ІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»

Факультет комп'ютерних наук та технологій
Кафедра комп'ютерних систем та мереж

Пояснювальна записка

до дипломного проєкту (роботи)
бакалавра
(ступінь вищої освіти (освітній ступінь))

на тему «РОЗРОБКА СИСТЕМИ ДЛЯ ОЦІНЮВАННЯ ПРОДУКТИВНОСТІ ПЕРСОНАЛЬНОГО КОМП'ЮТЕРА»

Виконав: студент 4 курсу, групи КНТз-512сп
спеціальності 123 Комп'ютерна інженерія

Освітня програма (спеціалізація)
Комп'ютерна інженерія
(код і назва спеціальності)

КІКЄЄВА В.С..
(прізвище та ініціали)
Керівник ІЛЬЯШЕНКО М.Б..
(прізвище та ініціали)
Рецензент ЄФИМЕНКО М.В.
(прізвище та ініціали)

2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет «Запорізька політехніка»

(повне найменування вищого навчального закладу)

Факультет Комп'ютерних наук і технологій

Кафедра «Комп'ютерні системи та мережі»

Ступінь вищої освіти (освітній ступінь) бакалаврський

Спеціальність 123 Комп'ютерна інженерія

(код і назва)

Освітня програма (спеціалізація) Комп'ютерна інженерія

(назва)

ЗАТВЕРДЖУЮ
Зав. кафедри
_____ Кудерметов Р.К.
"14" квітня 2025 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ СТУДЕНТУ

КІКЄЄВА Вікторія Станіславівна

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) «Розробка системи для оцінювання продуктивності персонального комп'ютера»

керівник проекту (роботи) ІЛЬЯШЕНКО М.Б.., к. т. н., доцент

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від "08" квітня 2025 року № 151

2. Строк подання студентом проекту (роботи) 01.06. 2025 року

3. Вихідні дані до проекту (роботи) .NET Framework 4.6.1, ToroBench, CrystalDiskMark

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1) Опис предметної області

2) Розробка вимог до програмного забезпечення

3) Реалізація комп'ютерної системи

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Слайди презентації

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-3	ІЛЬЯШЕНКО М.Б., к.т.н.. доцент		
Нормоконтроль	ЩЕРБАК Н.В., ст. викладач		

7. Дата видачі завдання 05.05. 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Опис предметної області	05.04.2025	
2	Види та засоби тестування	10.05.2025	
3	Технології .NET Framework	17.04.2025	
4	Тестування процесорів	23.04.2025	
5	Обрання програмного забезпечення	18.05.2025	
6	Реалізація комп'ютерної системи	25.05.2025	
7	Результати роботи системи	26.05.2025	
9	Захист роботи	.06.2025	

Студент _____ **Вікторія КІКЄЄВА**
(підпис) (ініціали та прізвище)

Керівник проекту (роботи) _____ **Матвій ІЛЬЯШЕНКО**
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
58 с., 25 рис., 3 табл., 1 дод., 18джерел.

OPENCL, TOROBENCH, БЕНЧМАРК, НАКОПИЧУВАЧІ, СТРЕС-ТЕСТИ, ТРОТЛІНГ, ЦЕНТРАЛЬНИЙ ПРОЦЕСОР

Метою даної роботи є створення програмного продукту для швидкого та зручного тестування продуктивності персональних комп'ютерів, який також містить функціонал онлайн-рейтингу для порівняння результатів з іншими користувачами. Основним завданням є розробка безкоштовного бенчмарку з простим інтерфейсом та швидким тестуванням продуктивності ПК, орієнтованого на операційну систему Windows. Станом на сьогодні, бенчмарки набули широкої популярності, особливо у сферах, де швидкодія комп'ютера має вирішальне значення для користувацького досвіду.

Новизна роботи полягає в акценті на значущості бенчмарків у процесі розробки та тестування програмного забезпечення - зокрема, у галузях комп'ютерної графіки та інших високопродуктивних застосунків. Бенчмарки слугують інструментом для розробників з метою оцінки ефективності, а також оптимізації своїх рішень на різних апаратних конфігураціях.

ABSTRACT

Explanatory note to the master’s work: 57 p., 25 figures, 3 tables, 1 дод., 18 sources.

OPENCL, TOROBENCH, BENCHMARK, DRIVES, STRESS TESTS, CENTRAL PROCESSOR

The purpose of this paper is to create a software product for fast and convenient testing of personal computer performance, which also contains online rating functionality for comparing results with other users. The main task is to develop a free benchmark with a simple interface and fast PC performance testing focused on the Windows operating system. As of today, benchmarks have gained widespread popularity, especially in areas where computer performance is crucial to the user experience.

The novelty of this work is the emphasis on the importance of benchmarks in the process of software development and testing - in particular, in the fields of computer graphics and other high-performance applications. Benchmarks serve as a tool for developers to evaluate the effectiveness and optimise their solutions on different hardware configurations.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ПК	-	Personal computer, Персональний комп’ютер
CPU	-	Central processing unit, Центральний процесор
GPU	-	Graphics processing unit, Графічна карта
GUI	-	Graphical User Interface, Графічний інтерфейс користувача
HDD	-	Hard disk drive, Жорсткий диск
SSD	-	Solid-state drive, Твердотільний накопичувач
RAM	-	Random Access Memory, Оперативна пам’ять

ВСТУП

З появою комп’ютерної техніки життя людства зазнало докорінних змін. Із часів друкарських машинок процес обробки та управління інформацією став значно простішим. Комп’ютери взяли на себе дедалі більше обчислювальних завдань, що сприяло стрімкому технічному прогресу. Наприкінці XX століття у багатьох сферах діяльності, де раніше ключову роль відігравала людина, її участь значно зменшилася.

Сьогодні, у зв’язку з постійним зростанням обчислювальної потужності комп’ютерів, одночасно ускладнюються й розширюються завдання, які на них покладаються. Проте не всі користувачі мають доступ до сучасного обладнання, що створює проблему недостатньої продуктивності при виконанні ресурсомістких задач.

Для аналізу взаємозв’язку між зростаючими потребами користувачів і потужністю персональних комп’ютерів у сфері IT були створені спеціальні програми - бенчмарки, які тестують продуктивність системи в умовах інтенсивного навантаження. Перед придбанням програмного забезпечення користувачі орієнтуються на мінімальні й рекомендовані системні вимоги, що визначаються саме через бенчмарки. Представники різних професій, пов’язаних з IT - від програмістів до геймерів і відеомонтажерів - звертають увагу на відповідність своєї системи технічним вимогам.

Метою даної роботи є розробка простого та швидкого інструменту для перевірки продуктивності ПК із функцією онлайн-рейтингу для порівняння результатів. Це дозволить більш детально дослідити питання продуктивності сучасних систем.

програмне забезпечення включає такі функції:

- вимірювання швидкості виконання арифметичних, текстових та інших операцій центрального процесора;

- тестування графічного процесора (GPU) під навантаженням;

- аналіз температури для виявлення тротлінгу;
- перевірка швидкості інтернет-з'єднання;
- тест швидкості запису та зчитування даних з накопичувачів;
- публічний рейтинг результатів тестів;
- інтуїтивно зрозумілий інтерфейс;
- функція автоматичного оновлення програми.

Таким чином, головним завданням є створення безкоштовного, зручного у використанні бенчмарку для швидкого тестування продуктивності ПК під управлінням ОС Windows.

1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

Бенчмарк - це спеціалізований інструмент, призначений для оцінювання продуктивності електронних пристроїв, зокрема персональних комп’ютерів. За допомогою програмного забезпечення для порівняльного аналізу виконуються серії тестів, які дозволяють визначити ефективність роботи ПК [1].

Результати, отримані під час таких тестувань, виражаються у вигляді оцінки (балу), яка слугує орієнтиром для порівняння між різними комп’ютерними системами. Чим вищий бал - тим вища продуктивність пристрою. Це значно спрощує аналіз систем, адже порівнювати числові показники тестів значно легше, ніж аналізувати складні технічні параметри. Бенчмарки також дозволяють оцінити швидкодію окремих компонентів (процесора, графічної карти, пам’яті тощо) та загальну ефективність роботи всієї системи.

Окремі бенчмарк-програми включають функції стрес-тестування, що дозволяють виявити явище тротлінгу - зниження продуктивності при перегріванні. Крім результатів тестування, більшість бенчмарків надають користувачу докладну інформацію про апаратні компоненти, які піддаються аналізу.

Окрім тестування «заліза», бенчмарки можуть застосовуватись і для оцінки продуктивності програмного забезпечення - зокрема, відеоігор або професійних програм для редагування фото та відео. Це дозволяє визначити, як саме програма працює на конкретному обладнанні, а також порівняти ефективність її роботи на різних системах. Деякі бенчмарки додатково тестують енергоефективність вимірюючи споживання ресурсів у різних режимах.

Таким чином, бенчмарки є потужним інструментом для комплексної оцінки продуктивності комп’ютерної системи. Вони допомагають користувачу прийняти обґрунтоване рішення щодо доцільності оновлення або заміни апаратних компонентів ПК.

1.1 Види та засоби тестування

Бенчмарки зазвичай класифікуються на мобільні (для смартфонів і планшетів) та настільні (призначені для персональних комп’ютерів), хоча також існують універсальні, або кросплатформні, варіанти тестувань. Основна мета таких інструментів - надати оцінку або числовий бал, який розраховується на основі співвідношення між складністю виконуваних завдань і часом, необхідним на їхнє завершення. Тестуванню підлягають як вся система, так і окремі її елементи, включаючи компоненти, операції та алгоритми.

У складі персонального комп’ютера ключовими елементами для тестування зазвичай виступають центральний процесор (CPU) та графічна карта (GPU). Саме ці модулі перевіряються за допомогою складних алгоритмів, які потребують значних обчислювальних ресурсів та часу. Окреме тестування продуктивності CPU та GPU дозволяє отримати точнішу оцінку їхньої ефективності в складі всієї системи.

Окрему категорію становлять стрес-тести - види тестування, які створюють максимальне навантаження на комп’ютер шляхом паралельного запуску численних складних операцій. Такі тести дозволяють не лише перевірити межі продуктивності, а й виявити явище тротлінгу - автоматичне зниження частоти роботи компонентів під час перегріву, що безпосередньо впливає на загальну швидкодію системи. У процесі стрес-тестування фіксуються температурні показники, що дозволяє виявити ефективність або недоліки системи охолодження. Якщо система охолодження не справляється, температура компонентів зростає, що провокує тротлінг - зниження продуктивності. Бенчмарки допомагають виявляти ці проблеми, що є корисним як для користувачів, так і для розробників та виробників комп’ютерного обладнання [2].

Окрім CPU та GPU, бенчмарки здатні тестувати й інші складові комп’ютера - оперативну пам’ять (RAM), накопичувачі (HDD/SSD), а також мережеві інтерфейси. Аналіз оперативної пам’яті дозволяє виявити її реальну

продуктивність, а перевірка накопичувачів - швидкість зчитування та запису даних. Мережеві тести дають змогу оцінити якість інтернет-з’єднання або локальної мережі.

Бенчмарки також використовуються для порівняння різних комп’ютерних систем або оцінювання ефективності оновлень. Проведення тестів до і після оновлення комплектуючих дозволяє чітко побачити приріст продуктивності та зробити обґрунтовані висновки щодо доцільності модернізації.

Водночас варто пам’ятати, що результати бенчмарків є орієнтовними і можуть варіюватися в залежності від умов проведення тестів та специфіки використання системи. На точність результатів також впливають налаштування тестового сценарію, тип обраного бенчмарку та коректність його виконання. Саме тому важливо ретельно підходити до аналізу отриманих даних і враховувати всі супутні фактори.

1.2 Стан та актуальність на ринку

На сьогодні бенчмарки широко застосовуються у різних галузях, особливо там, де продуктивність комп’ютера безпосередньо впливає на якість користувацького досвіду. Розглянемо деякі з основних напрямків.

Індустрія ігор, обробки відео та зображень. Геймери, відеоредактори та спеціалісти з обробки графіки активно використовують результати бенчмарків для оцінки можливостей своїх систем. Висока швидкодія комп’ютера забезпечує комфортну гру або швидке виконання завдань редагування. Особливу увагу вони приділяють графічним процесорам (GPU), адже саме ці компоненти відіграють ключову роль у відображенні візуальних ефектів, рендерингу, а також при роботі з відео та зображеннями [3].

Галузь інформаційних технологій. Фахівці IT-сектору у своїй роботі значною мірою залежать від стабільної та високої продуктивності комп’ютерного

обладнання. Ефективність розробки, тестування програмного забезпечення, адміністрування систем або виконання обчислювальних задач безпосередньо пов’язана з потужністю використовуваного ПК.

Сфера майнінгу криптовалют. У криптомайнінгу вирішальну роль відіграє продуктивність відеокарт, які використовуються для обчислення хеш-функцій. Бенчмарки допомагають майнерам обирати найефективніші апаратні рішення, орієнтуючись на продуктивність та енергоефективність, що дозволяє досягати максимального прибутку.

Крім того, бенчмарки є важливим інструментом у процесі розробки програмного забезпечення. Розробники застосовують їх для аналізу ефективності своїх програм на різних апаратних конфігураціях, а також для виявлення вузьких місць і оптимізації коду. За допомогою бенчмарків можна порівнювати продуктивність різних технологій та архітектур, що особливо актуально у сфері комп’ютерної інженерії та електроніки.

Отже, бенчмарки слугують універсальним засобом оцінки швидкодії комп’ютерних систем у різних сферах діяльності. Вони дозволяють користувачам орієнтуватися в поточному стані технічного ринку, порівнювати різні апаратні конфігурації та приймати обґрунтовані рішення щодо оновлення або вибору нового обладнання, орієнтуючись на реальні показники продуктивності.

2 РОЗРОБКА ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Torobench - це програмне забезпечення, розроблене для тестування продуктивності персональних комп’ютерів під керуванням операційної системи Windows. Однією з ключових особливостей Torobench є наявність онлайн-рейтингу, що дозволяє користувачам порівнювати свої результати з іншими. Основною метою проєкту є створення швидкого, безкоштовного та зручного інструменту для оцінки апаратної продуктивності ПК [4].

У реалізації Torobench використані такі основні технології та бібліотеки:

– Windows Forms і .NET - додаток створено на базі C# із використанням Windows Forms, що забезпечує сумісність і стабільну роботу в середовищі Windows;

– Dropbox API - для збереження та синхронізації даних рейтингу використовується хмарне сховище Dropbox, до якого програма підключається через відповідний API;

– Guna Framework - застосовується для створення сучасного та привабливого інтерфейсу, враховуючи обмеження стандартного UI в Windows Forms;

– AutoUpdater.NET - бібліотека, яка відповідає за автоматичне оновлення програми, забезпечуючи користувачеві завжди актуальну версію;

– OpenCL - обчислювальні алгоритми для графічного процесора реалізовані на OpenCL, що дозволяє підтримувати різні бренди GPU без прив’язки до конкретного виробника.

Torobench дозволяє оцінити продуктивність основних апаратних компонентів комп’ютера. Для цього використовуються такі обчислювальні задачі, як:

– пошук простих чисел;

– виконання арифметичних і тригонометричних операцій;

– піднесення чисел до степеня;

– вимірювання швидкості хешування;

– паралельні обчислення, включаючи множення великих матриць;

– обробка чисел з великою точністю та робота з рядками (наприклад, конкатенація).

Також Torobench підтримує стрес-тестування системи з можливістю моніторингу температури, навантаження на компоненти та виявлення тротлінгу у режимі реального часу. Після завершення тестів результати автоматично додаються до загального рейтингу, де користувач може переглядати найкращі показники, здійснювати пошук та сортування за різними параметрами.

Крім того, додаток підтримує автоматичне оновлення, включаючи функцію пропуску певних версій або отримання нагадувань про оновлення. Інтерфейс програми можна налаштувати під власні вподобання - доступні світла, темна або системна теми оформлення.

Torobench є багатопотоковим додатком, що дозволяє проводити тестування без заморожування чи блокування графічного інтерфейсу, забезпечуючи зручність у користуванні під час виконання ресурсоємних операцій.

2.1 Технології .NET Framework та Windows Forms

Розробка програми здійснювалася з використанням .NET Framework 4.6.1, що на момент створення застосунку був найновішою стабільною версією платформи. Для побудови інтерфейсу користувача було обрано Windows Forms, який забезпечує зручні інструменти для створення віконних додатків та налаштування елементів керування [5].

Однак для надання інтерфейсу сучасного вигляду в проєкті також був використаний Guna Framework - графічна бібліотека, яка значно розширює стандартні можливості Windows Forms. Завдяки Guna Framework вдалося створити стильний, привабливий та функціональний інтерфейс із підтримкою кастомізації. Бібліотека надає великий вибір графічних компонентів та інструментів стилізації, що дозволяє реалізовувати сучасні UI-рішення відповідно до актуальних дизайнерських вимог [6].

У кореневій директорії застосунку (рис. 2.1) було створено кілька класів-форм, кожна з яких відповідає за реалізацію окремих функціональних можливостей програми. Окрім основного проєкту застосунку, також було реалізовано додатковий проєкт, призначений для процесу встановлення бенчмарку.

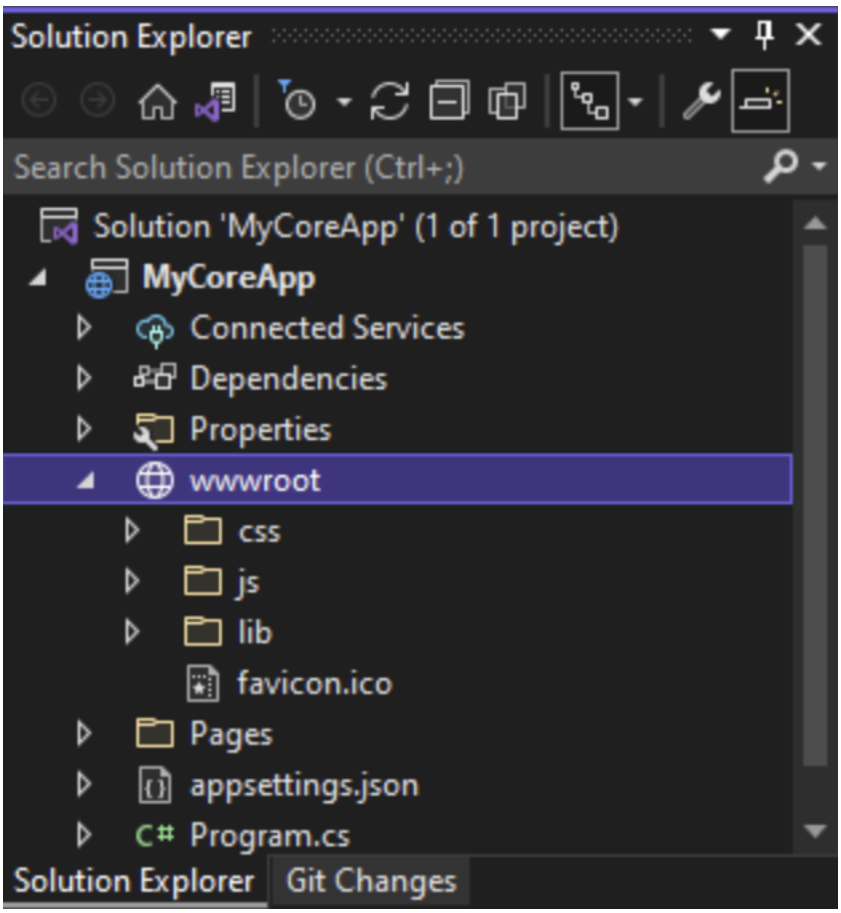


Рисунок 2.1 - Коренева папка програми

2.2 Тестування центрального процесора

Оцінювання продуктивності центрального процесора (CPU) персонального комп’ютера зазвичай поділяється на два основні етапи: перший - це вимірювання ефективності одного логічного ядра (Single-Core), а другий - тестування загальної продуктивності всіх логічних ядер (Multi-Core). Такий поділ зумовлений тим, що не все програмне забезпечення підтримує багатопотоковість, адже деякі задачі не підлягають розпаралеленню. Під час тестування процесор виконує низку обчислювальних операцій, а час, витрачений на їх обробку, слугує основою для розрахунку підсумкового балу продуктивності. Цей бал визначається як відношення часу виконання до визначеної константи. Чим менше часу потрібно на виконання задачі, тим вищу оцінку отримує система.

У реалізованому коді, який є частиною функціоналу програми, здійснюється тестування швидкодії CPU шляхом виконання серії обчислень. Після підготовки вхідних списків із даними, застосовуються два методи розрахунків: послідовна обробка на одному ядрі процесора та паралельна обробка на декількох ядрах із використанням бібліотеки Parallel [7].

У рамках цього коду виконуються наступні дії:

– встановлюється афінітна маска процесора, що обмежує виконання задачі одним ядром для забезпечення чистоти експерименту;

– обробляються списки чисел floatList і intList, кожен з яких містить по 2000 елементів. Над цими елементами виконуються арифметичні операції та застосовуються математичні функції;

– над списком stringList здійснюється операція конкатенації рядків та заміна символів;

– для точного заміру часу обчислень використовується клас Stopwatch;

– зібрані дані про швидкодію різних методів записуються до об'єкта rS;

– підсумкові результати виводяться у вигляді кількості операцій, виконаних на мільйон мілісекунд.

Головна мета цього фрагменту коду - проаналізувати вплив різних способів обчислень на продуктивність процесора та провести їх порівняння. Застосування бібліотеки Parallel дозволяє реалізувати багатопотокову обробку, а використання Stopwatch забезпечує точне визначення часу виконання, що в сукупності дозволяє провести якісний аналіз ефективності CPU.

2.2.1 Тестування у режимі одного ядра

Для здійснення тестування продуктивності процесора в однопотоковому режимі, програма запускається з налаштуванням, яке обмежує її виконання лише одним із доступних ядер центрального процесора. Такий підхід дозволяє ізолювати виконання обчислень на конкретному ядрі та точно виміряти його швидкодію без впливу багатоядерної обробки.

Для налаштування виконання коду на конкретному логічному ядрі центрального процесора використовується клас

`System.Diagnostics.Process`, який дозволяє встановити так звану «маску афінітету» - ідентифікатор ядра, на якому працюватиме програма (рис. 2.2).

```
long AffinityMask = (long)Proc.ProcessorAffinity;
AffinityMask &= 0x0001;
Proc.ProcessorAffinity = (IntPtr)AffinityMask;
```

Рисунок 2.2 - Встановлення процесу в однопотоковий режим

Згідно з даними з відкритих джерел, до операцій, що інтенсивно навантажують процесор, належать піднесення до степеня великих чисел, обчислення тригонометричних функцій, а також маніпуляції з рядками, зокрема операція конкатенації. Такі дії можуть повністю завантажити ресурси CPU, досягаючи 100% його використання.

Оцінювання завантаження процесора може здійснюватися різними методами, зокрема шляхом вимірювання часу виконання операцій у межах фіксованого проміжку часу, наприклад, однієї секунди. У програмі для моніторингу навантаження на CPU застосовується бібліотека OpenHardwareMonitor, яка дозволяє відстежувати різні сенсорні параметри системи. Аналогічні функції виконує популярний серед розробників інструмент CPUID HWMonitor, який часто застосовується під час створення бенчмарків.

Під час розробки було також вирішено додати тестування низки додаткових ресурсомістких операцій. До них належать: обчислення з числами, що мають велику кількість знаків після десяткової крапки, пошук простих чисел, а також сортування значних обсягів даних у великих списках.

2.3 Тестування графічного процесора

У середовищі .NET Framework не передбачено вбудованих засобів для оцінювання продуктивності графічного процесора, тому в рамках проєкту було

застосовано сторонні інструменти [8]:

– OpenCL - низькорівнева мова програмування, що забезпечує прямий доступ до графічного ядра відеокарти та дозволяє виконувати різноманітні алгоритми та обчислення безпосередньо на GPU;

– CUDAfy.NET - зовнішня бібліотека, що надає можливість використання потужностей GPU для виконання обчислювальних задач, дозволяючи розробникам працювати мовою C#.

Обидва інструменти є кросплатформними, завдяки чому тестування продуктивності може здійснюватися незалежно від виробника відеокарти.

Методи, використані для тестування GPU, включають всі обчислювальні алгоритми, застосовані раніше для ЦП, а також додаткові паралельні операції, зокрема множення та ділення матриць. Аналіз показав, що саме ці операції викликають максимальне навантаження графічного процесора (до 100%), що є ключовим для ефективного тестування.

Для реалізації обчислень було використано бібліотеку OpenCL, яка містить кілька спеціалізованих класів:

– OpenCL - базовий клас для взаємодії з GPU;

– MultiCL - для тестування з використанням усіх доступних апаратних ресурсів;

– EasyCL - спрощений варіант для реалізації базових сценаріїв GPU-програмування.

Методика оцінювання продуктивності аналогічна до тієї, що використовується при тестуванні центрального процесора, однак для відеокарт використовується інша константа, яка враховує вищу швидкість виконання обчислень на GPU.

У фрагменті коду, що реалізує GPU-тестування, визначено обчислювальне ядро (kernel), яке оперує з двома масивами (дійсні та цілі числа) та їхніми розмірами. Усередині ядра виконується цикл з арифметичними операціями над елементами масивів.

Після цього налаштовуються параметри OpenCL, передаються відповідні

дані та створюється окремий потік, у якому запускається обчислювальний цикл через метод `Execute` класу `OpenCL`. Цей цикл виконується задану кількість ітерацій, а стан виконання постійно оновлюється за допомогою `ProgressChangedEventHandler`.

У разі зовнішнього переривання процесу, виконання коду зупиняється. Після завершення обчислень проводиться підрахунок продуктивності графічного процесора, а отриманий результат записується в змінну `rS.GpuScore`.

2.4 Засоби для тестування системи

Для реалізації функціоналу стрес-тестування, що охоплює одночасну перевірку продуктивності CPU і GPU, а також моніторинг навантаження системи в реальному часі, доцільно використовувати асинхронні методи. Це дає змогу розподілити завдання між окремими потоками, забезпечуючи ефективне використання ресурсів [9].

У прикладі коду представлено метод `WhileCPU`, який є асинхронною процедурою, призначеною для виконання інтенсивних обчислень на CPU у фоновому потоці. Усередині методу реалізовано нескінченний цикл `while`, що містить паралельний код з використанням вкладених конструкцій `Parallel.ForEach` і `Parallel.For`.

Перший цикл `Parallel.ForEach` обробляє елементи списку `L` у паралельному режимі. Хоча конкретні дії з елементами списку не вказані, можна припустити, що йдеться про додаткові обчислення, які виконуються в кожному потоці.

Другий цикл `Parallel.For` також працює паралельно, виконуючи мільйон ітерацій для кожного елемента списку `L`. І хоча в коді не уточнено, які саме операції там реалізовані, ймовірно, це обчислення, що інтенсивно навантажують CPU.

У коді передбачено змінну `stop`, яка може бути використана для зупинки виконання циклу, що дозволяє керувати тривалістю стрес-тесту. Після завершення метод повертає значення `0`, що свідчить про коректне завершення роботи.

Такий підхід забезпечує максимальне навантаження на основні компоненти системи та дозволяє виявити троттлінг і перегрів шляхом спостереження за поточними показниками навантаження в режимі реального часу.

2.5 Тестування навантаження швидкості Інтернету

У цьому розділі розглядається процес перевірки швидкості Інтернет-з'єднання, мета якого - визначення реальної пропускної здатності мережі на комп’ютері та надання користувачеві результатів після завершення тестування. Для цього застосовується спеціалізований інструмент, який виконує обмін даними в обох напрямках і повертає результати, що містять інформацію про швидкість завантаження (download) та відвантаження (upload). Крім цього, визначаються й інші ключові параметри, зокрема ping та jitter, які суттєво впливають на якість з'єднання.

Після завершення тестування результати автоматично відображаються користувачеві. Крім того, в процесі перевірки він може спостерігати за перебігом завантаження та відвантаження даних у реальному часі, що підвищує прозорість і зручність взаємодії з додатком [10].

Проведення тесту швидкості Інтернету дозволяє оцінити ефективність наявного мережевого середовища та, у разі потреби, прийняти рішення щодо модернізації мережі. Швидкість завантаження - це параметр, що вказує на те, з якою швидкістю дані з Інтернету потрапляють на пристрій користувача. Цей показник є одним із ключових у повсякденному користуванні мережею, оскільки

впливає на зручність перегляду сторінок, відео, завантаження файлів тощо. Висока швидкість завантаження зазвичай забезпечує комфортне користування Інтернетом. Вона вимірюється у мегабітах за секунду (Мбіт/с) та відображає обсяг даних, що може бути передано з сервера до пристрою користувача за одиницю часу.

На якість завантаження можуть впливати різноманітні чинники, серед яких: швидкість інтернет-підключення, стабільність з'єднання, наявність шкідливого програмного забезпечення, технічні особливості мережі, а також завантаження каналів зв'язку в конкретному регіоні.

Швидкість відвантаження, своєю чергою, демонструє, з якою швидкістю дані передаються з комп’ютера користувача до мережі Інтернет. Вона є протилежністю швидкості завантаження і не менш важливою - особливо для завдань, пов’язаних із передаванням даних: завантаженням файлів у хмару, надсиланням електронних листів із вкладеннями, участю в відеоконференціях або стрімінгових трансляціях. Швидкість відвантаження також вимірюється в Мбіт/с і може бути обмежена як самим провайдером, так і технічними особливостями обладнання чи рівнем навантаження на мережу.

Ще одним важливим параметром є *ping* - це мережевий інструмент, що вимірює затримку під час передавання пакету даних від користувача до сервера і назад. Він використовується для перевірки доступності сервера або веб-ресурсу та загальної якості мережевого з'єднання. Вимірюється у мілісекундах (мс), і чим менше це значення, тим швидше здійснюється зв'язок. Високий пінг може вказувати на затримки, спричинені перевантаженням або низькою пропускною здатністю.

Окремо слід згадати `jitter` - параметр, що визначає змінність у часі доставки пакетів. Іншими словами, він вимірює нестабільність у передачі даних по мережі. Цей показник є критично важливим для забезпечення якісного потокового відеозв'язку або онлайн-ігор, де навіть незначні затримки можуть негативно впливати на досвід користувача. Jitter зазвичай вимірюється у мілісекундах або у відсотках від пінгу: що менше значення, то стабільніше та

якісніше з'єднання.

Для реалізації функціоналу тестування у програмі було створено окрему форму, яка забезпечує вимірювання згаданих параметрів. У цій формі відображається інформація про тип мережевого інтерфейсу (Wi-Fi, Ethernet тощо), а також графічні та текстові компоненти, що показують швидкість завантаження, відвантаження, ping і jitter. Такий підхід забезпечує наочність і зручність у використанні функції оцінки якості інтернет-з'єднання

Швидкість завантаження перевіряється шляхом спроби програми отримати файл обсягом 100 мегабайт із віддаленого сервера в мережі Інтернет. Такий розмір було обрано як оптимальний для універсального тестування, адже в користувачів швидкість підключення може суттєво відрізнятись. Вибір саме такого обсягу, на думку автора, дозволяє забезпечити комфортну тривалість тесту для більшості користувачів. Тестування реалізоване таким чином, що файл завантажується протягом фіксованого періоду часу. Проте, якщо швидкість інтернету користувача перевищує очікувану, процес завершиться раніше, дозволяючи отримати результати швидше. Натомість для користувачів із повільним інтернетом тривалість тесту залишиться сталою.

Для ідентифікації мережевого пристрою, що забезпечує з'єднання з Інтернетом, у програмі реалізовано окрему функцію, яка використовує класи `NetworkInterface` і `GatewayIPAddressInformation`. Ця функція дозволяє визначити назву та модель саме того інтерфейсу, який наразі активний і має доступ до Інтернету. Програма здійснює обхід усіх доступних адаптерів у системі, надсилаючи запити до віддаленого сервера, щоб перевірити, який із них забезпечує з'єднання. Назва пристрою зберігається для подальшого використання.

У реалізації використовується .NET-клас `NetworkInterface`, що надає доступ до всіх мережевих інтерфейсів пристрою. Метод `GetAllNetworkInterfaces` повертає масив усіх доступних мережевих адаптерів. У циклі `foreach` перевіряється кожен адаптер на наявність активного статусу (`OperationalStatus.Up`) та відповідність типу мережі (виключаючи, наприклад, `Loopback`). Якщо ці умови виконуються, за допомогою методу

`GetIPProperties()` перевіряється наявність шлюзових адрес через властивість `GatewayAddresses`. Якщо шлюз є, і він належить до IPv4-адресної сім'ї, назва пристрою зберігається в змінну `deviceName`, і цикл припиняється оператором `break`. Метод повертає збережену назву пристрою.

Для виконання самого тесту швидкості завантаження створено асинхронну функцію, яка не блокує інтерфейс програми, що дозволяє виконувати тест у фоновому режимі без втручання у взаємодію користувача з додатком. У функції ініціалізується таймер для контролю часу тестування, забезпечуючи однакову тривалість тесту як для користувачів із повільним, так і з швидким з'єднанням. Такий підхід дозволяє зберегти зручність користування.

Окрім обчислення кінцевого результату після завершення тесту, користувач у режимі реального часу бачить поточну швидкість завантаження та відображення прогресу через подію `DownloadProgressChanged`. Це забезпечує прозорість процесу та дозволяє оцінити стабільність інтернет-з'єднання під час тестування. В разі високої або низької швидкості, одиниці вимірювання автоматично змінюються - наприклад, з мегабітів на кілобіти або гігабіти, залежно від досягнутого значення. Також в інтерфейсі передбачено візуальні елементи, які демонструють хід тесту.

Після завершення перевірки швидкості завантаження програма автоматично переходить до тестування швидкості відвантаження даних.

Технічно реалізація перевірки швидкості завантаження виконана у вигляді методу на мові C#, який використовує клас `WebClient` для завантаження файлу з Інтернету. Метод повертає значення типу `double`, що відповідає швидкості в мегабітах за секунду (Mbps). Для вимірювання часу використовується таймер, а для відображення прогресу - прогрес-бар. Якщо завантаження триває понад 10 секунд, воно зупиняється, і користувач отримує результат, що був досягнутий на цей момент.

Код також містить логіку автоматичної зміни одиниць вимірювання залежно від значення: при швидкості менше 1 Мбіт/с вона відображається у кілобітах на секунду (Kbps), при значеннях від 1 до 1000 Мбіт/с - у мегабітах

(Mbps), а понад 1 Гбіт/с - у гігабітах (Gbps).

Крім того, якщо під час тесту швидкість дуже низька або з'єднання взагалі відсутнє, метод присвоює змінній `downloadSpeed` значення `-1`, що сигналізує про проблему з мережею. Це дозволяє виявити такі збої, як відсутність підключення або критично низьку швидкість передачі даних.

Загалом, ця програмна реалізація є ефективним інструментом для вимірювання швидкості завантаження з Інтернету. Вона дозволяє виявляти мережеві проблеми та аналізувати якість підключення. Проте варто враховувати, що результати можуть змінюватися залежно від таких чинників, як фізична відстань до сервера, тип мережі та поточне навантаження на неї.

Для перевірки швидкості відвантаження даних використовується аналогічний підхід до того, який застосовується для тестування швидкості завантаження. Основна відмінність полягає в тому, що файл для відвантаження створюється локально на комп’ютері користувача та має фіксований розмір, однаковий для всіх користувачів, що забезпечує уніфікованість умов тестування.

Для здійснення заміру швидкості використовується клієнт `WebClient`, який передає на віддалений сервер випадково згенеровані дані обсягом 10 мегабайт. Програмна логіка реалізована у вигляді двох методів:

- `GetUploadSpeed()` - відповідального за обчислення швидкості передавання даних;

- `UploadDataAsync()` - виконує безпосереднє відвантаження інформації на сервер і слідкує за його прогресом.

У методі `GetUploadSpeed()` спочатку створюється новий екземпляр `WebClient`, після чого генерується буфер із випадковими даними розміром 10 МБ. Далі ініціалізується таймер із лімітом у 10 секунд. Також налаштовуються параметри для візуального відображення процесу на інтерфейсі користувача - зокрема, мінімальні та максимальні значення анімації, які показують динаміку передавання.

Після цього викликається метод `UploadDataAsync()`, який відповідає за сам процес відвантаження. У ході його виконання здійснюється підрахунок

кількості переданих байтів, і відповідна інформація виводиться на графічний інтерфейс. Також в реальному часі визначається швидкість передачі даних, яка паралельно відображається користувачу для більшої наочності процесу.

Після завершення відвантаження `GetUploadSpeed()` обчислює підсумкову швидкість відправлення даних, ґрунтуючись на обсязі переданих даних і витраченому часу, та повертає це значення для подальшого відображення або аналізу.

Програмна реалізація тестування затримки (ping) представлена методом `GetPingTime()`, який дозволяє виміряти час, необхідний для передачі інформаційного пакету до сервера та отримання відповіді від нього з використанням протоколу `Ping`.

У тілі методу створюється екземпляр класу `Ping`, який використовується для відправлення запитів на задану адресу сервера та обробки відповіді. Об'єкт `Ping` поміщений у блок `using`, що гарантує його автоматичне закриття після виконання, запобігаючи витокам пам’яті.

У методі налаштовуються параметри Ping-запиту, зокрема `DontFragment` - для запобігання фрагментації пакетів, і `Ttl` - для обмеження максимальної кількості вузлів маршруту, через які може пройти пакет. Запит надсилається за допомогою методу `SendPingAsync()`, куди передається адреса сервера (`www.google.com`), тайм-аут у 5000 мілісекунд, буфер даних і відповідні параметри.

Після надсилання запиту очікується відповідь, яка зберігається в об'єкті `PingReply`. Якщо відповідь успішна (статус `IPStatus.Success`), то значення часу проходження пакету в обидві сторони (`reply.RoundtripTime`) записується у змінну `pingTime`. У разі помилки або відсутності відповіді `pingTime` залишається рівним -1.

Щодо тестування джитеру (jitter), воно реалізується у вигляді окремої функції, яка розраховує середнє значення затримки (ping) і розкид (джитер) на основі п’яти послідовних пінгів до сайту `www.google.com`. Для цього також

використовується клас `Ping` із бібліотеки .NET.

Функція є асинхронною, реалізованою з використанням ключових слів `async` та `await`, що дозволяє надсилати запити та обробляти відповіді без блокування потоку виконання. Параметр `DontFragment` встановлюється для уникнення розбиття пакетів при передачі, що зменшує ризик втрати даних.

Середнє значення затримки обчислюється за допомогою методу `Average()` з урахуванням лише тих значень, які перевищують 0. Для визначення джитеру виконується обчислення середньої абсолютної різниці між кожним пінгом і загальним середнім значенням, після чого результат ділиться на (кількість пінгів - 1).

На виході функція повертає значення джитеру у мілісекундах. Оскільки вона є асинхронною, її можна викликати з інших асинхронних методів або з синхронного коду з використанням `await`.

2.6 Тестування диску

Тестування швидкості читання та запису диска є ключовим етапом оцінки загальної продуктивності комп’ютера та його апаратних компонентів. Воно дозволяє визначити, з якою швидкістю пристрій обробляє операції з даними на диску.

Процес тестування зазвичай передбачає послідовне зчитування та запис інформації. При тестуванні читання програма фіксує швидкість, з якою дані зчитуються з накопичувача, а при тестуванні запису - швидкість, з якою вони записуються на диск.

Основним параметром, що характеризує продуктивність накопичувача, є швидкість передачі даних - обсяг інформації, який може бути передано за певний проміжок часу.

Результати такого тестування мають важливе значення для ряду завдань:

відтворення відео, редагування зображень, графічного дизайну, програмування та інших ресурсомістких процесів. Важливо враховувати, що реальна швидкість диска може змінюватися в залежності від типу даних, з якими ведеться робота, тому тести повинні максимально відповідати реальним умовам експлуатації.

Крім того, перевірка швидкості накопичувача може виявити несправності. Наприклад, суттєве зниження швидкості зчитування або запису може свідчити про зношення пристрою та необхідність його заміни [11].

Також подібне тестування дозволяє оцінити сумісність і узгодженість роботи різних компонентів комп’ютера. Якщо накопичувач демонструє високу швидкість, але інші компоненти не забезпечують належну передачу даних, це може бути ознакою потреби в їх оновленні.

У підсумку, тестування швидкості диска є невід’ємною частиною моніторингу продуктивності комп’ютера. Регулярна перевірка накопичувача та інших системних компонентів дозволяє підтримувати стабільну роботу пристрою та вчасно реагувати на можливі проблеми

Форма для тестування диска надає користувачеві інформацію про швидкість читання та запису, відображає модель накопичувача, що проходить тестування, а також демонструє поточний прогрес операцій читання та запису.

Код, який відповідає за визначення моделі диска, призначений для отримання назви пристрою з операційної системи. Для цього використовується клас `ManagementObjectSearcher` із бібліотеки `System.Management`, що дозволяє виконувати запити до WMI. У запиті отримується список усіх пристроїв зберігання даних, представлених через клас `Win32_DiskDrive`. Далі за допомогою циклу `foreach` здійснюється перебір знайдених об’єктів, з яких зчитується властивість `Model`. Отримане значення зберігається у змінній `deviceName`, після чого цикл переривається, оскільки потрібна лише інформація про перший пристрій. Результат роботи функції `GetDeviceName()` повертається у вигляді рядка, який можуть використовувати інші частини програми.

Реалізація тестування швидкості запису і читання диска входить до складу

програми, що перевіряє продуктивність накопичувача. Вона містить дві ключові функції: `Worker_DoWork` та `Process_OutputDataReceived`, які працюють паралельно.

Метод `Worker_DoWork` виконується в окремому потоці та ініціює запуск консольної утиліти `DiskTestConsoleApp.exe` через об'єкт `Process`, який управляє запуском зовнішніх процесів. Вивід утиліти обробляється функцією `Process_OutputDataReceived`, яка оновлює графічний інтерфейс відповідно до отриманих результатів.

`Process_OutputDataReceived` реагує на кожен новий рядок, виведений утилітою, аналізує значення швидкості читання і запису, і за потреби оновлює відповідні елементи інтерфейсу. Також функція керує прапорцем `IsReadNow`, що сигналізує про активну операцію читання.

Консольна частина програми - окремий проект на C#, що реалізує багатопотокове асинхронне тестування диска. Вона використовує буфер розміром 10 МБ та чергу глибиною 100 для створення тестових файлів. Загальний обсяг файлів дорівнює добутку розміру блоку на глибину черги. Запис і зчитування виконуються паралельно з використанням декількох потоків, що дозволяє симулювати високе навантаження.

По завершенні процесів записи та зчитування програма видаляє створені файли і зупиняє моніторинг навантаження на диск. Останній реалізовано через щосекундне зчитування показника “% Idle Time” з лічильника продуктивності “PhysicalDisk”.

Програма дозволяє налаштовувати параметри, зокрема розмір блоку та глибину черги, для аналізу їхнього впливу на швидкодію диска. Метод `GetDiskLoading()` запускається в окремому потоці й кожну секунду виводить на екран значення навантаження на диск. Цикл припиняється, коли змінна `stopMonitoring` приймає значення `true`.

Метод `WriteAsync()` відповідає за асинхронний запис даних у файли. Він створює буфер розміром `blockSizeInKiB`, заповнює його випадковими байтами, і запускає запис у кілька файлів паралельно, використовуючи

`FileStream`. Запис виконується `queueDepth` разів у кожному потоці. Прогрес операції виводиться на консоль кожні 10 ітерацій.

Метод `ReadAsync()` реалізує асинхронне зчитування даних з файлів, які були створені під час запису. Для кожного потоку відкривається окремий файл і зчитується `queueDepth` блоків розміром `blockSizeInKiB`. Також виводиться прогрес кожного потоку через кожні 10 ітерацій.

Після завершення зчитування всі тимчасові файли видаляються за допомогою `File.Delete()`.

Метод `GetDiskLoading()` реалізований як `Task<int>` і використовується для постійного моніторингу навантаження на диск під час роботи програми. У циклі кожну секунду зчитується значення простою диска і виводиться як зворотна метрика навантаження. Коли `stopMonitoring` дорівнює `true`, цикл завершується і повертається 0.

Метод `WriteAsync()` також містить точний контроль часу за допомогою `Stopwatch` і реалізує блокування потоків через `lock`, щоб уникнути одночасного доступу до одного ресурсу. Дані записуються унікальними потоками у власні файли. Кожен запис ведеться із виведенням проміжного відсотка завершення.

Метод `ReadAsync()` аналогічним чином використовує буфер і `FileStream` для читання файлів. Потоки працюють паралельно і виводять прогрес. Метод завершується зі значенням 0, що вказує на успішне завершення читання.

Ця реалізація дозволяє не лише тестувати швидкість доступу до даних, а й контролювати поточне навантаження на диск, що є важливим для глибокої оцінки пропускної здатності дискової підсистеми.

2.7 Порівняння показників системи

З метою порівняння результатів тестування було проведено на двох різних системах (табл. 2.1). Усі тести виконувалися з використанням однакової версії програми та в один і той самий період часу.

Таблиця 2.1 - Системи, використані при тестуванні

Назва системи	Назва процесору	Назва відеокарти	ОЗП	ОС
Тест 1	AMD A6-3400M	AMD Radeon HD 6320M	8GB	Windows 10
Тест 2	AMD Ryzen 7 4800H	NVIDIA RTX 3060	8GB	Windows 11

Системи суттєво різняться за рівнем продуктивності та виробниками компонентів, що дозволяє оцінити роботу програми на широкому спектрі користувацьких конфігурацій.

Процесор Тест 2 демонструє значно вищі результати за кількістю набраних балів (табл. 2.2). Ще помітнішою є різниця в продуктивності відеокарт обох систем (табл. 2.3).

Таблиця 2.2 - Порівняння результатів тестування CPU

Назва системи	Одне ядро	Мультиядерність
Тест 1	783	1523
Тест 2	2875	17235

Таблиця 2.3 - Порівняння результатів тестування GPU

Найменування системи	Відсоткове відношення результатів,%
Тест 1	1.85
Тест 2	100

Крім того, ефективність функції стрес-тестування для кожної з систем підтверджується відповідними результатами (рис.2.3 , 2.4).

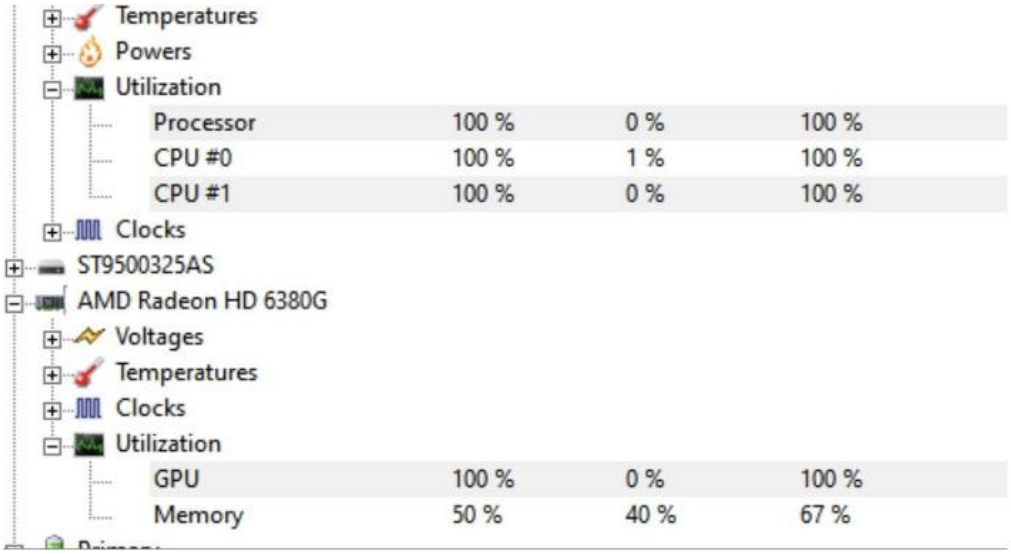


Рисунок 2.3 - Показники навантаження Системи 1 під час стрес-тестування

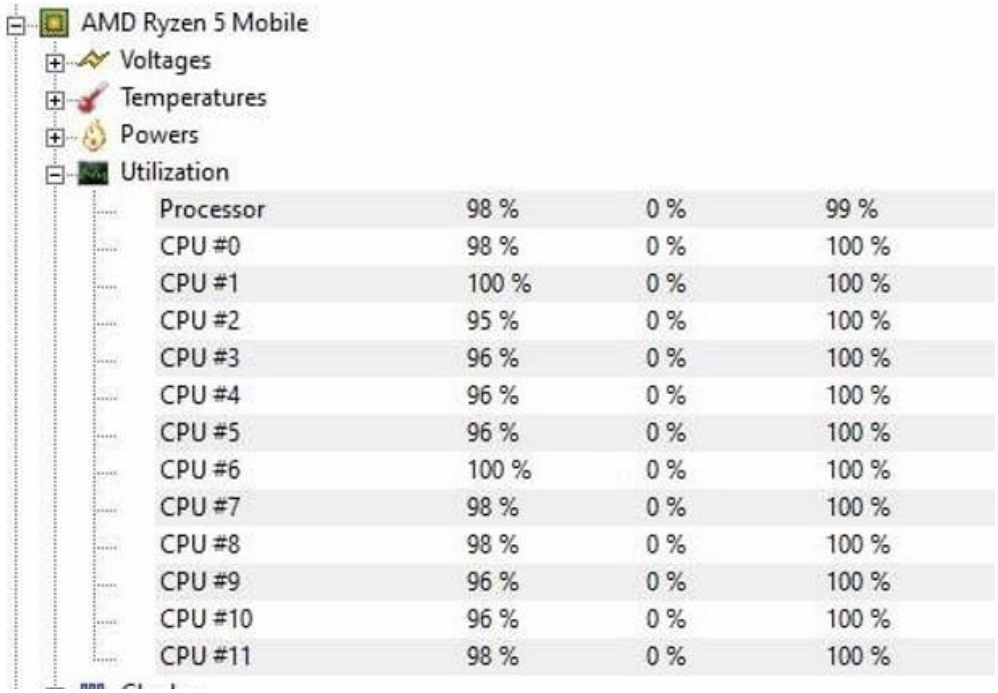


Рисунок 2.4 - Показники навантаження Системи 2 під час стрес-тестування

2.8 Показники потужності системи

Для реалізації рейтингу за продуктивністю доцільно використати вбудовану в застосунок таблицю, що містить дані про тестовані системи та результати їх тестувань. Передбачена також можливість сортування та пошуку в цій таблиці. Для реалізації таких функцій можна застосовувати бібліотеку System.LINQ, а відображення даних здійснювати за допомогою компонента DataGridView [12].

Таблиця рейтингу включає такі поля:

- позиція системи в рейтингу;
- назва процесора (CPU);
- назва відеокарти (GPU);
- обсяг оперативної пам’яті (RAM);
- найменування операційної системи (OS);
- кількість ядер процесора;
- тактова частота процесора;
- результати тестування в режимі Single-Core;
- результати тестування в режимі Multi-Core;
- результати тестування GPU.

Завдяки використанню System.LINQ можна виконувати сортування та пошук за заданими критеріями, що спрощує навігацію по таблиці. Компонент DataGridView забезпечує зручне представлення цієї інформації у форматі таблиці.

Використання технології Dropbox API дозволяє розробникам з відповідним обліковим записом керувати та зберігати дані своїх застосунків у хмарному сховищі Dropbox (рис. 2.5). Саме ця технологія застосовується для реалізації системи рейтингу продуктивності. Після завершення тестування процесора та відеокарти, результати автоматично зберігаються у хмарі у вигляді JSON-файлу.

Оскільки застосунок потребує зберігання лише однієї таблиці, а метою є створення безкоштовного рішення без використання платних хостингів,

доцільним є використання легко серіалізованого JSON-файлу замість повноцінної бази даних. Додатковою перевагою є невеликий обсяг даних: навіть за наявності тисяч записів розмір файлу не перевищуватиме кількох мегабайтів. У результаті формується система рейтингу, де всі результати зберігаються у віддаленому хмарному сховищі.

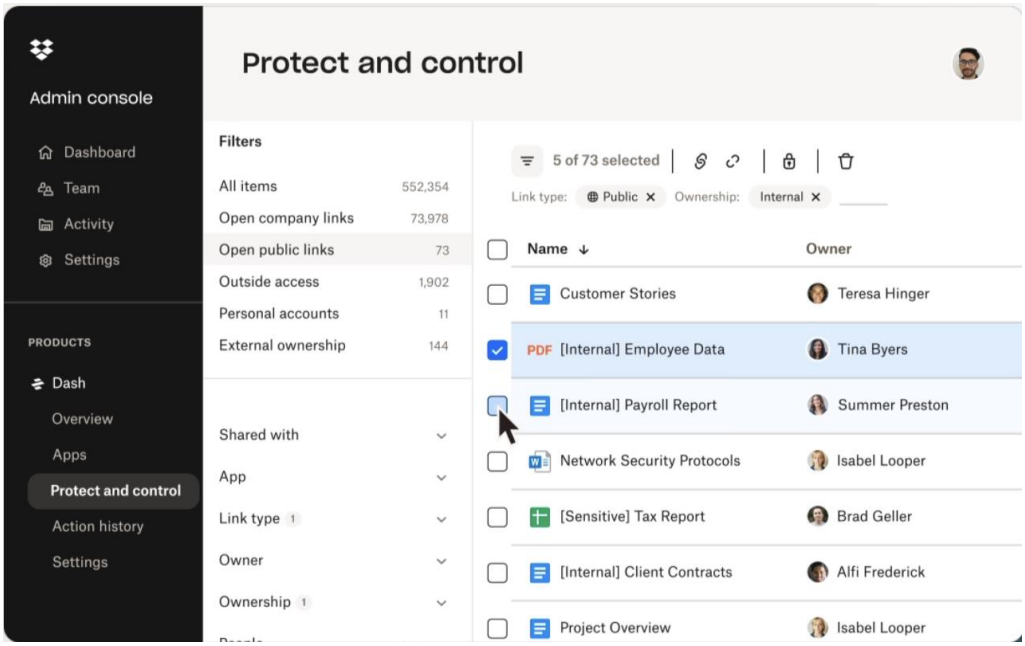


Рисунок 2.5 - Інтерфейс розробника Dropbox

2.9 Особливості програмного забезпечення системи

Механізм автооновлення застосунку реалізовано у двох варіантах.

Перший - оновлення через GitHub, що передбачає ручне завантаження оновлень шляхом клонування репозиторію. Такий підхід підходить для застосунків із відкритим вихідним кодом і орієнтований переважно на досвідчених користувачів.

Другий варіант автооновлення реалізовано з використанням сторонньої бібліотеки AutoUpdater.NET. У цьому випадку оновлення виконується

натисканням однієї кнопки, після чого програма автоматично завантажує та встановлює останню версію за кілька секунд. Для цього разом із AutoUpdater.NET використовується хмарне сховище Dropbox, у якому зберігається архів з оновленням та конфігураційний файл, що визначає параметри оновлення. За допомогою цього файлу можна задавати дату встановлення оновлення, пропускати певні версії та керувати загальною логікою оновлень.

Під час запуску бенчмарку здійснюється перевірка хмарного сховища, зчитується конфігураційний файл (лістинг 2.1, Додаток А), і користувачу надсилається повідомлення про доступність нового оновлення. У разі його відсутності, система повідомляє про актуальність встановленої версії.

Лістинг 2.1 - Файл-конфігуратор автооновлень

```
using AutoUpdaterDotNET;
using Dropbox.Api;
using Dropbox.Api.Files;
using System;
using System.IO;
using System.Threading.Tasks;

namespace YourApplicationNamespace
{
    public class UpdateConfigurator
    {
```

2.10 Візуальний вигляд програми

Окрім основного функціоналу, бенчмарк-програма надає додаткові можливості для налаштування зовнішнього вигляду. Користувач може обрати одну з трьох тем: світлу, темну або системну. Системна тема автоматично адаптується до налаштувань операційної системи - програма звертається до реєстру Windows та зчитує відповідне значення, визначаючи активну тему пристрою. Це особливо зручно для користувачів, які працюють у різний час доби.

Для реалізації цієї функції використано бібліотеку Guna Framework, що забезпечує зміну кольорової гами віджетів відповідно до вибраної теми.. Світла тема базується на м’яких світлих кольорах, що сприяють легкому сприйняттю інформації, тоді як темна - на темних тонах, зручних для роботи в умовах низького освітлення.

Крім системної, користувач має змогу вручну обрати будь-яку з двох інших тем. Усі зміни інтерфейсу зберігаються та автоматично застосовуються під час наступного запуску, що дозволяє персоналізувати програму під власні вподобання. Можливість налаштування вигляду інтерфейсу підвищує комфорт роботи, покращує користувацький досвід та сприяє ефективнішому використанню програми.

2.11 Встановлення програмного забезпечення

Для реалізації інсталятора було використано вбудований Setup-проєкт із .NET Framework (рис. 2.6). У проєкті містяться усі необхідні файли застосунку, а сам інсталятор налаштовано на встановлення однієї версії програми [13]. Процес встановлення включає такі етапи:

– вибір директорії для встановлення;

– вибір облікового запису користувача.

Після завершення інсталяції створюються ярлики на робочому столі та в меню «Пуск» для швидкого доступу до програми.

Щоб у результаті побудови інсталятора отримати файл формату .msi, необхідно правильно налаштувати файлову систему проєкту, зокрема папки “Application Folder”, “User’s Desktop” і “User’s Programs Menu”. Папка “Application Folder” визначає структуру каталогу застосунку після його встановлення.

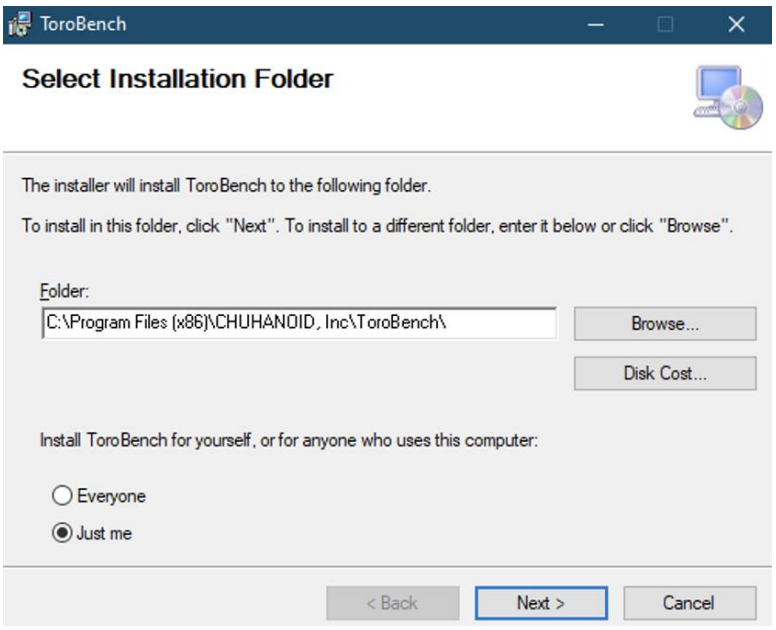


Рисунок 2.6 - Вікно встановлення програми

Усі файли, розміщені в цій папці, будуть перенесені до основної директорії програми у незмінному вигляді (рис.2.7). Тут зберігаються основні файли проєкту, додаткові бібліотеки та фреймворки, а також головний виконуваний файл ToroBench.exe.

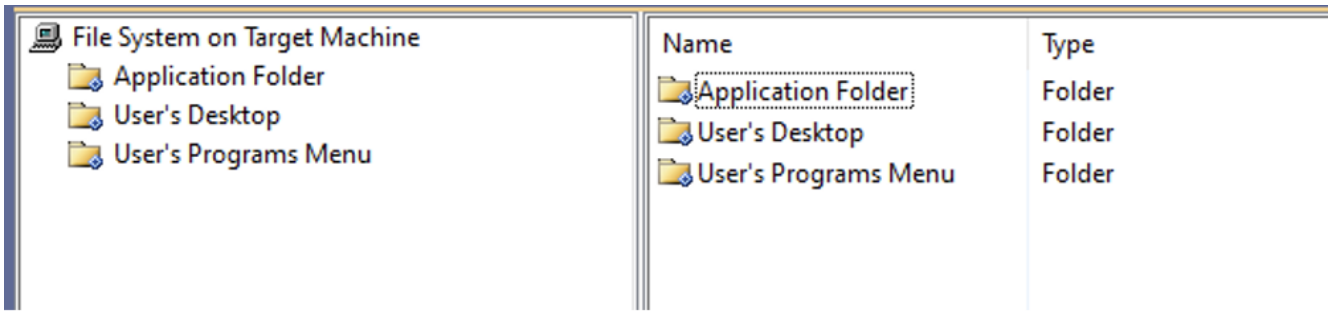


Рисунок 2.7 - Файлова система проекту

Крім того, у цій папці містяться підкаталоги ‘img’ і ‘updates’, які, відповідно, містять графічні ресурси програми та файли, пов’язані з оновленнями. Також у “Application Folder” (рис. 2.8) включено підпроєкт, що відповідає за консольне тестування швидкості читання та запису на диск.

Папка “User’s Desktop” містить лише ярлик, який посилається на виконуваний файл програми, розміщений в основній директорії за стосунку (рис.2.9). Вона визначає, які елементи з’являться на робочому столі користувача після завершення встановлення. У цьому випадку, після інсталяції користувач отримає зручний ярлик із назвою програми, що забезпечує швидкий доступ до її запуску без необхідності переходу до місця її збереження.

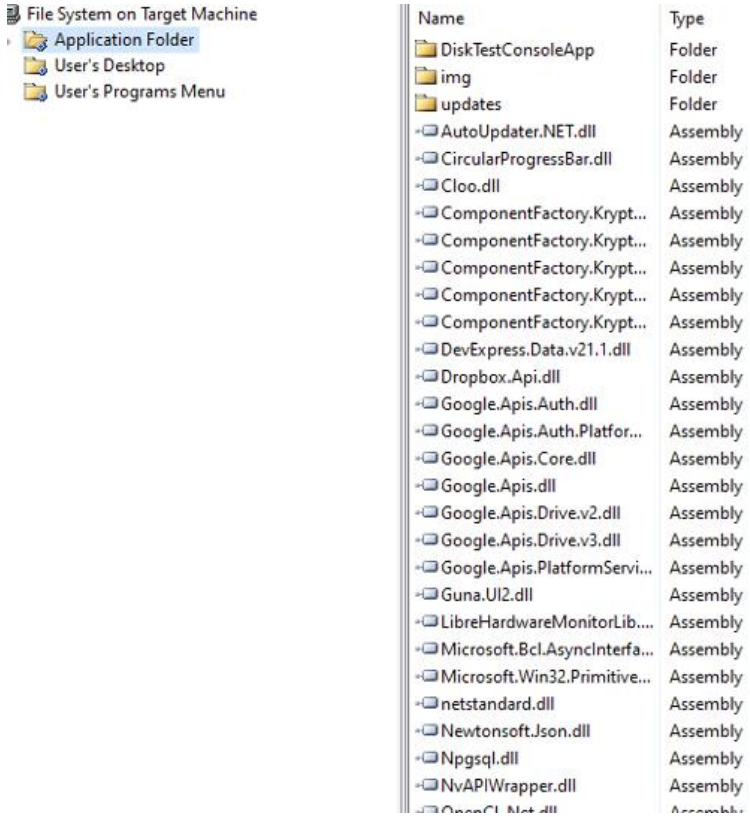


Рисунок 2.8 - Вміст папки ‘Application Folder’

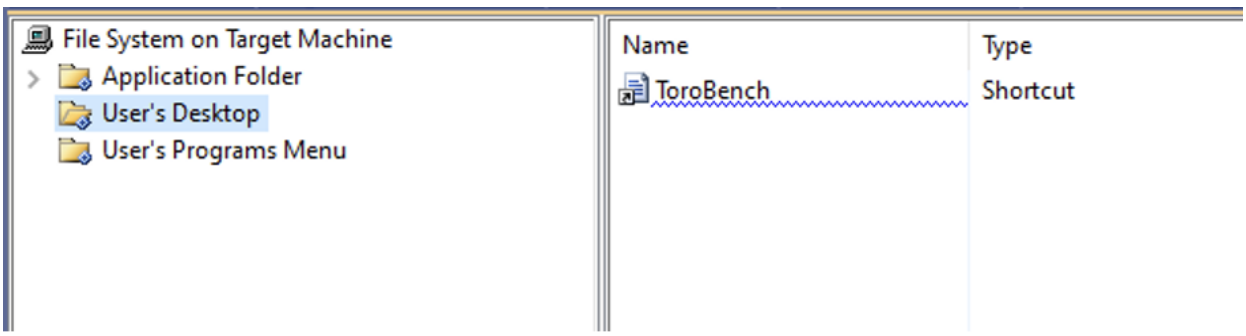


Рисунок 2.9 Вміст папки ‘User’s Desktop’

Папка “User’s Programs Menu” також містить лише ярлик на основний виконуваний файл програми, що забезпечує зручний доступ до застосунку через меню Пуск операційної системи (рис.2.10). Вона визначає, які елементи будуть додані до цього меню після встановлення.

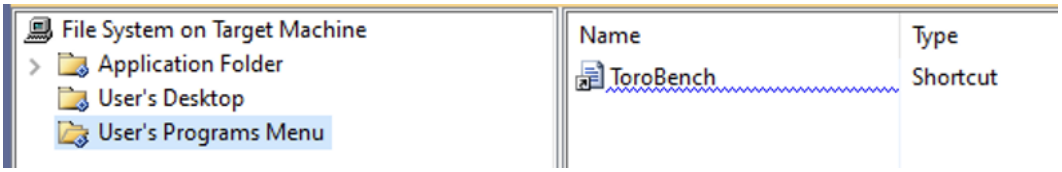


Рисунок 2.10 - Вміст папки ‘User’s Programs Menu’

У результаті, після інсталяції користувач матиме в меню Пуск ярлик для швидкого запуску програми, що дозволяє легко відкривати її у будь-який момент.

3 РЕАЛІЗАЦІЯ КОМП’ЮТЕРНОЇ СИСТЕМИ

У наступних підрозділах буде представлено опис інтерфейсу програми з точки зору користувача, доповнений відповідними зображеннями. Крім того, буде здійснено порівняння результатів тестування з аналогічними бенчмарками для оцінки точності та ефективності роботи застосунку.

3.1 Основні розділи системи

На головному вікні програми міститься інформація про реєстрацію. На зображенні представлено інтерфейс реєстрації або входу для програми під назвою "SystemInf". Інтерфейс виконаний у мінімалістичному стилі з сірою прямокутною формою по центру на світло-зеленому фоні з округлими геометричними

елементами (рис. 3.1). У верхній частині форми розміщено великий чорний заголовок "SystemInf". Нижче знаходяться два поля для введення даних: поле "Username" з прикладом заповнення "info@systeminfo.com" та поле "Password" з прихованим паролем у вигляді зірочок. Під полями розташовані дві зелені кнопки - "Sign Up" для реєстрації та "Get started" для початку роботи, розділені словом "or". Справа від форми розміщена декоративна іконка у вигляді спідометра або системного датчика з кольоровою круговою шкалою, що переходить від синього через жовтий до червоного кольору, що символізує моніторинг системних показників. Загальний дизайн свідчить про те, що це стартова сторінка програми для діагностики або моніторингу комп'ютерних систем.

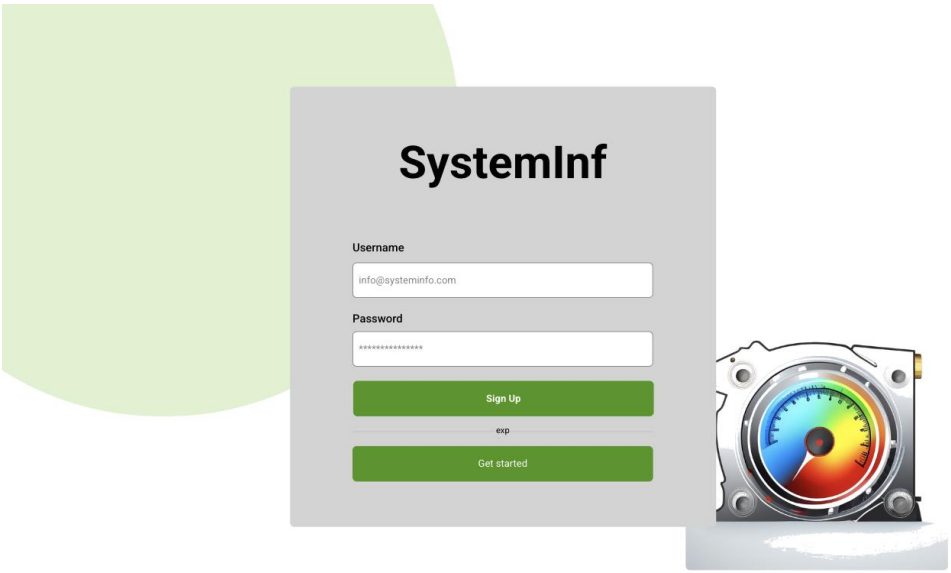


Рисунок 3.1 - Вигляд головного вікна програми

3.2 Тестування процесора та відеокарти

Тестування центрального та графічного процесорів виконується у вікні ProgressForm, яке має однаковий інтерфейс для обох типів тестів (рис. 3.2). У цьому ж вікні здійснюється перевірка в режимах Single-Core та Multi-Core:

спочатку запускається однопотокове тестування, після чого виконується багатопотокове [14]. Користувач може в будь-який момент зупинити процес, а відображення прогресу дозволяє контролювати поточний етап виконання тесту (рис.3.3).

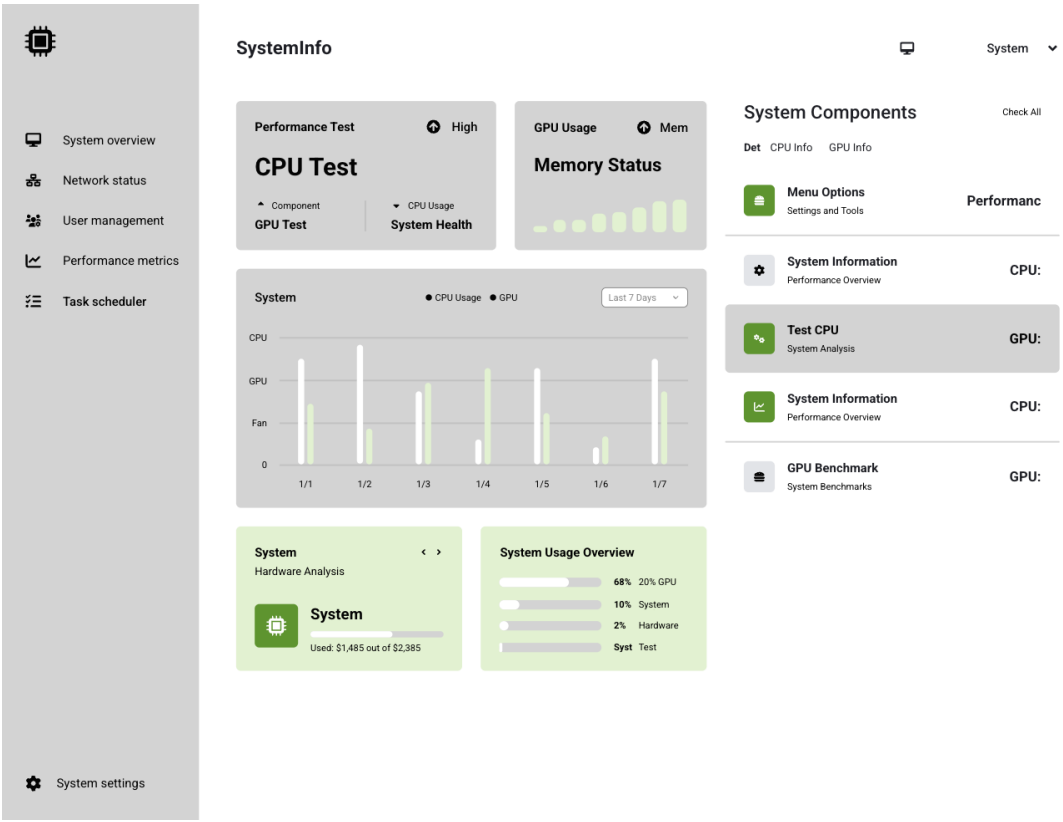


Рисунок 3.2 - Вікно тестування CPU та GPU

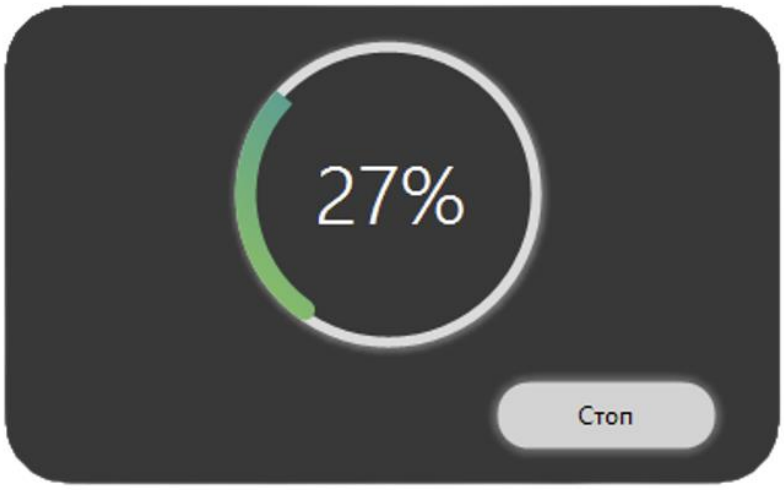


Рисунок 3.3 - Вікно тестування CPU та GPU

3.3 Тестування системи

Вікно тестування відображає користувачеві детальну інформацію про CPU та GPU [15], включаючи відсоток навантаження, температуру та оцінку продуктивності кожного компонента, які оновлюються в режимі реального часу (рис. 3.4). Додатково виводяться граничні значення зазначених параметрів, що дозволяє виявляти можливі аномалії. У вікні також зазначено тривалість поточного тестування. За потреби користувач може в будь-який момент зупинити процес перевірки.

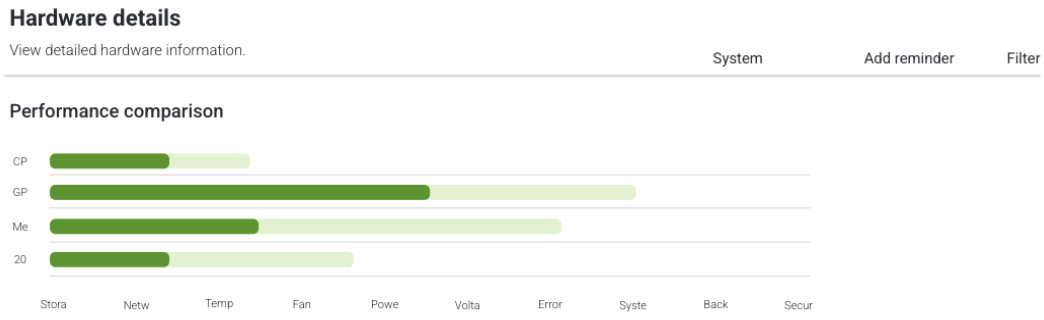


Рисунок 3.4 - Вікно стрес-тестування

У вікні тестування також відображається інформація про диск комп’ютера. Завантаження диска показується у відсотках і оновлюється в реальному часі безпосередньо в інтерфейсі програми. Температура накопичувача подається в градусах Цельсія. Оцінка тестування, розташована в останньому блоці сенсорів і показників диска, відображає поточну швидкість послідовного читання даних. Для кожного сенсора встановлено граничні значення: мінімальні - для швидкості читання, максимальні - для температури та рівня завантаження диска.

У вікні рейтингу відображається інформація про найвищі результати систем з відповідними апаратними конфігураціями. Користувач може сортувати дані таблиці за різними стовпцями, а також здійснювати пошук за конкретними

значеннями. Це забезпечує зручну організацію даних і швидкий доступ до потрібної інформації в рейтингу (рис. 3.5).

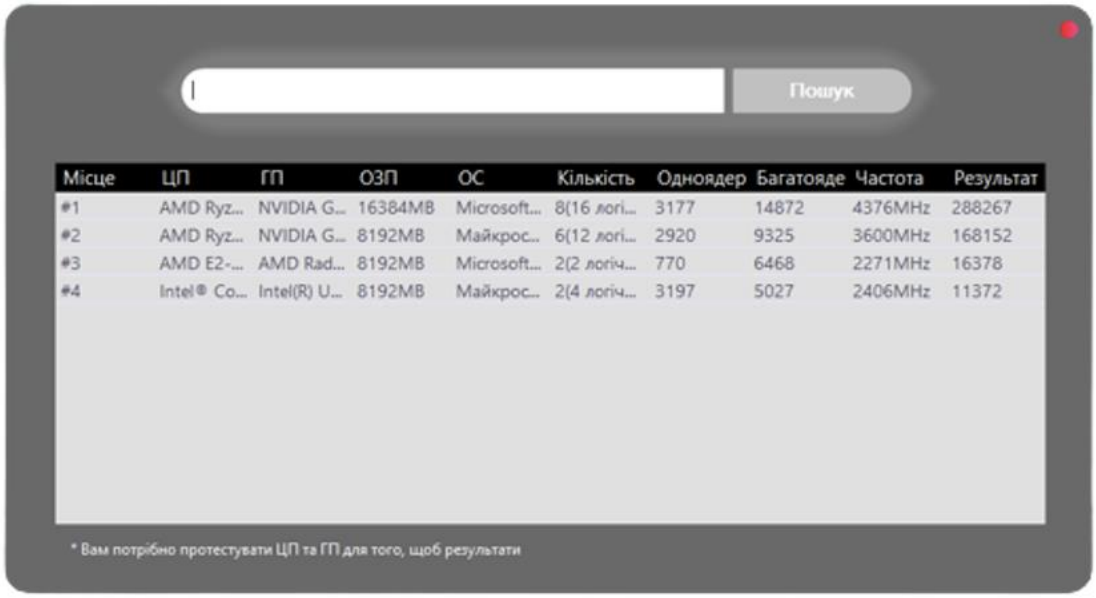


Рисунок 3.5 - Вікно рейтингу

У вікні програми реалізовано функцію пошуку результатів тестування. Користувач може ввести запит у текстове поле та розпочати пошук, натиснувши кнопку "Find", що дозволяє швидко знаходити потрібні записи в таблиці.

Варто зазначити, що результати з’являються в рейтингу лише після завершення тестування як процесора, так і відеокарти. Якщо протестовано лише один із цих компонентів, дані не потрапляють до таблиці рейтингу. Такий підхід дозволяє уникнути наявності неповних записів і забезпечує більш точне уявлення про конфігурацію системи, на якій проводилося тестування.

Вікно тестування диску відображає дані щодо швидкості запису та читання, а також модель накопичувача, яка вказується у верхній частині вікна над елементами, що демонструють прогрес виконання кожного підтипу тесту.

Компоненти прогресу відображаються різними кольорами, що підвищує зручність та зрозумілість для користувача. Під час виконання тестування користувач бачить відсотковий індикатор завершення, який дозволяє відстежувати хід процесу, а по завершенню - отримує підсумкові результати.

Зупинити тестування вручну неможливо, оскільки його виконання реалізовано в окремому процесі.

Вікно тестування швидкості інтернету за зовнішнім виглядом подібне до вікна перевірки швидкості читання та запису диску (див. 3.6). У цьому вікні відображаються два кольорово розмежовані компоненти, що покращує сприйняття інформації користувачем. Вони демонструють швидкість завантаження та відвантаження даних через мережу. У верхній частині вікна вказується модель мережевого пристрою, який забезпечує підключення до Інтернету на комп’ютері користувача.

Рисунок 3.6 - Вікно тестування диску

Швидкість завантаження та відвантаження даних в Інтернеті оновлюється в реальному часі та відображається послідовно (рис.3.7). Варто зазначити, що анімація компонента, відповідального за візуалізацію прогресу тестування, змінює свою швидкість залежно від фактичної швидкості з’єднання на комп’ютері користувача.

Параметри ping та jitter виводяться у нижній частині вікна тестування і відображаються першими - ще до початку перевірки швидкості завантаження та

відвантаження. Вони надають користувачу попередню оцінку стабільності мережевого з’єднання.

Рисунок 3.7 - Вікно тестування швидкості інтернету

3.4 Оновлення програмного забезпечення системи

У вікні автооновлення програми відображається інформація про поточну версію застосунку, а також доступні параметри керування оновленням. Користувач може переглянути список запланованих змін у майбутніх версіях, обрати варіант пропуску конкретного оновлення або ж відразу розпочати його встановлення [16]. Окрім цього, реалізовано додаткові можливості за допомогою бібліотеки AutoUpdater.NET, що забезпечує гнучке та зручне управління процесом оновлення програми (рис. 3.8).

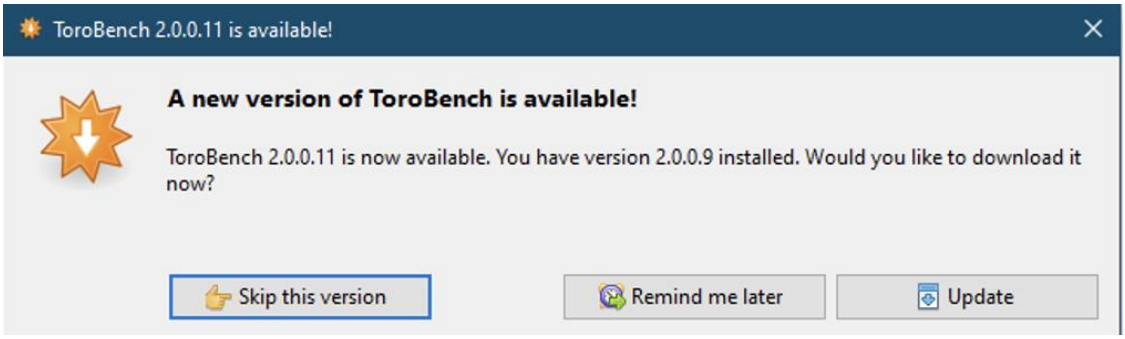


Рисунок 3.8 - Функціонал бібліотеки AutoUpdater.NET

Якщо оновлення відсутнє і на комп’ютері користувача вже встановлена актуальна версія програми, після завершення перевірки на наявність оновлень основна програма відображає окреме діалогове вікно з відповідним повідомленням (рис. 3.9). Це інформує користувача про те, що оновлення не потрібне, оскільки використовується найновіша версія застосунку.

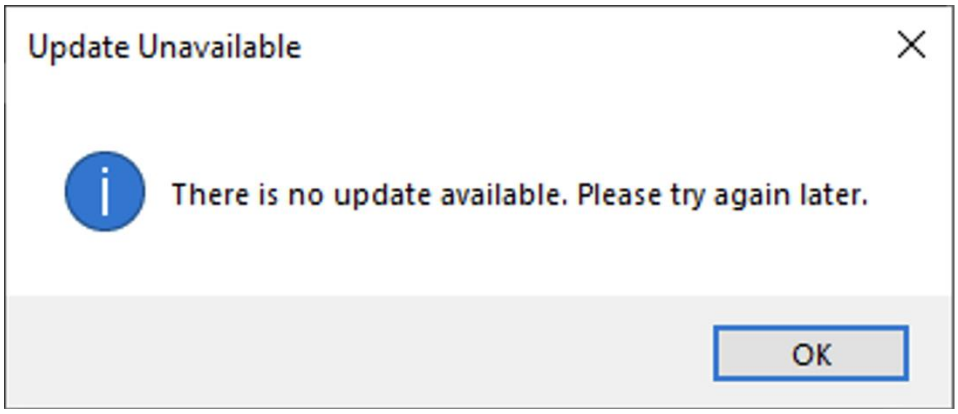


Рисунок 3.9 - Діалогове вікно про відсутність оновлень

Зміна теми інтерфейсу програми спричиняє істотні зміни кольорової гами, що були спеціально підібрані для покращення користувацького досвіду. Вибір теми визначається умовами використання програми (рис.3.10). Наприклад, під час тестування пристрою в добре освітленому приміщенні або на відкритому повітрі доцільно використовувати світлу тему - це забезпечує кращу видимість і чіткість елементів інтерфейсу. У темних умовах або в нічний час доцільно активувати темну тему, що сприяє комфортному користуванню та зменшує навантаження на зір.

У сучасних умовах існують сторонні додатки, які здатні автоматично змінювати тему інтерфейсу залежно від часу доби або індивідуального розкладу користувача. У таких випадках доцільно обрати опцію "System", яка використовує системну тему операційної системи та автоматично адаптує вигляд програми відповідно до поточних налаштувань системи.

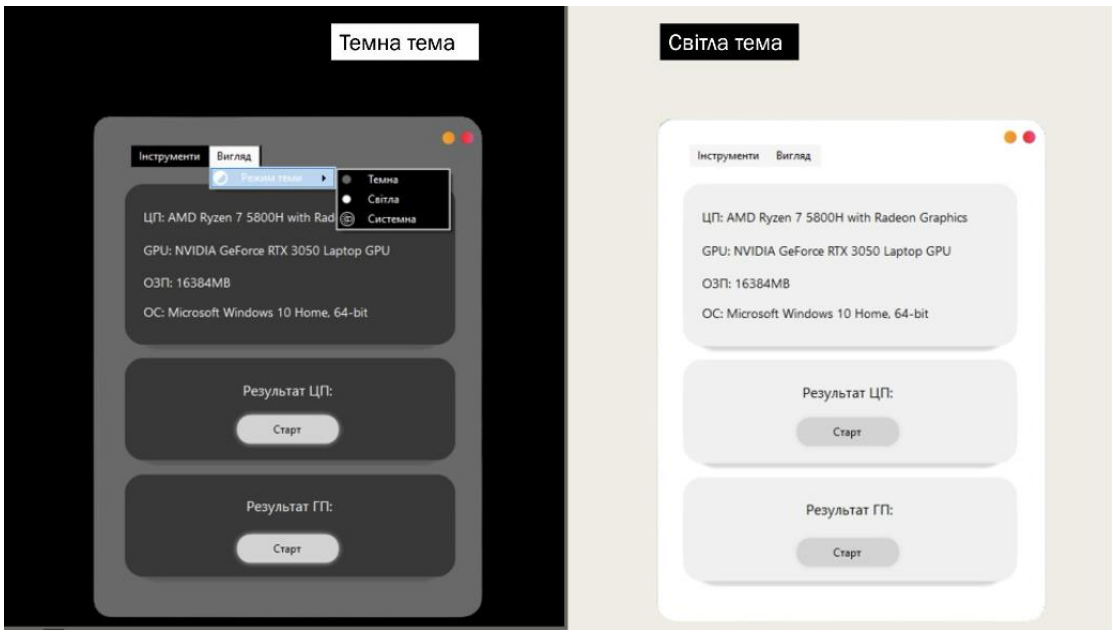


Рисунок 3.10 - Вигляд головного вікна світлої та темної теми

3.4 Результати роботи системи

У наступних підрозділах представлено порівняльний аналіз із відомими світовими аналогами. Розглянуто результати їхніх тестувань, а також проаналізовано загальний функціонал кожного з інструментів.

Geekbench 5 - один із найвідоміших та найпоширеніших бенчмарків у світі, який широко використовується для оцінки продуктивності центрального та графічного процесорів на різних платформах (рис.3.11) [17].

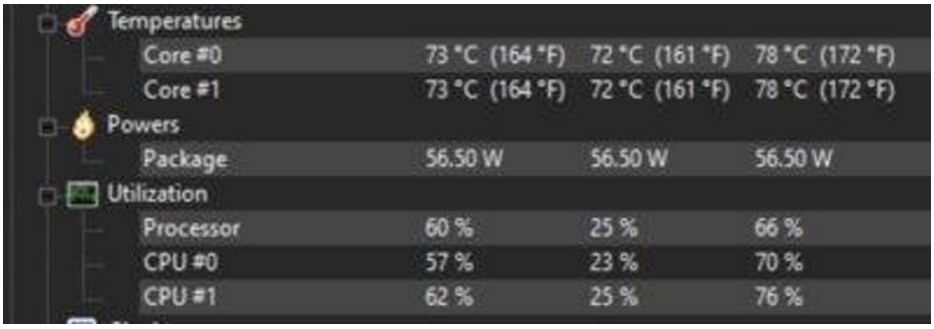


Рисунок 3.11 - Навантаження ЦП при тестуванні Geekbench 5

Порівнюючи ToroBench (рис.3.12) із популярним аналогом Geekbench 5, можна зробити такі висновки. Насамперед, на відміну від більшості аналогів, зокрема Geekbench 5, ToroBench є повністю безкоштовним і пропонує ширший функціонал. Зокрема, до нього входять стрес-тестування CPU та GPU, а також інтегрована система рейтингів, яких немає у Geekbench 5.

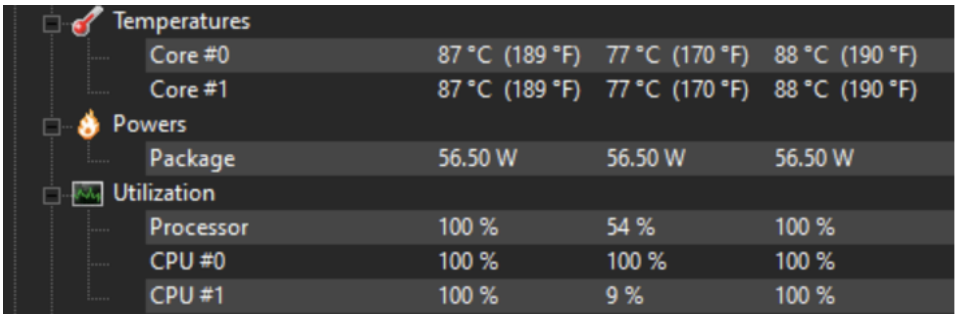


Рисунок 3.12 - Навантаження ЦП при тестуванні ToroBench

Крім того, під час тестування в ToroBench навантаження на компоненти та їх температурні показники зазвичай вищі, ніж у Geekbench. Це дозволяє виявляти пікову продуктивність і перевіряти стабільність роботи апаратного забезпечення за умов інтенсивних навантажень.

Таким чином, ToroBench є конкурентоспроможним рішенням для тестування продуктивності CPU та GPU, забезпечуючи поглиблений аналіз та розширені можливості, що робить його ефективним інструментом для користувачів, зацікавлених у детальній оцінці роботи свого ПК.

Порівнюючи ToroBench з іншими відомими бенчмарками, такими як CPU-Z та GPU-Z, можна зробити висновок, що ці застосунки мають обмежений функціонал. Зокрема, у CPU-Z та GPU-Z відсутні додаткові можливості, такі як вбудований онлайн-рейтинг або тестування швидкості хешування GPU. До того ж, ці програми розділені за функціональністю - CPU-Z призначений виключно для процесора, а GPU-Z - для графічного процесора, що видно вже з їхніх назв (рис. 3.13 та рис. 3.14).

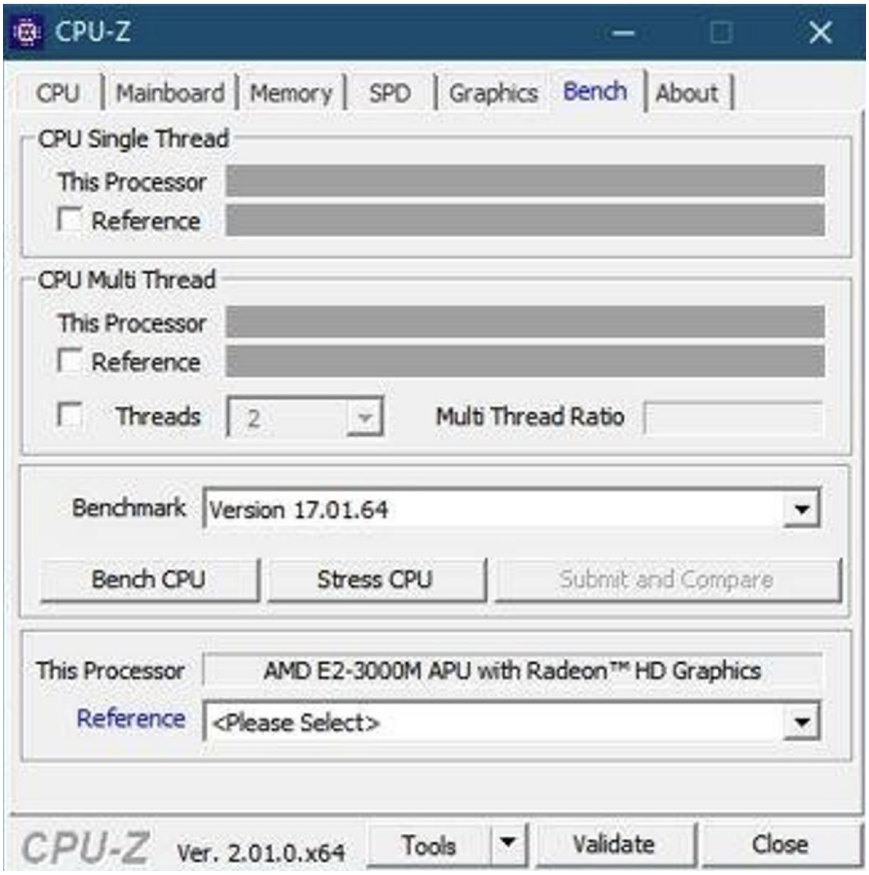


Рисунок 3.13 - Вікно бенчмарка CPU-Z

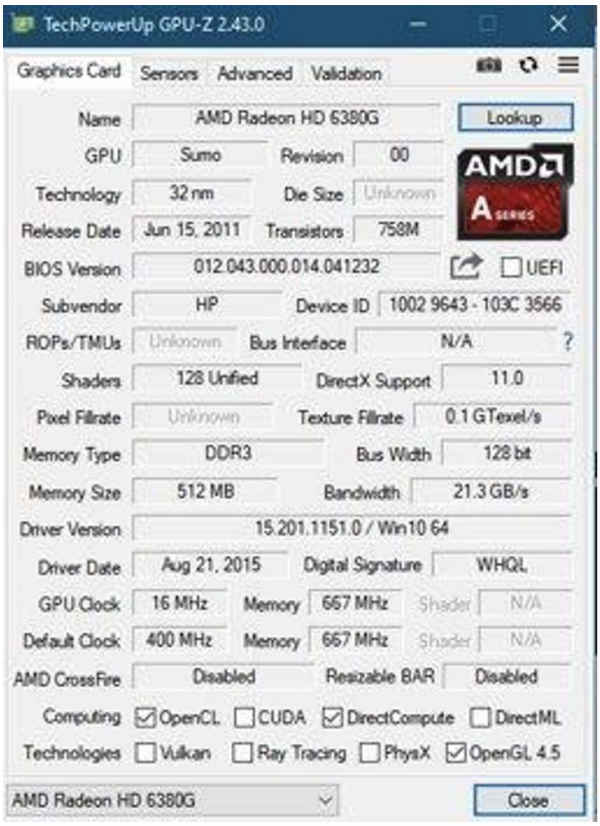


Рисунок 3.14 - Вікно бенчмарка GPU-Z

CrystalDiskMark - це безкоштовний інструмент для вимірювання швидкості читання та запису на пристроях зберігання даних. Він дозволяє оцінити швидкодію різних типів накопичувачів, зокрема жорстких дисків, твердотільних накопичувачів та USB-накопичувачів (рис.3.15) [18].

Програма підтримує кілька режимів тестування, зокрема послідовний і випадковий доступ до даних. Крім того, користувач може змінювати розмір блоку даних і кількість повторень тесту, що забезпечує гнучкість налаштувань та можливість глибшого аналізу продуктивності накопичувача.

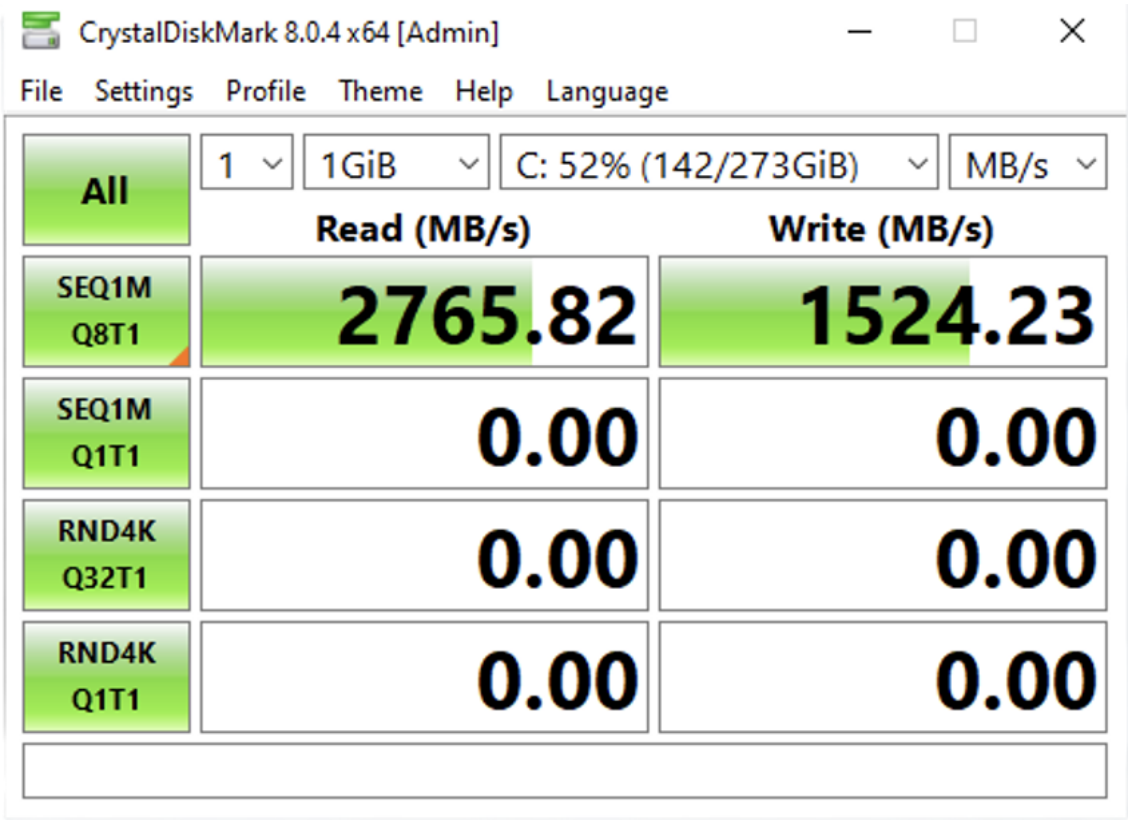


Рисунок 3.15 - Вікно програми CrystalDiskMark

CrystalDiskMark є широко вживаним інструментом для оцінки продуктивності накопичувачів, що дозволяє підібрати оптимальну модель для завдань, які вимагають високої швидкості передачі даних - зокрема, для відеомонтажу, ігор чи обробки великих обсягів інформації.

Водночас основною перевагою тестування швидкості читання та запису, реалізованого в застосунку ToroBench, є його триваліший і більш глибокий

характер, а також покращена взаємодія з користувачем під час самого процесу тестування. Відображення прогресу в реальному часі, інформативний інтерфейс і майже ідентичні результати порівняно з CrystalDiskMark, отримані за коротший час, роблять ToroBench вигідною альтернативою з точки зору зручності та ефективності тестування.

ВИСНОВКИ

Було розроблено десктопний застосунок із розширеним функціоналом для тестування продуктивності персональних комп’ютерів. У процесі створення було освоєно принципи десктопної розробки, реалізовано інтеграцію з хмарними сервісами, а також використано низку бібліотек, специфічних для реалізації бенчмаркінгу. Проведено аналіз роботи апаратних компонентів ПК та їх взаємодії на програмному рівні.

Станом на сьогодні застосунок повністю підтримується розробником: регулярно випускаються оновлення, а вихідний код відкритий для всіх охочих на GitHub. Програму можна інсталювати як стандартний застосунок на будь-який комп’ютер з ОС Windows. Вона забезпечує швидкий і зручний спосіб оцінки продуктивності системи з можливістю порівняння результатів з іншими користувачами через онлайн-рейтинг.

Основні можливості додатку:

– тестування продуктивності CPU (арифметичні, рядкові та інші типи операцій);

– аналіз продуктивності GPU під час обробки відео;

– перевірка швидкості читання та запису даних на диск;

– тестування швидкості інтернет-з'єднання;

– моніторинг температури та виявлення тротлінгу під навантаженням;

– формування онлайн-рейтингу продуктивності;

– функція автоматичного оновлення.

Крім цього, додаток дозволяє зберігати результати тестів для подальшого аналізу та порівняння, а також передбачає режим автоматичного запуску тестування, що дає змогу виконувати тривалі навантаження без участі користувача - зручно для перевірки стабільності системи.

Загалом, програма є ефективним інструментом оцінювання продуктивності ПК, який вирізняється широким набором функцій та простотою використання.

Застосунок постійно вдосконалюється й залишається доступним для всіх охочих. Основна мета - створення безкоштовного, зручного та функціонального бенчмарку для Windows - була повністю реалізована.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. What are CIS Benchmarks? URL:https://aws.amazon.com/whatis/cis-benchmarks (date of access: 17.04 2025).

2. Як провести бенчмаркінг ПК? URL: https://blog.acer.com/ua/discussion/2766/yak-provesti-benchmarking-pk (дата звернення 14.05.2025).

3. Одрі Роггенкамп, Ієн Рутковскі, Ніта Руткоскі. Benchmark Series: Microsoft Access 2019 Levels 1&2.EMC Paradigm . 2019.

4. ToroBench. URL: https://github.com/SerhiyToroniy/ToroBench (date of access: 05.04.2025).

5. .NET Framework. URL: https://uk.wikipedia.org/wiki/.NET_Framework (date of access: 17.04.2025).

6. Guna Framework is here for a better and growing user experience. URL: https://gunaui.com/ (date of access: 11.05.2025).

7. CIS Benchmarks List. URL: https://www.cisecurity.org/cis-benchmarks (date of access: 24.04.2025).

8. Для чого потрібна програма для тестування графічної карти? URL: https://www.guru99.com/uk/best-benchmark-software.html (дата звернення 29.04.2025).

9. Найкращі програми і програмне забезпечення для стрес-тесту GPU. URL: https://www .ccleaner.com/speccy/ (дата звернення 13.05.2025).

10. Перевірка швидкості Інтернет URL: https://www.chereda.net/kyiv/other/159 (дата звернення 29.04.2025).

11. Найкраще програмне забезпечення GPU Benchmark для ПК. URL: https://www.guru99.com/uk/best-benchmark-software.html (дата звернення 18.04.2025).

12. Dropbox Forms. URL: https://sign.dropbox.com/uk-UA/about/terms/service-specific-terms (date of access: 01.05.2025).

13. Що таке Net Framework? URL: https://intelserv.net.ua/ blog/material/id/247 (дата звернення 28.04.2025).

14. Авраменко А.С., Авраменко В.С., Косенюк Г.В. Тестування програмного забезпечення. Навчальний посібник. – Черкаси: ЧНУ імені Богдана Хмельницького, 2017. – 284 с.

15. Basemark GPU. URL: https://www.basemark.com/benchmarks/basemark-gpu/ (дата звернення 23.05.2025).

16. AutoUpdater.NET. URL: https://github.com/ravibpatel/AutoUpdater.NET (date of access: 18.05.2025).

17. Geekbench 5. URL: https://pingvin. pro/gadgets/ news-gadgets/geekbench-5-new-benchmark-by-primate-labs.html (date of access: 19.05.2025).

18. CrystalDiskMark? URL: https://kvitnik.klipsa.v.ua/chi-tochniy-crystaldisk mark/ (date of access: 29.04.2025).

ДОДАТОК А
КОНФІГУРАЦІЯ АВТООНОВЛЕНЬ

Лістинг А.1 - Файл-конфігуратор автооновлень

```
using AutoUpdaterDotNET;
using Dropbox.Api;
using Dropbox.Api.Files;
using System;
using System.IO;
using System.Threading.Tasks;

namespace YourApplicationNamespace
{
    public class UpdateConfigurator
    {
        private readonly string _dropboxToken;
        private readonly string _localXmlPath;
        private readonly string _dropboxXmlPath;
        private readonly string _appVersion;

        public UpdateConfigurator(string dropboxToken, string
localXmlPath, string dropboxXmlPath, string appVersion)
        {
            _dropboxToken = dropboxToken;
            _localXmlPath = localXmlPath;
            _dropboxXmlPath = dropboxXmlPath;
            _appVersion = appVersion;
        }

        public void InitializeAutoUpdater()
        {
            // Set the update XML URL to your Dropbox shared link
            // Note: Make sure to replace 'dl=0' with 'dl=1' at the
end of the URL to get direct download link

AutoUpdater.Start("https://www.dropbox.com/s/yourdirectlink/updateCo
nfig.xml?dl=1");

            // Configure AutoUpdater events
            AutoUpdater.CheckForUpdateEvent +=
AutoUpdaterOnCheckForUpdateEvent;
            AutoUpdater.ApplicationExitEvent +=
AutoUpdater_ApplicationExitEvent;
        }

        private void
AutoUpdaterOnCheckForUpdateEvent(UpdateInfoEventArgs args)
        {
```

```
            if (args.IsUpdateAvailable)
            {
                Console.WriteLine($"Update available:
{args.CurrentVersion} -> {args.InstalledVersion}");
                // Let AutoUpdater handle the update process
            }
            else
            {
                Console.WriteLine("No update available.");
            }
        }

        private void AutoUpdater_ApplicationExitEvent()
        {
            // Code to execute before application exits for update
            Console.WriteLine("Application is closing for
update...");
        }

        public async Task UpdateConfigurationFile(string newVersion,
string newDownloadUrl, string changelogUrl, bool mandatory)
        {
            // Update local XML file
            string xmlContent = CreateXmlContent(newVersion,
newDownloadUrl, changelogUrl, mandatory);
            File.WriteAllText(_localXmlPath, xmlContent);

            // Upload to Dropbox
            await UploadToDropbox(_localXmlPath, _dropboxXmlPath);
        }

        private string CreateXmlContent(string version, string
downloadUrl, string changelogUrl, bool mandatory)
        {
            // Calculate MD5 of your zip file
            string md5 = CalculateMD5(downloadUrl);

            return $@"<?xml version=""1.0"" encoding=""UTF-8""?>
<item>
  <version>{version}</version>
  <url>{downloadUrl}</url>
  <changelog>{changelogUrl}</changelog>
  <mandatory>{mandatory.ToString().ToLower()}</mandatory>
  <checksum algorithm=""MD5"">{md5}</checksum>
</item>";
        }

        private string CalculateMD5(string filePath)
        {
            // If the file is local
            if (File.Exists(filePath))
            {
```

```
                using (var md5 =
System.Security.Cryptography.MD5.Create())
                using (var stream = File.OpenRead(filePath))
                {
                    byte[] hash = md5.ComputeHash(stream);
                    return BitConverter.ToString(hash).Replace("-",
"").ToLowerInvariant();
                }
            }

            // For this example just return empty - in real app
you'd download the file first
            return string.Empty;
        }

        private async Task UploadToDropbox(string localFilePath,
string dropboxPath)
        {
            using (var dbx = new DropboxClient(_dropboxToken))
            using (var fileStream = File.Open(localFilePath,
FileMode.Open))
            {
                var updated = await dbx.Files.UploadAsync(
                    dropboxPath,
                    WriteMode.Overwrite.Instance,
                    body: fileStream);

                Console.WriteLine($"Upload complete. File ID:
{updated.Id}");
            }
        }

        public async Task<string> GetSharedLink(string dropboxPath)
        {
            using (var dbx = new DropboxClient(_dropboxToken))
            {
                var share = await
dbx.Sharing.CreateSharedLinkWithSettingsAsync(dropboxPath);
                // Convert to direct download link by replacing dl=0
with dl=1
                return share.Url.Replace("dl=0", "dl=1");
            }
        }
    }
}
```

Національний університет «Запорізька політехніка»
Кафедра «Комп'ютерних систем та мереж»

ДИПЛОМНИЙ ПРОЄКТ

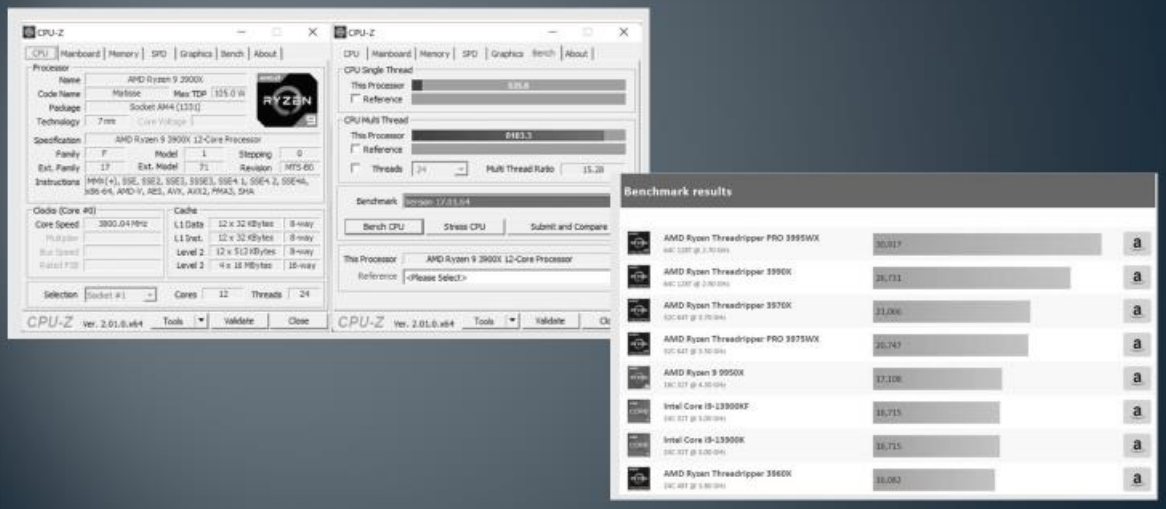
НА ТЕМУ: «РОЗРОБКА СИСТЕМИ ДЛЯ ОЦІНЮВАННЯ ПРОДУКТИВНОСТІ ПЕРСОНАЛЬНОГО КОМП'ЮТЕРА

Доповідач: студент групи КНТз – 512сп
Вікторія КІКЄЄВА
Керівник: Матвій ІЛЬЯШЕНКО

Мета проєкту - створення програмного продукту для швидкого та зручного тестування продуктивності персональних комп'ютерів, з функціоналом онлайн-рейтингу для порівняння результатів з іншими користувачами.

Завданням проєкту є розробка безкоштовного бенчмарку з простим інтерфейсом та швидким тестуванням продуктивності ПК.

Бенчмарк (від англійського "benchmark" - орієнтир, еталон) - це еталонний показник, який використовується для порівняння результатів, показників або характеристик. Він може стосуватися різних сфер, таких як інформатика, фінанси та бізнес.



ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ:

- ✓ Torobench;
- ✓ .NET Framework 4.6.1;
- ✓ Guna Framework .

Бібліотеки:

- ✓ Parallel та Stopwatch.

Для моніторингу навантаження на CPU - бібліотека OpenHardwareMonitor

Інструменти :

- ✓ OpenCL;
- ✓ CUDAfy.NET .

Засоби для тестування системи

- ✓ метод WhileCPU;
- ✓ нескінченний цикл while:
  - ✓ Parallel.ForEach і Parallel.For:
    - ✓ Parallel.ForEach обробляє елементи списку L у паралельному режимі.;
    - ✓ Parallel.For працює паралельно, виконуючи мільйон ітерацій для кожного елемента списку L., обчислення, що інтенсивно навантажують CPU.

Тестування навантаження швидкості Інтернету

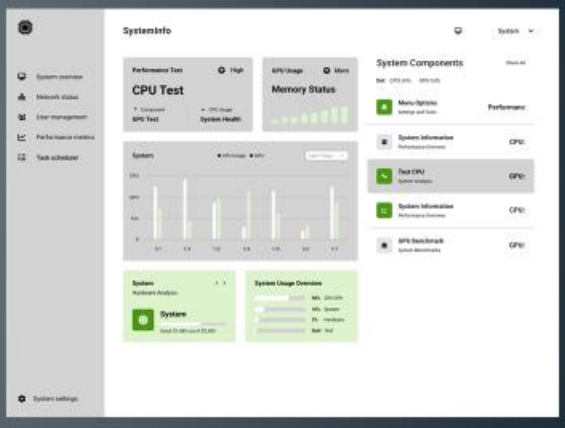
- ✓ клієнт WebClient:
  - ✓ GetUploadSpeed;
  - ✓ UploadDataAsync.

Тестування диска клас :

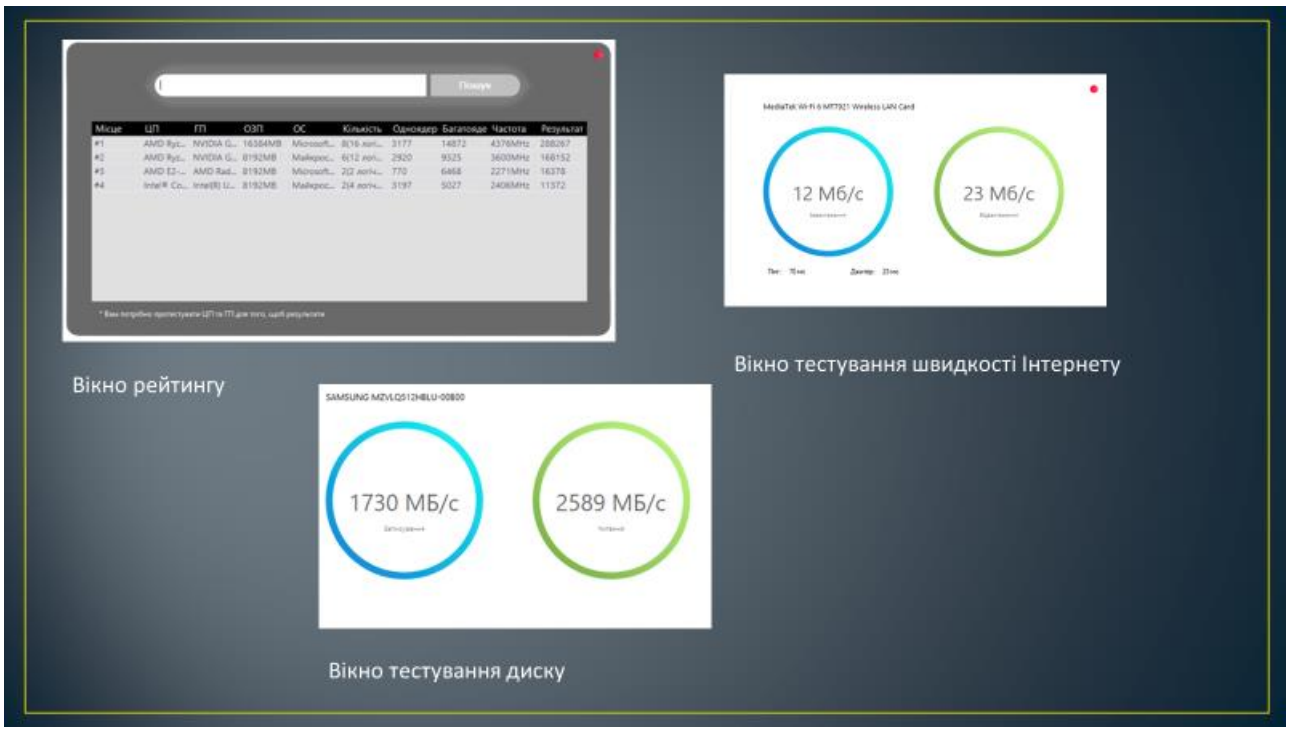
- ✓ ManagementObjectSearcher із бібліотеки System.Management

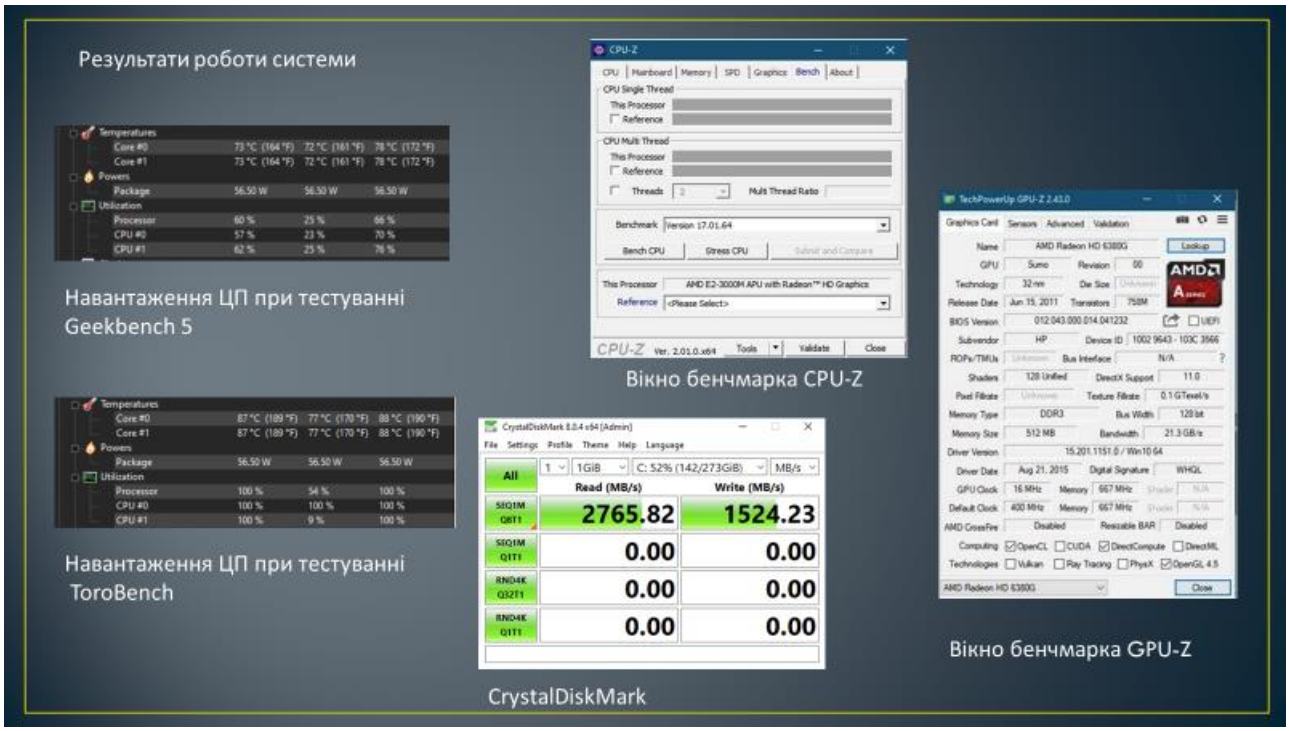
РЕАЛІЗАЦІЯ КОМП'ЮТЕРНОЇ СИСТЕМИ

Реєстрація або вхід до програми



Вікно тестування CPU та GPU


Вікно рейтингу
12 Мб/с
23 Мб/с
Вікно тестування швидкості Інтернету
1730 МБ/с
2589 МБ/с
Вікно тестування диску


Результати роботи системи
Навантаження ЦП при тестуванні Geekbench 5
Вікно бенчмарка CPU-Z
Навантаження ЦП при тестуванні ToroBench
CrystalDiskMark
Вікно бенчмарка GPU-Z

ВИСНОВОК

Було розроблено десктопний застосунок із розширеним функціоналом для тестування продуктивності персональних комп'ютерів. У процесі створення було освоєно принципи десктопної розробки, реалізовано інтеграцію з хмарними сервісами, а також використано низку бібліотек, специфічних для реалізації бенчмаркінгу. Проведено аналіз роботи апаратних компонентів ПК та їх взаємодії на програмному рівні.

Доклад завершено.

ДЯКУЮ ЗА УВАГУ !