» на панелі інструментів.	1. Відкривається файл або сторінка з інструкцією користувача.	Виконано
ТВ-7	Перевірка коректного завершення програми	1. Натиснути кнопку «X» у головному вікні.	1. Програма успішно закривається без помилок.	Виконано

Результати тестування показали, що всі основні функції працюють згідно з очікуваннями. Графіки оновлюються вчасно, перемикання між режимами виконується без помилок, інтерфейс реагує на дії користувача коректно.

4.3 Інструкція користувача програмного застосунку

1. Загальні відомості

Назва: TestPC

Версія: 1.0

Рік: 2025

Ця інструкція призначення для користування застосунком TestPC для моніторингу ЦП, ОЗУ та мережі.

Системні вимоги:

- Операційна система: Windows 10 / 11
- Процесор: x64, 1.8 GHz або вище
- Оперативна пам'ять: не менше 4 ГБ
- Вільне місце на диску: 100 МБ

- .NET Framework: версія 6.0 або новіша

2. Встановлення програми

1. Завантажте програмний застосунок з [посилання \(див. рисунок 4.1\)](#).
2. Встановіть програму, виконуючи інструкції на екрані.
3. Відкрийте файл TestPC.exe

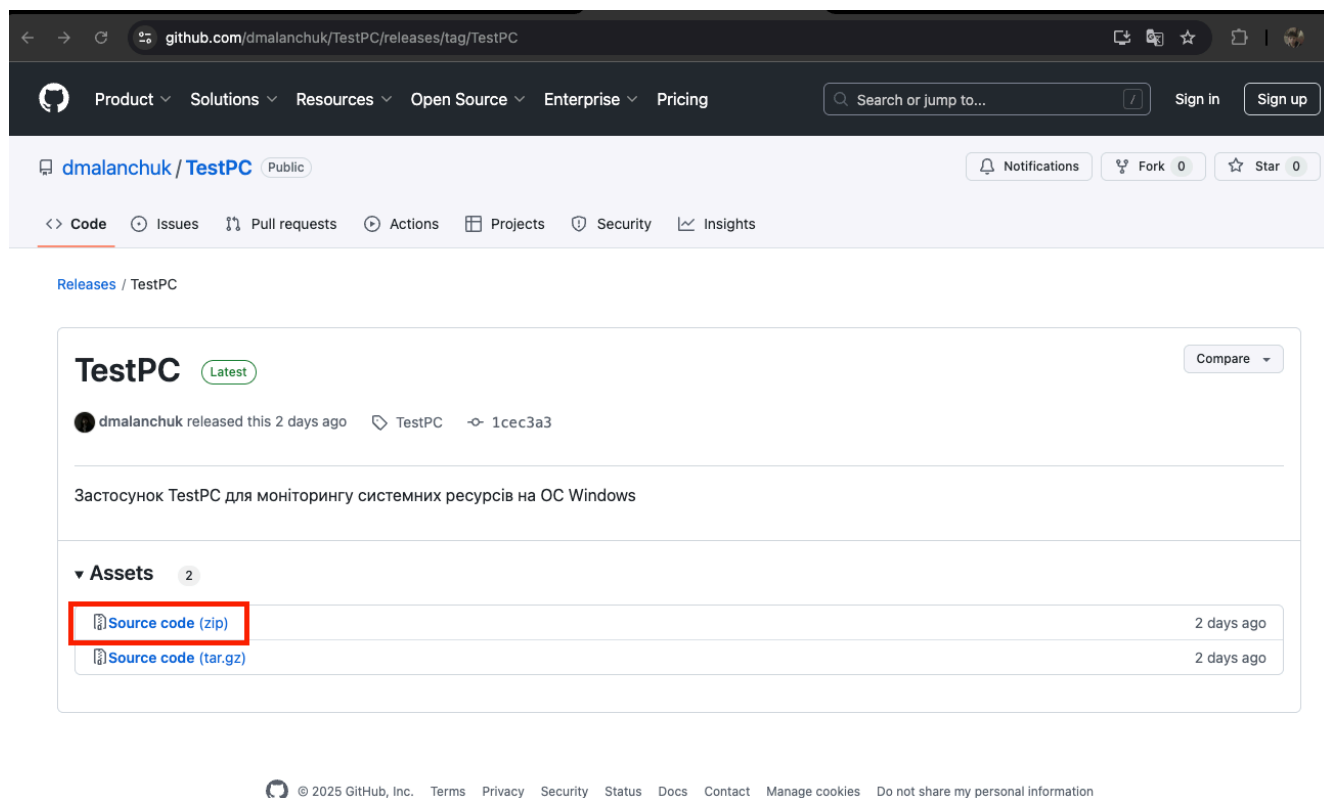


Рисунок 4.1 - Завантажувальний файл

3. Початок роботи

Відкрийте застосунок клацнувши два рази по TestPC.exe, авторизація не потрібна, програма зразу же почне працювати та моніторити навантаження ваших комплектуючих.

На головному вікні ви побачите такі кнопки як: процесор, пам'ять, мережа та про застосунок, також ви побачите графік, який буде відображати режим який ви ввімкнули (див. рисунок 4.2).

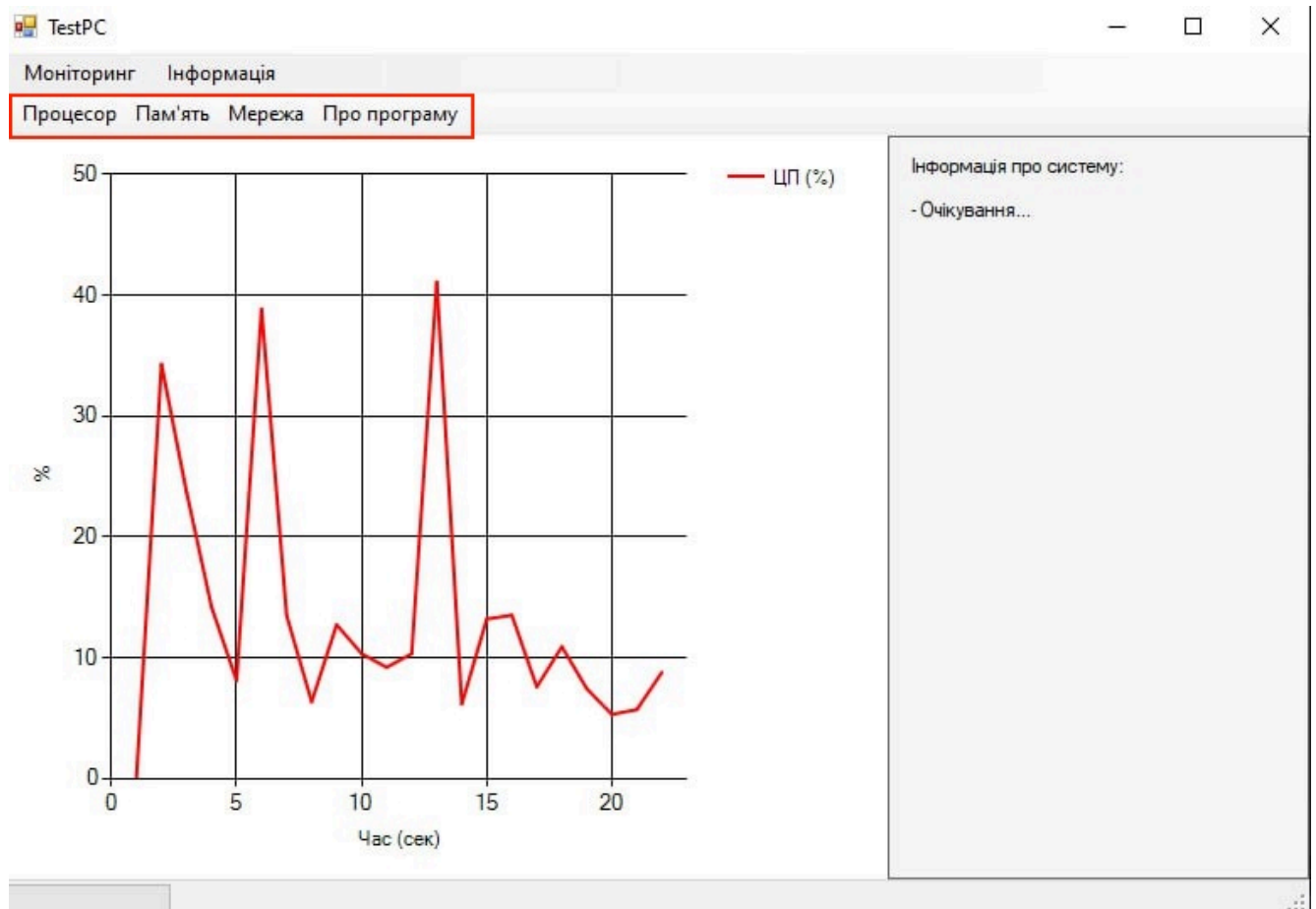


Рисунок 4.2 - Кнопки управління

4. Опис основних функцій та їх використання

Основні функції застосунку моніторинг ресурсів, на панелі управління є чотири кнопки, три з яких це режими: “процесор”, “пам’ять”, “мережа” та четверта “про програму” - показує опис програми, поточну версію та автора. Щоб завершити програму достатньо натиснути на “X” у правому верхньому куті. (див. рисунок 4.3).

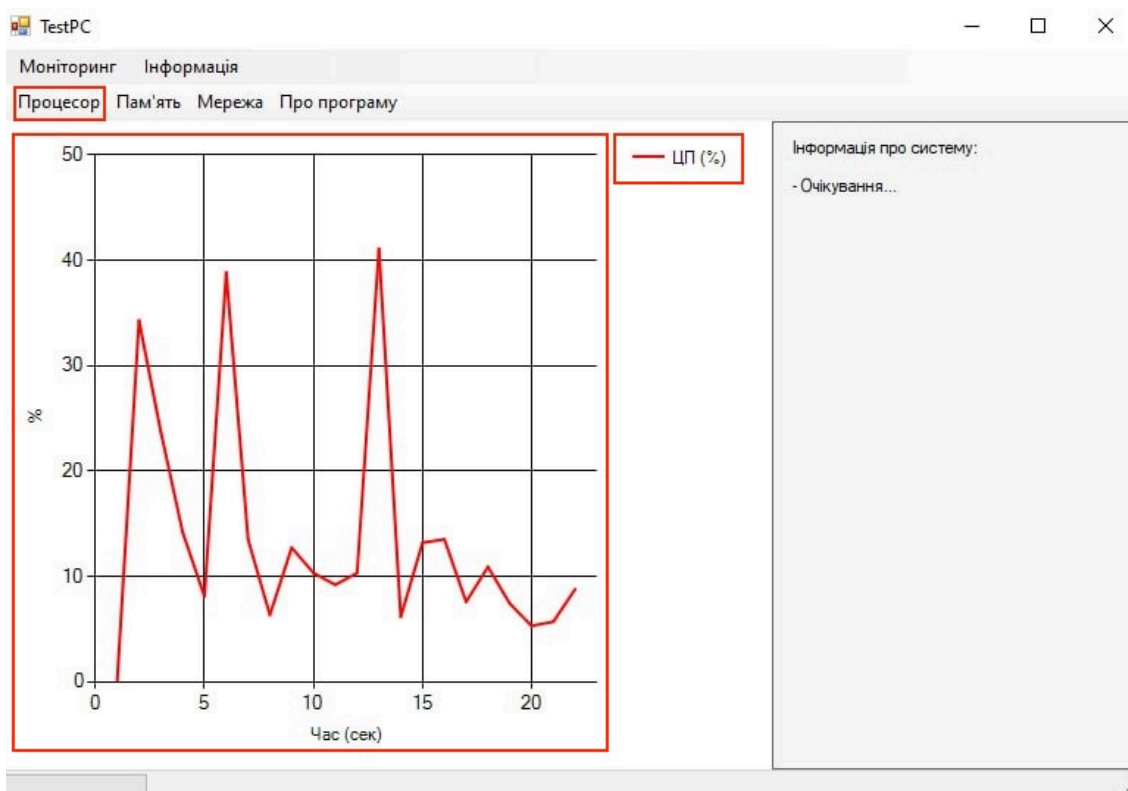


Рисунок 4.3 - Вікно з режимом “процесор”

На рисунку ми бачимо графік, який відображає завантаження ЦП у процентах. Режими виглядають ідентично, міняється тільки колір графіка, та одиниці вимірювання.

5. Вирішення проблем

У процесі роботи з програмою можуть виникати типові ситуації, пов’язані з налаштуванням операційної системи, відсутністю необхідних компонентів або збоєм у доступі до системних ресурсів. Створемо таблицю в якій будуть проблеми та спосіб їх вирішення (див. таблиця 4.2).

Таблиця 4.2 - найпоширеніші проблеми та способи їх вирішення.

Ідентифікатор	Проблема	Можлива причина	Рекомендація щодо вирішення
1	2	3	4
1	Програма не запускається	Відсутній .NET Framework	Встановити .NET 6.0 або новішу версію

Продовження таблиці 4.2

1	2	3	4
2	Графіки не оновлюються	Відсутній доступ до системних лічильників	Запустити програму з правами адміністратора
3	Значення навантаження завжди 0	Проблема з PerformanceCounter	Перевірити цілісність системних лічильників: lodctr /r через PowerShell
4	Вікно програми не відображається повністю	Низька роздільна здатність екрана	Збільшити розмір вікна вручну

Контактна інформація служби підтримки:

У разі виникнення складнощів, які неможливо вирішити самотійно, звертайтеся за електронною адресою: support@testpccomp.ua.

6. Оновлення програми

З метою підтримки стабільної та безпечної роботи застосунку, а також для впровадження нових функцій та виправлення виявлених помилок, рекомендується регулярно перевіряти наявність оновлень.

Процедура оновлення:

1. Закрийте запущену версію програми.
2. Видаліть стару версію або перезapiшіть файли новими.
3. Встановіть оновлення, виконавши TestPC_Setup.exe.
4. Запустіть оновлений застосунок.

FAQ – Часто задавані питання:

Питання:

Що робити, якщо програма після оновлення не запускається?

Відповідь:

Переконайтесь, що встановлено .NET Framework. Спробуйте запуск від імені адміністратора.

Питання:

Як дізнатись поточну версію застосунку?

Відповідь:

У меню Інформація → Про програму відображає поточну версію застосунку.

7. Відповіді на типові запитання користувачів

1. Чи потрібно встановлювати додаткове програмне забезпечення для роботи програми?

Ні. Для роботи застосунку необхідно лише, щоб у системі був встановлений .NET Framework 6.0 або новіший. Більшість сучасних версій Windows вже містять його за замовчуванням.

2. Чому на графіках не відображаються дані?

Можливо, програмі не надано доступ до системних лічильників. Спробуйте запустити застосунок від імені адміністратора. Натисніть на програму правою кнопкою миші і “запустити від імені адміністратора”.

3. Програма показує навантаження 0%, хоча система завантажена. Чому?

Це може бути пов'язано з некоректною роботою лічильника PerformanceCounter. Рекомендується відновити лічильники командою `lodctr /r` у PowerShell.

4. Чи є версія для Linux / macOS?

Ні, поточна версія програми розроблена винятково для Windows.

5. Чи можливо налаштувати інтервал оновлення графіків?

У базовій версії інтервал оновлення становить 1 секунду. У майбутніх оновленнях планується можливість гнучкого налаштування.

6. Як оновити програму до нової версії?

Потрібно вручну завантажити нову версію GitHub і встановити її, перезаписавши файли.

8. Додаткова інформація

Розробник: Марисик Даніїл Миколайович.

Група: 2ПІ-22б.

Контакти для зворотного зв'язку: <https://github.com/dmalanchuk>.

Довідкова секція

Операція: Запуск програми.

Опис Дії: Подвійний клік по ярлику або запуск TestPC.exe запускаю програму.

Операція: Увімкнення режиму “процесор”.

Опис Дії: Натисніть кнопку «Процесор» на панелі або в меню. Відобразиться червоний графік.

Операція: Увімкнення режиму “пам’ять”.

Опис Дії: Натисніть кнопку «пам’ять» на панелі або в меню. Відобразиться зелений графік вільної пам’яті.

Операція: Увімкнення режиму “мережа”.

Опис Дії: Натисніть кнопку «мережа» на панелі або в меню. Відобразиться синій графік, який показує швидкість прийому/передачі в КБ/с.

Операція: Перегляд інформації про програму.

Опис Дії: Натисніть кнопку «про програму» на панелі або в меню для відображення інформації про застосунок.

Операція: Завершення роботи.

Опис Дії: Натисніть кнопку «X» у правому верхньому куті, та завершіть роботу програми.

4.4 Висновки

У результаті тестування застосунку TestPC, було підтверджено його працездатність, стабільність роботи, та відповідність функціональним вимогам, всі тест-кейси були пройдені без помилок. Застосунок продемонстрував коректну роботу в усіх основних режимах: “процесор”, “пам’ять”, “мережа”. Перемикання між режимами відбулось швидко та без затримок.

ВИСНОВКИ

Під час написання курсової роботи, було написано програмний застосунок, який відстежує навантаження ЦП, ОЗУ та мережі комп'ютера. Для написання поставленого завдання, використовувалась мова програмування C++ та середовище Visual Studio від компанії Microsoft.

Також було розроблено блок-схеми роботи застосунку та функціонального класу. У першому розділі було розглянуто актуальність теми, та аналоги, проаналізовано їхні функціональні можливості.

У другому розділі було виконано розробку графічного інтерфейсу, створено дві графічні схеми для головного вікна та вікна про програму, а також розроблені три структурні схеми.

У третьому розділі було створено функціональний клас MonChart.h в якому описана вся логіка роботи застосунку TestPC, моніторинг відбувається в реальному часі, відображення нової точки на графіку малюється кожну секунду.

У четвертому розділі проведено тестування програмного застосунку. Була розроблена методика тестування, створено 10 тестових випадків, які перевірили всі основні сценарії роботи програми. Застосунок пройшов усі тести успішно. Крім того, складено повну інструкцію користувача, яка включає загальну інформацію, опис встановлення, використання, довідкові матеріали та рекомендації з вирішення типових проблем.

У ході курсової роботи було виконано всі поставлені задачі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Головна сторінка CPU-Z: <https://www.cpubid.com/software/cpu-z.html>.
2. Головна сторінка HWiNFO: <https://www.hwinfo.com>.
3. Офіційний сайт SiSoftware Sandra: <https://sisoftware-sandra-download.com>.
4. Документація по Java: <https://dev.java/learn/>.
5. Документація по C#: https://learn.microsoft.com/uk-ua/dotnet/csharp/tour-of-csharp/?utm_source=chatgpt.com.
6. Документація по C++: <https://learn.microsoft.com/en-us/cpp/?view=msvc-170>.
7. Документація по Python/Pip: <https://pypi.org/project/pip/>.
8. Джерело для завантаження Microsoft Visual Studio: <https://visualstudio.microsoft.com/>.
9. Джерело для завантаження Clion: <https://www.jetbrains.com/ru-ru/clion/>.
10. Джерело для завантаження Visual Studio Code: <https://code.visualstudio.com/docs/?dv=win64user>.

Додаток А

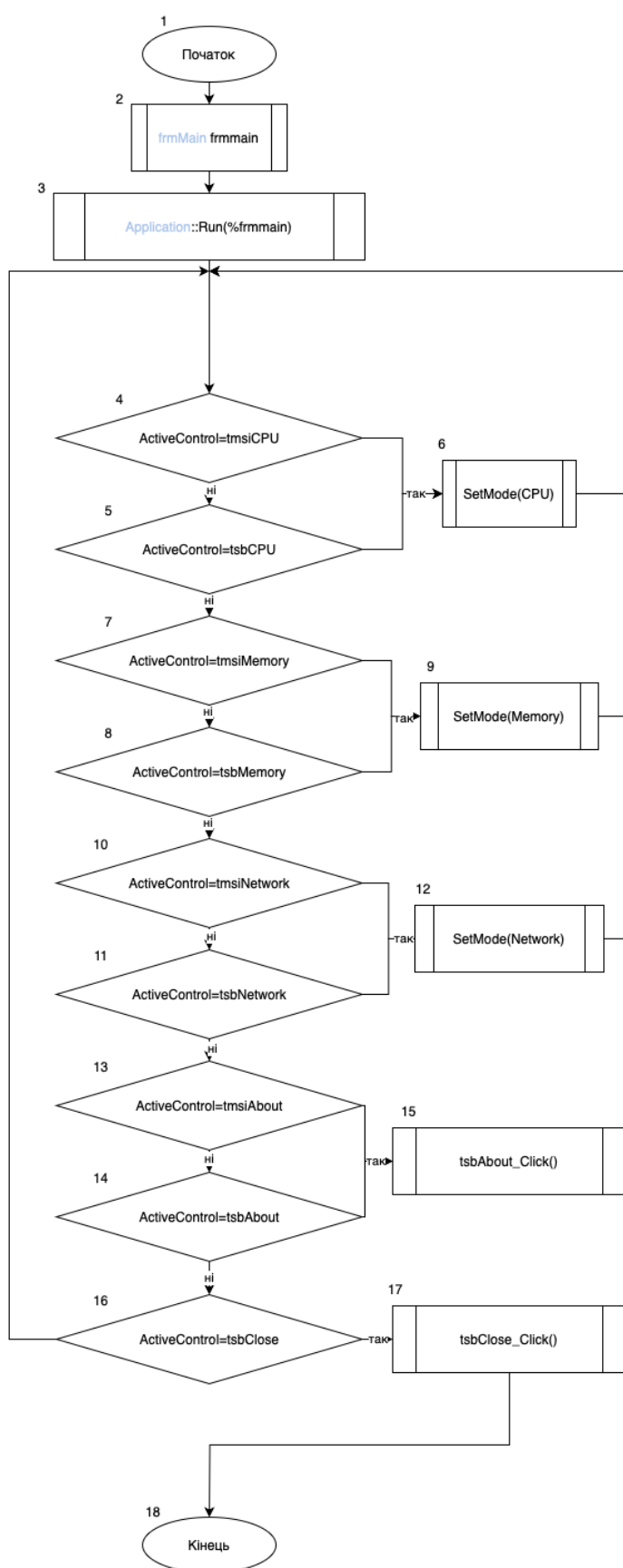


Рисунок А.1 - Блок-схема алгоритму роботи застосунку



Рисунок А.2 - Алгоритм роботи функціонального класу

Додаток Б

```

#pragma once
#include <Windows.h>
#include <iphlpapi.h>
#pragma comment(lib, "iphlpapi.lib")

namespace Monitoring {
    using namespace System;
    using namespace System::Diagnostics;
    using namespace System::Windows::Forms;
    using namespace System::Windows::Forms::DataVisualization::Charting;

    public enum class ChartMode { CPU, RAM, Network };

    public ref class MonChart {
    private:
        Chart^ chartPerformance;
        Timer^ updateTimer;
        PerformanceCounter^ cpuCounter;
        PerformanceCounter^ ramCounter;

        UInt64 lastBytesReceived;
        UInt64 lastBytesSent;

        Series^ currentSeries;
        ChartMode mode;
        int timeCounter;

    public:
        MonChart(Chart^ chart) {
            this->chartPerformance = chart;
            this->cpuCounter = gcnew PerformanceCounter("Processor", "% Processor
Time", "_Total");
            this->ramCounter = gcnew PerformanceCounter("Memory", "Available
MBytes");
            this->timeCounter = 0;
            this->mode = ChartMode::CPU;

            this->updateTimer = gcnew Timer();
            this->updateTimer->Interval = 1000; // 1 секунда
            this->updateTimer->Tick += gcnew EventHandler(this,
&MonChart::UpdateData);

```

```

        this->lastBytesReceived = GetNetworkBytes(true);
        this->lastBytesSent = GetNetworkBytes(false);
    }

    void SetMode(CharMode newMode) {
        this->mode = newMode;
        this->timeCounter = 0;
        this->chartPerformance->Series->Clear();
        this->chartPerformance->ChartAreas->Clear();
        this->chartPerformance->Legends->Clear();

        ChartArea^ chartArea = gcnew ChartArea("MainArea");
        chartArea->AxisX->Title = "Час (сек)";
        chartArea->AxisY->Title = (mode == ChartMode::RAM) ? "МБ" : ((mode ==
ChartMode::Network) ? "KB/s" : "%");
        chartArea->BackColor = System::Drawing::Color::White;
        this->chartPerformance->ChartAreas->Add(chartArea);

        this->currentSeries = gcnew Series(GetSeriesName());
        this->currentSeries->ChartType = SeriesChartType::Line;
        this->currentSeries->BorderWidth = 2;
        this->currentSeries->Color = GetSeriesColor();
        this->chartPerformance->Series->Add(currentSeries);
        this->chartPerformance->Legends->Add(gcnew Legend("Legend1"));

        if (mode == ChartMode::Network) {
            this->lastBytesReceived = GetNetworkBytes(true);
            this->lastBytesSent = GetNetworkBytes(false);
        }

        this->updateTimer->Start();
    }
private:
    String^ GetSeriesName() {
        switch (this->mode) {
            case ChartMode::CPU: return "ЦП (%)";
            case ChartMode::RAM: return "Вільна ОЗУ (МБ)";
            case ChartMode::Network: return "Мережа (KB/s)";
        }
        return "Невідомо";
    }

    System::Drawing::Color GetSeriesColor() {
        switch (this->mode) {
            case ChartMode::CPU: return System::Drawing::Color::Red;

```



```

    case ChartMode::RAM: return System::Drawing::Color::Green;
    case ChartMode::Network: return System::Drawing::Color::Blue;
    }
    return System::Drawing::Color::Black;
}

void UpdateData(Object^ sender, EventArgs^ e) {
    float value = 0.0f;

    if (this->mode == ChartMode::CPU) {
        value = this->cpuCounter->NextValue();
    }
    else if (this->mode == ChartMode::RAM) {
        value = this->ramCounter->NextValue();
    }
    else if (this->mode == ChartMode::Network) {
        UInt64 currentBytesReceived = GetNetworkBytes(true);
        UInt64 currentBytesSent = GetNetworkBytes(false);

        float netInKB = float(currentBytesReceived - lastBytesReceived) / 1024.0f;
        float netOutKB = float(currentBytesSent - lastBytesSent) / 1024.0f;
        value = netInKB + netOutKB; lastBytesReceived = currentBytesReceived;
        lastBytesSent = currentBytesSent;
    }

    timeCounter++;
    currentSeries->Points->AddXY(timeCounter, value);

    if (currentSeries->Points->Count > 60) {
        currentSeries->Points->RemoveAt(0);
    }

    chartPerformance->ResetAutoValues();
}

UInt64 GetNetworkBytes(bool incoming) {
    PMIB_IFTABLE pIfTable;
    DWORD dwSize = 0;
    GetIfTable(NULL, &dwSize, FALSE);
    pIfTable = (MIB_IFTABLE*)malloc(dwSize);
    if (GetIfTable(pIfTable, &dwSize, FALSE) == NO_ERROR) {
        UInt64 totalBytes = 0;
        for (DWORD i = 0; i < pIfTable->dwNumEntries; i++) {
            MIB_IFROW row = pIfTable->table[i];

```

```
        if (row.dwType == IF_TYPE_ETHERNET_CSMACD || row.dwType ==  
IF_TYPE_IEEE80211) {  
            totalBytes += incoming ? row.dwInOctets : row.dwOutOctets;  
        }  
    }  
    free(pIfTable);  
    return totalBytes;  
}  
free(pIfTable);  
return 0;  
}  
};  
}
```