Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра ЕОМ



**Пояснювальна записка**

до курсового проєкту

з дисципліни «Системне програмне забезпечення»

на тему : «Розробка програмного забезпечення для тестування

продуктивності кешу та оперативної пам’яті комп’ютера»

Виконав:

ст. гр. КІ-34

Палій Назарій

Перевірив:

Мархивка В.С.

Львів 2023

АНОТАЦІЯ

**Мій проєкт має на меті розробити програмне забезпечення, спрямоване на для тестування продуктивності кешу та оперативної пам’яті комп’ютера**.

В першому розділі виконується аналітичний огляд методів відновлення даних та огляд існуючих розробок, їхні проблеми та обмеження.

В другому розділі розглядається розробка архітектури програми, зокрема розбиття утиліти на модулі, принципи Model-View-ViewModel(MVVM), використання об'єктно-орієнтованого підходу (ООП), принципи доступу до кешу процесора. Для реалізації програми використовується мова програмування С# з використанням графічної системи WPF.

В третьому розділі детально описано реалізацію функціональності утиліти продуктивності кешу і оперативної пам’яті.

В четвертому розділі детально продемонстровано, описано графічний інтерфейс утиліти, в результаті тестування програма працює стабільно і забезпечує можливість тестування пам’яті комп’ютера. Розроблена утиліта є навчальним проєктом, який вже можна використовувати для тестування пам’яті комп’ютера.

Зміст

Бла бла бла

**Завдання на курсовий проект**

**Завдання**: Утиліта тестування продуктивності кешу та оперативної пам’яті комп’ютера.

1. **Функції, що підлягають реалізації**:
   1. **Тестування швидкості кешу та оперативної пам’яті**: розроблена програма вимірює швидкість трьох рівнів кешу та швидкість оперативної пам’яті. Вимірювання проходять у чотирьох режимах: читання, запис, копіювання та затримка.
   2. **Базові дані про характеристики комп’ютера**: програма визначає характеристики процесора, кешу, оперативної пам’яті та материнської плати.
   3. **Запис історії тестування** : Усі тестування, що проводились, зберігаються у файлі з історією вимірюваня та сортуються за датою. Вони доступні користувачу в окремому вікні.

**Платформа**: ОС Windows

**Мова програмування**: C#

**Графічний користувацький інтерфейс** : Windows Presentation Foundation.

Загальні вимоги:

1. Передбачити використання піктограм для швидкого запуску утиліт.

2. Реалізувати рortable-версію та інсталятор розробленої утиліти.

3. Реалізувати help-систему для розробленої утиліти.

4. Обов'язкова наявність графічної частини

ВСТУП

Тестування швидкості кешу та оперативної пам'яті допомагає оцінити продуктивність комп'ютерної системи. Швидкість доступу до кешу та оперативної пам'яті має прямий вплив на загальну швидкість роботи комп'ютера. Чим швидше можна отримати доступ до необхідних даних, тим швидше виконується програмне забезпечення та завдання. Тому тестування швидкості кешу та оперативної пам'яті є важливим завданням у сучасних комп’ютерах.

Мій проєкт має на меті розробити програмне забезпечення, спрямоване на тестування швидкості кешу та оперативної пам'яті. Основним завданням програми є визначення швидкості роботи кешу та оперативної пам'яті на комп’ютерах з операційною системою Windows. Моя програма буде забезпечувати чіткі показники швидкості пам’яті системи.

Програма буде розроблена з використанням мови програмування C#, що дозволить забезпечити оптимальну продуктивність та ефективність роботи програми. Крім того, я надаю особливу увагу інтерфейсу програми, забезпечуючи його зручність та інтуїтивність для користувача. Це дозволить навіть не досвідченим користувачам легко взаємодіяти з програмою та відновлювати їх втрачені дані.

Моя програма вимірюватиме швидкість за чотирма різними показниками, як читання, запис, копіювання та затримка. Крім того, буде можливість збереження фото з показниками вимірів та перегляд попередніх тестувань.

Завдяки використанню операційної системи Windows, програма забезпечує повну сумісність та оптимальну роботу на сучасних комп'ютерах. Мій проєкт допоможе користувачам ефективно відслідковувати швидкість пам’яті на свому комп’ютері.

1. **Аналітичний огляд**

У цьому розділі проведено аналіз і огляд існуючих розробок і методів у галузі моніторингу мережі комп'ютера з OC Windows. Описано різні підходи, програмні засоби та технології, що використовуються для відновлення даних на накопичувальних пристроях.

* 1. **Огляд методів тестування пам’ті**

Для тестування швидкості кеш-пам'яті та оперативної пам'яті можна використовувати комбінацію синтетичних бенчмарків, реальних робочих навантажень та інструментів моніторингу обладнання. Нижче наведено кілька найпоширеніших методів тестування швидкості кеш-пам'яті та оперативної пам'яті:

1. **Тести пропускної здатності пам'яті:** Тести пропускної здатності пам'яті оцінюють швидкість, з якою дані можуть бути прочитані з оперативної пам'яті і записані в неї. Такі інструменти можуть виміряти максимальну пропускну здатність пам'яті, досягнуту вашою системою. Ці тести передбачають передачу великих обсягів даних і дають уявлення про загальну продуктивність оперативної пам'яті.

2. **Тести затримки кеш-пам'яті:** Тести затримки кеш-пам'яті зосереджені на вимірюванні часу, необхідного для доступу до даних з різних рівнів кеш-пам'яті. Такі інструменти можуть надавати специфічні для кешу бенчмарки для оцінки затримки кешу і частоти звернень до кешу. Ці тести допомагають оцінити ефективність і швидкість реакції ієрархії кешу.

3. **Навантажувальні тести процесора:** Стрес-тести процесора можуть опосередковано оцінити швидкість кеш-пам'яті та оперативної пам'яті, створюючи високе навантаження на процесор і спостерігаючи за його продуктивністю. Такі інструменти можуть генерувати інтенсивні обчислювальні навантаження, які створюють стрес для підсистем кеш-пам'яті та оперативної пам'яті. Інструменти моніторингу можуть допомогти відстежувати продуктивність системи під час цих тестів.

4. **Реальні програми:** Запуск реальних програм, які вимагають багато пам'яті, може допомогти оцінити швидкість кеш-пам'яті та оперативної пам'яті. Наприклад, програмне забезпечення для редагування відео, наукові симуляції або сервери баз даних можуть пред'являти значні вимоги до підсистеми пам'яті. Інструменти моніторингу можуть дати уявлення про використання пам'яті, частоту звернень до кешу та загальну продуктивність під час виконання цих завдань.

5. **Інструменти моніторингу обладнання:** Інструменти моніторингу апаратного забезпечення дозволяють спостерігати за поведінкою різних компонентів системи, включаючи кеш-пам'ять і оперативну пам'ять. Такі інструменти можуть надавати інформацію в режимі реального часу про розмір кешу, частоту звернень до кешу, таймінги пам'яті та інші показники, пов'язані з продуктивністю. Ці інструменти дозволяють відстежувати швидкість кеш-пам'яті та оперативної пам'яті під різними робочими навантаженнями або під час виконання певних тестів.

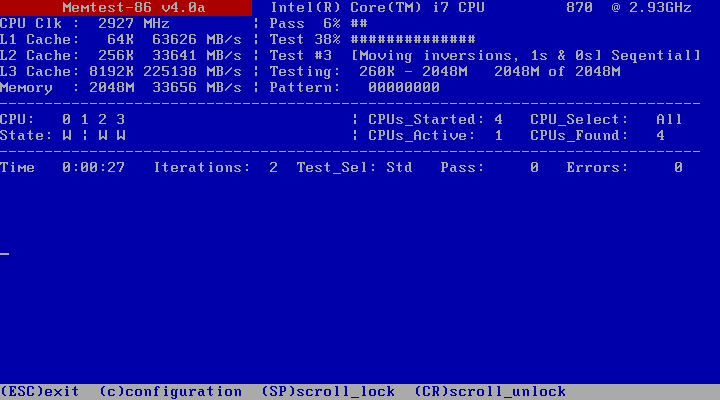
6. **Мікробенчмарки:** Як згадувалося раніше, мікротести можуть бути корисними для тестування швидкості кеш-пам'яті та оперативної пам'яті. Ці тести призначені для ізоляції певних операцій або шаблонів доступу до пам'яті для вимірювання їх продуктивності. Безпосередньо націлившись на операції з кешем і пам'яттю, ви можете оцінити їх швидкість в контрольованих сценаріях.

Під час тестування швидкості кеш-пам'яті та оперативної пам'яті важливо враховувати такі фактори, як розмір кешу, асоціативність кешу, таймінги пам'яті та архітектуру вашої системи. Крім того, рекомендується запустити кілька тестів різними методами, щоб отримати повне уявлення про продуктивність кеш-пам'яті та оперативної пам'яті в різних сценаріях.

* 1. **Огляд існуючих розробок у сфері тестування пам’ті**

Аналіз відомих програм відновлення втрачених даних може бути здійснений з різних поглядів, включаючи глибин досліду, надійність, інтерфейс користувача та ціну. Ось огляд кількох популярних програм для тестування пам’ті:

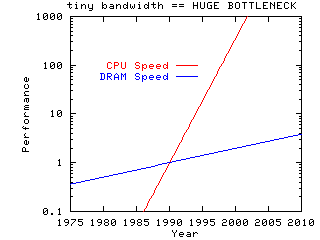
1. **Memtest86 -** це широко використовуваний інструмент діагностики пам'яті, призначений для тестування і аналізу стану і стабільності оперативної пам'яті комп'ютерної системи (ОЗП). Він зазвичай використовується для виявлення потенційних проблем з модулями пам'яті, таких як несправні мікросхеми оперативної пам'яті,неправильні таймінги пам'яті або проблеми сумісності.



*Рис. 1. 1. Вікно "Memtest86"*

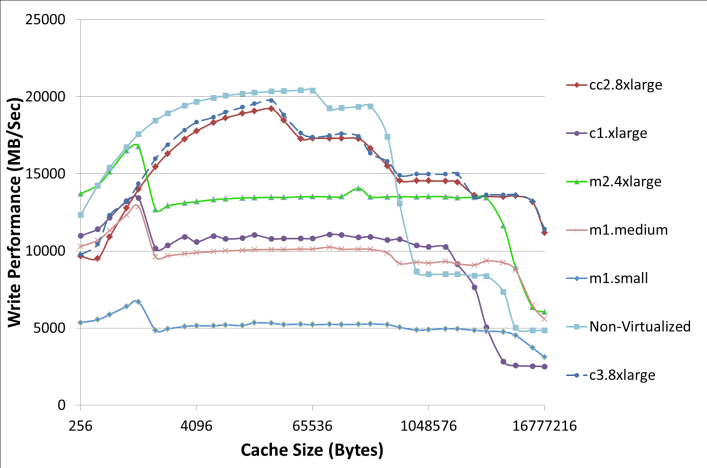
1. STREAM benchmark, LMBench, CacheBench, CPU-Z, HWiNFO, Intel VTune Amplifier

**The STREAM benchmark-** це широко використовуваний інструмент бенчмаркінгу, призначений для вимірювання стабільної пропускної здатності пам'яті комп'ютерної системи. Він зосереджений на оцінці продуктивності підсистеми пам'яті шляхом вимірювання швидкості, з якою дані можуть бути передані в основну пам'ять (ОЗП) і з неї.



*Рис. 1. 2. "*STREAM benchmark *"*

1. **CacheBench -** це інструмент бенчмаркінгу, призначений для оцінки продуктивності та характеристик кешу. Він зосереджується на оцінці частоти звернень до кешу, затримки кешу та пропускної здатності кешу, щоб отримати уявлення про ефективність та результативність ієрархії кешу.



*Рис. 1. 3. "*CacheBench*"*

1. **CPU-Z -** це популярний інструмент для моніторингу системної інформації та моніторингу комп'ютерів на базі Windows. Він надає детальну інформацію про CPU (процесор), материнську плату, оперативну пам'ять та інші компоненти системи. CPU-Z розроблений компанією CPUID, відомою своїми програмними утилітами, пов'язаними з апаратним забезпеченням.

Зображення, що містить текст, знімок екрана, монітор, число

Автоматично згенерований опис

*Рис. 1. 4. "*CPU-Z*"*

Ці програми є не численними на ринку. При виборі програми важливо враховувати конкретні потреби та характеристики пам’яті, які потрібно протестувати.

**Аналіз завдання та вибір підходів до проектування утиліти**

* 1. **Аналіз функцій та вибір структури програмного рішення**
     1. **Уточнення функціоналу :**
* **Параметри тестування** : розроблена програма має 4 опції для тестування здатність пам'яті отримувати дані, здатність пам'яті зберігати дані, ефективність пам'яті при копіюванні даних з одного місця в інше в межах пам'яті, час, необхідний пам'яті, щоб відповісти на запит на читання або запис.
* **Підтримка трьох рівнів кешу системи та оперативної пам’яті**: програма робить вимірювання швидкості для трьох рівнів кешу системи L1-Кеш, L2-Кеш та L3-Кеш. Для кожного з рівнів наводяться дані швидкості по чотирьох показниках, які наведені у попередньому заголовку. Також за аналогічними показниками тестується і оперативна пам’ять.
* **Збереження попередніх результатів** : усі виконувані системою тестування зберігаються в системі, та відображаються в окремій вкладці. Це зроблено для відстежування характеристик системи у довгостроковий термін та для виявлення потенційих проблем при різкій зміні результатів тестувань.
* **Функції моніторингу системної інформації**: програма надає інформацію про системні характеристики комп’тера про процесор, материнську плату, оперативну пам'ять, відображаються такі показники як назва процесора, тип, об’єм пам’яті системи та інше.  
  + 1. **Вибір структури на рівні модулів**

Вибір структури на рівні модулів є ключовим етапом в розробці утиліти,

оскільки від цього залежить організація функціональних компонентів та взаємодія між ними. Правильно обрана структура модулів дозволяє досягти високої модульності, зрозумілості коду, полегшує розширення та підтримку системи.

При виборі структури модулів для мого курсового проекту, я розглянув різні

аспекти утиліти і врахував їхню функціональну залежність та взаємодію. З метою забезпечення чіткого поділу відповідальностей та зменшення зв'язків між модулями, я обрав наступну структуру на рівні модулів яка зображена на рис.2. 1.

**Зображення, що містить текст, знімок екрана, Шрифт, Прямокутник

Автоматично згенерований опис**

*Рис. 2. 1 Структура програмної реалізації утиліти*

Опис кожного з модулів :

* **Модуль тестування пам’яті** відповідає за безпосередній процес тестування пам’яті за 4 параметрами.
* **Модуль перегляду даних про систему** відповідає за відображення системних даних.
* **Модуль налаштувань** відповідає за налаштування мови програми, важливих рекомендацій.
* **Модуль перегляду історії тестів** відповідає за перегляд інформації про виконані тести в даній програмі, посортовані за датою виконання.
* **Модуль системного трею** відповідає за інтеграцію утиліти і системного трею. Він забезпечує функціонал згортання, розгортання і закриття утиліти.

* 1. **Вибір архітектурного рішення**
     1. **Архітектура : MVVM(Model-View-ViewModel)**

**MVVM** - шаблон проєктування, що застосовується під час проєктування архітектури застосунків (додатків). MVVM орієнтований на такі сучасні платформи розробки, як Windows Presentation Foundation та Silverlight від компанії Microsoft.

MVVM полегшує відокремлення розробки графічного інтерфейсу від розробки бізнес логіки (бек-енд логіки), відомої як модель (можна також сказати, що це відокремлення представлення від моделі).

Модель представлення є частиною, яка відповідає за перетворення даних для їх подальшої підтримки і використання. З цієї точки зору, модель представлення більше схожа на модель, ніж на представлення і оброблює більшість, якщо не всю, логіку відображення даних. Модель представлення може також реалізовувати патерн медіатор, організовуючи доступ до бек-енд логіки навколо множини правил використання, які підтримуються представленням.

**Зображення, що містить текст, знімок екрана, Шрифт, Графіка

Автоматично згенерований опис**

*Рис. 2. 2 MVVM діаграма*

Переваги шаблону проєктування *MVVM :*

* Розділення відповідальності: MVVM дозволяє чітко розділити роль моделі даних, представлення користувача (вигляду) і логіки представлення (ViewModel). Це полегшує розробку, тестування і підтримку коду.
* Підтримка повторного використання: Завдяки розділенню моделі, вигляду і логіки представлення, можна повторно використовувати і компонувати їх у різних частинах програми. Це сприяє швидкому розвитку і зменшенню дублювання коду.
* Тестованість: MVVM полегшує тестування, оскільки логіка бізнес-логіки знаходиться в ViewModel, яка може бути легко протестована без залежностей від вигляду. Це дозволяє проводити модульні та автоматичні тести для впевненості в якості коду.
* Зручність для масштабування: MVVM надає гнучкість для масштабування програми. Можна додавати нові вигляди, моделі або логіку представлення без значного впливу на існуючий код.
  + 1. **Об’єктно – орієнтований підхід**

Використання об'єктно-орієнтованого програмування (ООП) для розробки програмного забезпечення має багато переваг і наступні аспекти:

* **Модульність**: ООП дозволяє розбити програму на окремі модулі або класи, які взаємодіють між собою через інтерфейси. Це полегшує розробку, тестування і підтримку коду, оскільки зміни в одному модулі мають менший вплив на інші частини програми.
* **Поновлюваність**: ООП підтримує принципи спадкування (inheritance) і поліморфізму, що дозволяє використовувати існуючий код для створення нових класів. Це сприяє повторному використанню коду, зменшенню дублювання і забезпечує більшу поновлюваність.
* **Спрощення складних задач**: ООП дозволяє моделювати реальні об'єкти і концепції за допомогою класів і об'єктів. Це полегшує розв'язання складних задач шляхом розбиття їх на менші, логічні складові.
* **Збільшена розширюваність**: ООП підтримує принципи розширення (extension) і абстракції (abstraction), що дозволяє додавати нові функціональності до існуючих класів без зміни їх внутрішньої реалізації. Це дозволяє легко розширювати програму і додає гнучкість для майбутнього розвитку.
* **Забезпечення безпеки**: ООП дозволяє використовувати інкапсуляцію (encapsulation), що означає, що дані та методи, які з ними пов'язані, знаходяться в межах класу і недоступні для зовнішніх об'єктів. Це забезпечує безпеку даних і унеможливлює неправильне використання або зміну даних.
* **Покращена організація командної розробки**: ООП сприяє розділенню ролей та відповідальностей між членами команди. Кожен може працювати над окремими класами або модулями, що полегшує співпрацю і розвиток проекту.
* **Підтримка спадкування і поліморфізму**: ООП дозволяє створювати ієрархії класів, де дочірні класи можуть успадковувати властивості та методи від батьківських класів. Це дозволяє використовувати поліморфізм для заміщення об'єктів різними класами, що полегшує роботу зі змінними типами даних і дозволяє створювати більш гнучкі програми.

Об'єктно-орієнтоване програмування є потужним і ефективним підходом для розробки програмного забезпечення, особливо великих проектів. Воно сприяє збільшенню повторного використання коду, зменшенню складності, покращенню організації та розширюваності програми.

* 1. **Вибір засобів для реалізації**
     1. **Вибір мови програмування**

Для написання програми я обрав мову програмування С# використання Windows Presentation Foundation для реалізації графічного інтерфейсу.

* + 1. **Вибір середовища**

Середовищем для написання програми є Visual Studio 2022.

* + - 1. **Вибір «двигуна» для відновлення даних**

**Windows File Recovery** - це утиліта командного рядка від Microsoft для відновлення видалених файлів. Вона є вільно доступною для Windows 10 версії 2004 (травневе оновлення 2020) і новіших версій з Microsoft Store.

1. **Програмна реалізація утиліти**
   1. **Розробка алгоритму роботи утиліти**

**Зображення, що містить текст, знімок екрана, чорно-білий, графічний дизайн

Автоматично згенерований опис**

*Рис. 3. 1 Схема алгоритму роботи програми*

**Зображення, що містить текст, знімок екрана, чорно-білий, графічний дизайн

Автоматично згенерований опис***Рис. 3. 2 Схема алгоритму роботи програми*

*Зображення, що містить текст, знімок екрана, місяць

Автоматично згенерований опис*

*Рис. 3. 3 Схема алгоритму роботи програми*

Зображення, що містить текст, знімок екрана, дизайн

Автоматично згенерований опис

*Рис. 3. 4 Схема алгоритму роботи програми*

* 1. **Розробка діаграми класів та опис методів**

Зображення, що містить текст, знімок екрана, Шрифт, програмне забезпечення

Автоматично згенерований опис

*Рис. 3. 5 Діаграма класів*

* + 1. **Опис методів для Functions.cs :**

**Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований опис**

*Рис. 3. 6 Вміст класу* Functions.cs

Метод RunScript є статичним і використовується для виконання скрипту або пакетного файлу (.bat) у віконному процесі. Основні дії, які виконуються цим методом, наступні:

1. Створення нового процесу за допомогою класу Process.
2. Встановлення параметрів для запуску процесу, використовуючи об'єкт ProcessStartInfo. Зокрема, вказується шлях до файлу .bat у властивості FileName, встановлюються параметри відображення вікна (CreateNoWindow = true) та виконання процесу без використання оболонки (UseShellExecute = false). Також встановлюється перенаправлення потоків виводу та помилок (RedirectStandardOutput = false, RedirectStandardError = true).
3. Прив'язка об'єкта ProcessStartInfo до об'єкта Process.
4. Запуск процесу за допомогою методу Start().
5. Очікування завершення роботи процесу за допомогою методу WaitForExit(), що блокує виконання до тих пір, поки процес не завершиться.
6. В блоках catch та finally виконується обробка помилок та закриття процесу для звільнення ресурсів. При виникненні помилки виводиться повідомлення про помилку до консолі.

private static void RunScript(string file\_name)

{

Process process = new Process();

try

{

// Set the process start information

ProcessStartInfo startInfo = new ProcessStartInfo();

startInfo.FileName = file\_name + ".bat"; // The path to your .bat file

startInfo.CreateNoWindow = true; // Do not create a window

startInfo.UseShellExecute = false; // Do not use the shell to execute

startInfo.RedirectStandardOutput = false; // Redirect the output stream

startInfo.RedirectStandardError = true; // Redirect the error stream

// Assign the start information to the process

process.StartInfo = startInfo;

// Start the process

process.Start();

// Wait for the process to exit

process.WaitForExit();

}

catch (Exception ex)

{

Console.WriteLine("An error occurred: " + ex.Message);

}

finally

{

// Close the process to release resources

process.Close();

process.Dispose();

}

}

Метод **Read\_Data** є статичним і використовується для зчитування даних з текстового файлу. Метод використовується для зчитування виміряних даних з текстового файла. У залежності від значення параметра **avg\_or\_max** обирається відповідний шаблон регулярного виразу (**pattern**). Якщо **avg\_or\_max** дорівнює **true**, використовується шаблон **"average=(.\*?),"**, якщо **false**, то **"max=(.\*?),"**

private static string Read\_Data(string file\_name, bool avg\_or\_max)

{

string speed = "";

try

{

StreamReader reader = new StreamReader("reports\\report\_" + file\_name + ".txt");

string content = reader.ReadToEnd();

reader.Close();

string pattern = "";

if(avg\_or\_max == true)

{

pattern = @"average=(.\*?),"; // Regular expression pattern

}

else

{

pattern = @"max=(.\*?),"; // Regular expression pattern

}

MatchCollection matches = Regex.Matches(content, pattern);

if (matches.Count > 0)

{

foreach (Match match in matches)

{

speed = match.Groups[1].Value.Trim();

}

}

else

{

}

}

catch (Exception ex)

{

Console.WriteLine("Error: " + ex.Message);

}

return speed;

}

Метод **Benchmark\_Info** є статичним і використовується для виконання бенчмарку та отримання інформації з результатів.

public static string Benckmark\_Info(string file\_name, bool avg\_or\_max)

{

string data = "";

RunScript(file\_name);

data = Read\_Data(file\_name, avg\_or\_max);

return data;

}

Метод **GetComponent** є статичним і використовується для отримання компонентів системи на основі вказаного класу та синтаксису запиту. Основні дії, які виконуються цим методом, наступні:

1. Створення об'єкту **StringBuilder** з назвою **outputBuilder**, який буде використовуватися для збереження результатів запиту.
2. В блоках **try-catch** виконується пошук об'єктів керування (**ManagementObject**) за допомогою **ManagementObjectSearcher**. Запит виконується для класу **hwclass** в просторі імен "root\CIMV2".
3. У циклі **foreach** проходиться крізь кожен об'єкт керування, отриманий з **ManagementObjectSearcher**, і витягується значення, вказане в параметрі **syntax**, з використанням індексатора **mj[syntax]**.
4. Отримане значення конвертується в рядок (**string**) і додається до **outputBuilder** за допомогою метода **AppendLine**, який додає рядок та символ нового рядка.

public static string GetComponent(string hwclass, string syntax)

{

StringBuilder outputBuilder = new StringBuilder();

try

{

ManagementObjectSearcher mos = new ManagementObjectSearcher("root\\CIMV2", "SELECT \* FROM " + hwclass);

foreach (ManagementObject mj in mos.Get())

{

string output = Convert.ToString(mj[syntax]);

outputBuilder.AppendLine(output);

}

}

catch (Exception ex)

{

outputBuilder.AppendLine("An error occurred: " + ex.Message);

}

return outputBuilder.ToString();

}

Метод **SplitString** є статичним і використовується для розділення рядка **data** на підрядки та отримання певних елементів з цих підрядків.

public static string SplitString(string data, int num)

{

string[] parts = data.Split('\n');

string result = "";

if(num == 1)

{

result = parts[0].Trim() + " , " + parts[1].Trim();

}

if(num == 2)

{

result = parts[2].Trim();

}

if (num == 3)

{

result = parts[3].Trim();

}

return result;

}

Метод **Save\_Screenshot** використовується для збереження знімка екрану елемента **FrameworkElement** у файл з вказаним шляхом **path**. Основні дії, які виконуються цим методом, наступні:

1. Створення об'єкту **RenderTargetBitmap** з розмірами, відповідними розмірами **FrameworkElement**, а також з заданою роздільною здатністю 96 dpi.
2. Виклик методу **Render** для **RenderTargetBitmap**, який рендерить **FrameworkElement**, використовуючи **bmp.Render(element)**.
3. Створення об'єкту **PngBitmapEncoder**, який використовується для кодування **RenderTargetBitmap** у формат PNG.
4. Додавання створеного кадру **BitmapFrame** до **PngBitmapEncoder** за допомогою **encoder.Frames.Add(BitmapFrame.Create(bmp))**.

public static void Save\_Screenchot(FrameworkElement element, string path)

{

if (string.IsNullOrEmpty(path))

{

// The string is either null or empty

path = Directory.GetParent(Environment.CurrentDirectory).Parent.FullName;

}

path += "\\image.png";

RenderTargetBitmap bmp = new RenderTargetBitmap((int)element.ActualWidth, (int)element.ActualHeight, 96, 96, PixelFormats.Pbgra32);

bmp.Render(element);

PngBitmapEncoder encoder = new PngBitmapEncoder();

encoder.Frames.Add(BitmapFrame.Create(bmp));

FileStream fs = new FileStream(path, FileMode.Create);

encoder.Save(fs);

fs.Close();

}

* + 1. **Опис методів для MainWindow.cs :**

Зображення, що містить текст, знімок екрана, Шрифт, Мобільний пристрій

Автоматично згенерований опис

*Рис. 3. 7 Вміст класу* MainWindow.cs

Метод **Close\_Btn\_Click** є обробником події натискання кнопки **Close\_Btn**. Цей метод викликає функцію **Close()**, яка закриває поточне вікно або додаток. У результаті, коли користувач натисне кнопку "Close\_Btn", викликана функція **Close()** закриє вікно або додаток, залежно від контексту, у якому використовується цей метод.

private void Close\_Btn\_Click(object sender, RoutedEventArgs e)

{

Close();

}

Метод **Hide\_Btn\_Click** є обробником події натискання кнопки **Hide\_Btn**. У цьому методі встановлюється властивість **WindowState** поточного вікна на значення **WindowState.Minimized**. Це призводить до мінімізації вікна до панелі завдань або рядка програм, що знаходиться внизу екрана. Коли користувач натисне кнопку "Hide\_Btn", викликана функція змінить стан вікна на мінімізований, зменшуючи його візуальне відображення на екрані.

private void Hide\_Btn\_Click(object sender, RoutedEventArgs e)

{

WindowState = WindowState.Minimized;

}

Метод **MovingWin** є обробником події перетягування вікна за допомогою миші. У цьому методі перевіряється, чи натиснута ліва кнопка миші (**e.LeftButton == MouseButtonState.Pressed**). Якщо ця умова виконується, викликається функція **DragMove()**, яка дозволяє пересувати вікно, утримуючи його за заголовок або іншу відповідну область.

private void MovingWin(object sender, MouseButtonEventArgs e)

{

if(e.LeftButton == MouseButtonState.Pressed)

{

DragMove();

}

}

Метод **NotifyIcon\_MouseClick** є обробником події кліку миші на сповіщенні (піктограмі) в системному лотку. У цьому методі перевіряється, чи була натиснута ліва кнопка миші (**e.Button == Forms.MouseButtons.Left**).

Якщо ця умова виконується, виконуються наступні дії:

1. **Show()**: Викликається функція **Show()**, яка відображає вікно програми.
2. **WindowState** для поточного вікна на **WindowState.Normal**. Це відновлює вікно до його нормального розміру та положення, якщо воно було мінімізоване або знаходилося в іншому стані.

private void NotifyIcon\_MouseClick(object sender, Forms.MouseEventArgs e)

{

if (e.Button == Forms.MouseButtons.Left)

{

Show(); // Show the application window

WindowState = WindowState.Normal; // Restore the window if it was minimized

}

}

Метод **OnExitClicked** є обробником події, яка відбувається при натисканні кнопки або елемента для виходу з програми. У цьому методі виконуються наступні дії:

1. **this.Close()**: Викликається функція **Close()** для поточного вікна або форми, що призводить до закриття вікна програми.
2. **notifyIcon.Dispose()**: Викликається функція **Dispose()** для **notifyIcon**, яка звільняє ресурси, пов'язані з використанням сповіщення (піктограми) в системному лотку.

private void OnExitClicked(object sender, EventArgs e)

{

this.Close();

notifyIcon.Dispose();

}

Метод **OnMaximizeClicked** є обробником події, яка відбувається при натисканні кнопки або елемента для максимізації вікна програми. У цьому методі виконуються наступні дії:

1. **Show()**: Викликається функція **Show()**, яка відображає вікно програми. Це дозволяє показати вікно користувачу, якщо воно було приховане або згорнуте.
2. **WindowState = WindowState.Normal**: Встановлюється значення **WindowState** для поточного вікна на **WindowState.Normal**. Це відновлює вікно до його нормального розміру та положення, якщо воно було згорнуте або знаходилося в іншому стані.

private void OnMaximizeClicked(object sender, EventArgs e)

{

Show(); // Show the application window

WindowState = WindowState.Normal; // Restore the window if it was minimized

}

Метод **OnMinimizeClicked** є обробником події, яка відбувається при натисканні кнопки або елемента для згортання вікна програми. У цьому методі встановлюється значення **WindowState** для поточного вікна на **WindowState.Minimized**, що призводить до мінімізації вікна до панелі завдань або рядка програм, що знаходиться внизу екрана.

private void OnMinimizeClicked(object sender, EventArgs e)

{

WindowState = WindowState.Minimized;

}

* + 1. **Опис методів для HomeView.cs :**

Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований опис

*Рис. 3. 8 Вміст класу* HomeView.cs

Метод **InitializeTextBoxes** використовується для ініціалізації текстових полів у графічному інтерфейсі користувача (UI). Cтворення рядка **text**, який містить об'єднання всіх значень текстових полів, розділених символом нового рядка.

private void InitializeTextBoxes()

{

CPU\_Name\_tb.Text = (Functions.GetComponent("Win32\_Processor", "Name")).TrimEnd();

CPU\_Type\_tb.Text = (Functions.GetComponent("Win32\_Processor", "Caption")).TrimEnd();

CPU\_Revision\_tb.Text = (Functions.GetComponent("Win32\_Processor", "Revision")).TrimEnd();

CPU\_Cores\_tb.Text = (Functions.GetComponent("Win32\_Processor", "NumberOfCores")).TrimEnd();

CPU\_LOgicalProc\_tb.Text = (Functions.GetComponent("Win32\_Processor", "NumberOfLogicalProcessors")).TrimEnd();

CPU\_ClockSpeed\_tb.Text = (Functions.GetComponent("Win32\_Processor", "MaxClockSpeed")).TrimEnd() + " MHZ";

Motherboard\_tb.Text = (Functions.GetComponent("Win32\_BaseBoard", "SerialNumber")).TrimEnd();

L1\_Size\_tb.Text = Functions.SplitString(((Functions.GetComponent("Win32\_CacheMemory", "MaxCacheSize")).TrimEnd()),1) + " Kb";

L2\_Size\_tb.Text = Functions.SplitString(((Functions.GetComponent("Win32\_CacheMemory", "MaxCacheSize")).TrimEnd()), 2) + " Kb";

L3\_Size\_tb.Text = Functions.SplitString(((Functions.GetComponent("Win32\_CacheMemory", "MaxCacheSize")).TrimEnd()), 3) + " Kb";

string dram = (Functions.GetComponent("Win32\_ComputerSystem", "TotalPhysicalMemory")).TrimEnd();

float megabytes = float.Parse(dram);

megabytes = megabytes / 1073741824;

DRAM\_Size\_tb.Text = megabytes.ToString("0.00") + " Gb";

text = CPU\_Name\_tb.Text + "\n" + CPU\_Type\_tb.Text + "\n" + CPU\_Revision\_tb.Text + "\n" + CPU\_Cores\_tb.Text + "\n" + CPU\_LOgicalProc\_tb.Text + "\n" + CPU\_ClockSpeed\_tb.Text + "\n" + Motherboard\_tb.Text + "\n" + L1\_Size\_tb.Text + "\n" + L2\_Size\_tb.Text + "\n" + L3\_Size\_tb.Text + "\n" + DRAM\_Size\_tb.Text;

}

Метод **Start\_Btn\_Click** викликається при натисканні кнопки "Start" у графічному інтерфейсі користувача. Основна дія, яка виконується цим методом, полягає у записі тексту **text** у файл "Last\_metrics.txt" у папці "History".

private void Start\_Btn\_Click(object sender, RoutedEventArgs e)

{

InitializeTextBoxes();

File.WriteAllText("History\\Last\_metrics.txt", text);

}

Метод **Renew\_last\_metrics** використовується для оновлення текстових полів у графічному інтерфейсі користувача з даними, що зберігаються в файлі "Last\_metrics.txt".

private void Renew\_last\_metrics()

{

string path = "History\\Last\_metrics.txt";

if (new FileInfo(path).Length != 0)

{

string fileContent = File.ReadAllText(path);

string[] parts = fileContent.Split('\n');

CPU\_Name\_tb.Text = parts[0].Trim();

CPU\_Type\_tb.Text = parts[1].Trim();

CPU\_Revision\_tb.Text = parts[2].Trim();

CPU\_Cores\_tb.Text = parts[3].Trim();

CPU\_LOgicalProc\_tb.Text = parts[4].Trim();

CPU\_ClockSpeed\_tb.Text = parts[5].Trim();

Motherboard\_tb.Text = parts[6].Trim();

L1\_Size\_tb.Text = parts[7].Trim();

L2\_Size\_tb.Text = parts[8].Trim();

L3\_Size\_tb.Text = parts[9].Trim();

DRAM\_Size\_tb.Text = parts[10].Trim();

}

}

Метод **Button\_Click** викликається при натисканні кнопки для вибору шляху файлу та збереження знимку екрана, функціонал:

1. Створюється екземпляр класу **FolderBrowserDialog** з простору імен **System.Windows.Forms**. Це вікно дозволяє користувачеві вибрати папку на комп'ютері.
2. Створюється змінна **DestinationDirectory** для збереження шляху до обраної папки.
3. Викликається метод **ShowDialog** для відображення вікна вибору папки. Результат вибору зберігається у змінну **result**.
4. Якщо користувач обрав папку та натиснув кнопку "OK", то присвоюється вибраний шлях до змінної **DestinationDirectory**.

private void Button\_Click(object sender, RoutedEventArgs e)

{

WinForms.FolderBrowserDialog dialog = new WinForms.FolderBrowserDialog();

string DestinationDirectory = "";

WinForms.DialogResult result = dialog.ShowDialog();

if (result == WinForms.DialogResult.OK)

{

DestinationDirectory = dialog.SelectedPath;

}

Functions.Save\_Screenchot(this, DestinationDirectory);

}

}

* + 1. **Опис методів для BenchmarkView.cs**

Зображення, що містить текст, знімок екрана, Мобільний пристрій, Шрифт

Автоматично згенерований опис

*Рис. 3. 9 Вміст класу* BenchmarkView.cs

Метод **Renew\_last\_metrics** використовується для оновлення текстових полів у графічному інтерфейсі користувача з даними, що зберігаються в файлі "Last\_ benchmark.txt".

private void Renew\_last\_benchmark()

{

string path = "History\\Last\_benchmark.txt";

if(new FileInfo(path).Length != 0)

{

string fileContent = File.ReadAllText(path);

string[] parts = fileContent.Split('\n');

Read\_L1.Text = parts[0].Trim();

Read\_L2.Text = parts[1].Trim();

Read\_L3.Text = parts[2].Trim();

Read\_DRAM.Text = parts[3].Trim();

Write\_L1.Text = parts[4].Trim();

Write\_L2.Text = parts[5].Trim();

Write\_L3.Text = parts[6].Trim();

Write\_DRAM.Text = parts[7].Trim();

Copy\_L1.Text = parts[8].Trim();

Copy\_L2.Text = parts[9].Trim();

Copy\_L3.Text = parts[10].Trim();

Copy\_DRAM.Text = parts[11].Trim();

Latency\_L1.Text = parts[12].Trim();

Latency\_L2.Text = parts[13].Trim();

Latency\_L3.Text = parts[14].Trim();

Latency\_DRAM.Text = parts[15].Trim();

}

}

Метод **InitializeTextBoxes з**аписує результати бенчмарку для читання, запису, копіювання та затримки рівня L1, L2, L3 та DRAM відповідно в текстові поля. Формує рядок **text**, який містить всі результати бенчмарків у відповідному форматі.

private void InitializeTextBoxes()

{

Read\_L1.Text = Functions.Benckmark\_Info("read\_L1", false) + " MGb/s";

Read\_L2.Text = Functions.Benckmark\_Info("read\_L2", false) + " MGb/s";

Read\_L3.Text = Functions.Benckmark\_Info("read\_L3", false) + " MGb/s";

Read\_DRAM.Text = Functions.Benckmark\_Info("read\_DRAM", false) + " MGb/s";

Write\_L1.Text = Functions.Benckmark\_Info("write\_L1", false) + " MGb/s";

Write\_L2.Text = Functions.Benckmark\_Info("write\_L2", false) + " MGb/s";

Write\_L3.Text = Functions.Benckmark\_Info("write\_L3", false) + " MGb/s";

Write\_DRAM.Text = Functions.Benckmark\_Info("write\_DRAM", false) + " MGb/s";

Copy\_L1.Text = Functions.Benckmark\_Info("copy\_L1", false) + " MGb/s";

Copy\_L2.Text = Functions.Benckmark\_Info("copy\_L2", false) + " MGb/s";

Copy\_L3.Text = Functions.Benckmark\_Info("copy\_L3", false) + " MGb/s";

Copy\_DRAM.Text = Functions.Benckmark\_Info("copy\_DRAM", false) + " MGb/s";

Latency\_L1.Text = Functions.Benckmark\_Info("latency\_L1", true) + " ns";

Latency\_L2.Text = Functions.Benckmark\_Info("latency\_L2", true) + " ns";

Latency\_L3.Text = Functions.Benckmark\_Info("latency\_L3", true) + " ns";

Latency\_DRAM.Text = Functions.Benckmark\_Info("latency\_DRAM", true) + " ns";

text = Read\_L1.Text + "\n" + Read\_L2.Text + "\n" + Read\_L3.Text + "\n" + Read\_DRAM.Text + "\n" + Write\_L1.Text + "\n" + Write\_L2.Text + "\n" + Write\_L3.Text + "\n" + Write\_DRAM.Text + "\n" + Copy\_L1.Text + "\n" + Copy\_L2.Text + "\n" + Copy\_L3.Text + "\n" + Copy\_DRAM.Text + "\n" + Latency\_L1.Text + "\n" + Latency\_L2.Text + "\n" + Latency\_L3.Text + "\n" + Latency\_DRAM.Text;

}

Метод **Save\_History** додає останні результати бенчмарків до історії, збереженої в файлі "History\_benchmark.txt".

private void Save\_History()

{

string path = "History\\History\_benchmark.txt";

string text = "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

text += "\t";

text += DateTime.Now.ToString("yyyy-MM-dd HH:mm");

text += "\n";

text += "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

text += "Read:\n";

text += "L1: \t\t" + Read\_L1.Text + "\n";

text += "L2: \t\t" + Read\_L2.Text + "\n";

text += "L3: \t\t" + Read\_L3.Text + "\n";

text += "Memory: \t" + Read\_DRAM.Text + "\n\n";

text += "Write:\n";

text += "L1: \t\t" + Read\_L1.Text + "\n";

text += "L2: \t\t" + Write\_L2.Text + "\n";

text += "L3: \t\t" + Write\_L3.Text + "\n";

text += "Memory: \t" + Write\_DRAM.Text + "\n\n";

text += "Copy:\n";

text += "L1: \t\t" + Copy\_L1.Text + "\n";

text += "L2: \t\t" + Copy\_L2.Text + "\n";

text += "L3: \t\t" + Copy\_L3.Text + "\n";

text += "Memory: \t" + Copy\_DRAM.Text + "\n\n";

text += "Latency:\n";

text += "L1: \t\t" + Latency\_L1.Text + "\n";

text += "L2: \t\t" + Latency\_L2.Text + "\n";

text += "L3: \t\t" + Latency\_L3.Text + "\n";

text += "Memory: \t" + Latency\_DRAM.Text + "\n";

text += "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n\n";

string existingContent = File.ReadAllText(path);

// Prepend the new text to the existing content

string updatedContent = text + Environment.NewLine + existingContent;

// Write the updated content back to the file

File.WriteAllText(path, updatedContent);

}

Метод **Start\_Btn\_Click** оновлює результати бенчмарків, зберігає їх в файл "Last\_benchmark.txt" та зберігає історію результатів бенчмарків.

private void Start\_Btn\_Click(object sender, RoutedEventArgs e)

{

InitializeTextBoxes();

//Functions.Write\_Last\_Result("Last\_benchmark.txt", text);

using (StreamWriter writer = new StreamWriter("History\\Last\_benchmark.txt"))

{

writer.WriteLine(text);

}

Save\_History();

}

Метод **Button\_Click** викликається при натисканні кнопки для вибору шляху файлу та збереження знимку екрана, функціонал:

1. Створюється екземпляр класу **FolderBrowserDialog** з простору імен **System.Windows.Forms**. Це вікно дозволяє користувачеві вибрати папку на комп'ютері.
2. Створюється змінна **DestinationDirectory** для збереження шляху до обраної папки.
3. Викликається метод **ShowDialog** для відображення вікна вибору папки. Результат вибору зберігається у змінну **result**.
4. Якщо користувач обрав папку та натиснув кнопку "OK", то присвоюється вибраний шлях до змінної **DestinationDirectory**.

private void Button\_Click(object sender, RoutedEventArgs e)

{

WinForms.FolderBrowserDialog dialog = new WinForms.FolderBrowserDialog();

string DestinationDirectory = "";

WinForms.DialogResult result = dialog.ShowDialog();

if (result == WinForms.DialogResult.OK)

{

DestinationDirectory = dialog.SelectedPath;

}

Functions.Save\_Screenchot(this, DestinationDirectory);

}

}

* + 1. **Опис методів для SettingsView.cs :**

Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований опис

*Рис. 3. 10 Вміст класу* SettingsView.cs

Метод **ChangeLanguage** використовується для зміни мови в програмі. Він виконує наступні дії:

1. Встановлює поточну культуру (**CurrentCulture**) та культуру користувацького інтерфейсу (**CurrentUICulture**) поточного потоку на заданий **LanguageCode**.
2. Очищує колекцію злитих словників ресурсів (**MergedDictionaries**) у поточному додатку (**Application.Current.Resources**).
3. Створює новий словник ресурсів (**ResourceDictionary**) із посиланням на файл словника мови (**Dictionary-{LanguageCode}.xaml**). Вказане посилання відносне та вказує на певний файл ресурсів для заданої мови.
4. Додає створений словник ресурсів до колекції злитих словників ресурсів поточного додатку.

public static void ChangeLanguage(string LanguageCode)

{

Thread.CurrentThread.CurrentCulture = new CultureInfo(LanguageCode);

Thread.CurrentThread.CurrentUICulture = new CultureInfo(LanguageCode);

Application.Current.Resources.MergedDictionaries.Clear();

ResourceDictionary resourceDictionary = new ResourceDictionary()

{

Source = new Uri($"/Localization/Dictionary-{LanguageCode}.xaml", UriKind.Relative)

};

Application.Current.Resources.MergedDictionaries.Add(resourceDictionary);

}

У методі **ComboBox\_SelectionChanged** відбувається обробка події зміни вибраного елемента в ComboBox. Основна дія, яка відбувається в цьому методі, полягає в виклику методу **ChangeLanguage(selectedTag)** для зміни мови програми на обрану.

За допомогою **selectedItem.Tag.ToString()** отримується значення атрибуту **Tag** вибраного елемента, яке відповідає коду мови.

Нарешті, метод **ChangeLanguage(selectedTag)** викликається для зміни мови програми на обрану.

private void ComboBox\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

ComboBox comboBox = (ComboBox)sender;

ComboBoxItem selectedItem = (ComboBoxItem)comboBox.SelectedItem;

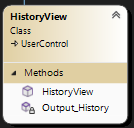
string selectedTag = selectedItem.Tag.ToString();

selectedLang = selectedTag;

ChangeLanguage(selectedTag);

}

**2.2.6  Опис методів для HistoryView.cs :**

****

*Рис. 3. 11 Вміст класу* HistoryView.cs

У методі **Output\_History** відбувається виведення змісту файлу з історією (**History\_benchmark.txt**) до елементу TextBox з назвою **History\_tb**.

private void Output\_History()

{

string path = "History\\History\_benchmark.txt";

if (new FileInfo(path).Length != 0)

{

string fileContent = File.ReadAllText(path);

History\_tb.Text = fileContent;

}

}

* 1. **Опис архітектури графічного інтерфейсу**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, Шрифт

Автоматично згенерований опис**

*Рис. 3. 8 Діаграма класів для графічного інтерфейсу*

Оскільки ця програма розроблялась за допомогою шаблону проєктування **MVVM(Model-View-ViewModel)**, то логіка взаємодії графічного інтерфейсу і логіка роботи методів роз’єднана в окремі елементи.

Графічний інтерфейс моєї програми можна поділити на 2 частини :

Основне вікно MainWindow.xaml :

Зображення, що містить знімок екрана, текст, програмне забезпечення, Мультимедійне програмне забезпечення

Автоматично згенерований опис

*Рис. 3. 9 Основне графічне вікно*

Додаткові панелі :

HomeView.xaml :

Зображення, що містить текст, знімок екрана, програмне забезпечення, Операційна система

Автоматично згенерований опис

*Рис. 3. 10 Додтакова панель* HomeView.*xaml*

BenchmarkView.xaml :

Зображення, що містить текст, знімок екрана, програмне забезпечення, дизайн

Автоматично згенерований опис

*Рис. 3. 11 Додтакова панель* BenchmarkView.*xaml*

HistoryView.xaml :

Зображення, що містить знімок екрана, текст, монітор, програмне забезпечення

Автоматично згенерований опис

*Рис. 3. 12 Додтакова панель* HistoryView.*xaml*

SettingsView.xaml :

Зображення, що містить текст, електроніка, знімок екрана, програмне забезпечення

Автоматично згенерований опис

*Рис. 3. 13 Додтакова панель* SettingsView.*xaml*

Моя програма написана за такою методикою, що на головному вікні, в залежності від опції, вибраної зліва, прикріпляється одна з чотирьох можливих панелей, наприклад :

Зображення, що містить текст, знімок екрана, програмне забезпечення, Мультимедійне програмне забезпечення

Автоматично згенерований опис

*Рис. 3. 14 Приклад накладання додаткової панелі на основне вікно*

Для коректної роботи програми з такою структурою, потрібно написати два класи, які допомагатимуть перемикати панелі на головному вікні і правильно їх відображати.

Клас ObservableObject :

internal class ObservableObject : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

protected void OnPropertyChanged([CallerMemberName] string name = null)

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));

}

}

Цей код визначає клас ObservableObject, який є внутрішнім (internal) і реалізує інтерфейс INotifyPropertyChanged з простору імен System.ComponentModel. ObservableObject надає базову функціональність для реалізації шаблону проектування "Observer" в програмах на платформі WPF, зокрема, для реалізації прив'язки даних і сповіщення про зміну властивостей об'єктів.

Метод OnPropertyChanged викликає подію PropertyChanged і сповіщає про зміну властивості. Цей метод приймає назву властивості (яку отримує автоматично завдяки [CallerMemberName] атрибуту) і генерує подію, передаючи назву зміненої властивості.

Цей код використовується як базовий клас для реалізації моделей даних або властивостей, за якими треба спостерігати. При зміні значення властивості, ви можете викликати метод OnPropertyChanged, щоб сповістити всіх слухачів (інші об'єкти, які будуть викликатись), що властивість була змінена. Це дозволяє забезпечити автоматичне оновлення і відображення змінених даних у візуалізації програми.

Клас RellayComand :

internal class RellayComand : ICommand

{

private Action<object> \_execute;

private Func<object, bool> \_canExecute;

public event EventHandler CanExecuteChanged

{

add { CommandManager.RequerySuggested += value; }

remove { CommandManager.RequerySuggested -= value; }

}

public RellayComand(Action<object> execute)

{

\_execute = execute;

}

public bool CanExecute(object parameter)

{

return \_canExecute == null || \_canExecute(parameter);

}

public void Execute(object parameter)

{

\_execute(parameter);

}

}

Цей код визначає клас RelayCommand, який реалізує інтерфейс ICommand з простору імен System.Windows.Input. RelayCommand дозволяє виконувати команди від користувача в інтерфейсі користувача.

Цей код використовується для створення команд, які можна пов'язати з елементами інтерфейсу користувача, такими як кнопки, для виконання певних операцій. Використовуючи цей RelayCommand, ви можете передати метод виконання команди при створенні екземпляра RelayCommand, і цей метод буде виконуватись при виконанні команди.

Безпосередньо для перемикання між додатковими панелями для кожої з них описана логіка у класі MainViewModel.cs :

internal class MainViewModel : ObservableObject

{

public RellayComand RecoveryViewCommand { get; set; }

public RellayComand SettingsViewCommand { get; set; }

public RellayComand AboutViewCommand { get; set; }

public RellayComand DiskInfoViewCommand { get; set; }

public RecoveryViewModel RecoveryVM { get; set; }

public SettingsViewModel SettingsVM { get; set; }

public AboutViewModel AboutVM { get; set; }

public DiskInfoViewModel DiskInfoVM { get; set; }

private object \_currentView;

public object CurrentView

{

get { return \_currentView; }

set

{

\_currentView = value;

OnPropertyChanged();

}

}

public MainViewModel()

{

RecoveryVM = new RecoveryViewModel();

SettingsVM = new SettingsViewModel();

AboutVM = new AboutViewModel();

DiskInfoVM = new DiskInfoViewModel();

CurrentView = RecoveryVM;

RecoveryViewCommand = new RellayComand(o =>

{

CurrentView = RecoveryVM;

});

SettingsViewCommand = new RellayComand(o =>

{

CurrentView = SettingsVM;

});

AboutViewCommand = new RellayComand(o =>

{

CurrentView = AboutVM;

});

DiskInfoViewCommand = new RellayComand(o =>

{

CurrentView = DiskInfoVM;

});

}

}

* 1. **Створення та опис програмного проекту**

Розроблена утиліта під назвою Data Recovery Tool є структурованим програмним рішеням, де реалізація графічного інтерфейсу, відновленням даних, роботою з файлами, локалізацією, відокремлені як окремі модулі та групуються відповідно до папок та класів.

Загалом проект містить :

* 13 класів.
* 7 xaml файлів з графічною розміткою.
* 2 xaml файли з словниками для локалізації.
* 2 папки з ресурсами(2 іконки, 1 зображення, 3 текстові файли).
* Додатковий проект для реалізації інсталятора.

Зображення, що містить текст, знімок екрана, програмне забезпечення, Мультимедійне програмне забезпечення

Автоматично згенерований опис Зображення, що містить текст, знімок екрана, програмне забезпечення, Операційна система

Автоматично згенерований опис

*Рис. 3. 15 та 3.16 Вміст проекту*

Проект містить такі папки :

1. Core – в ній містяться файли, потрібні для коректної роботи і відображення графічного інтерфейсу(описані в розділі 3.3).  
   
2. MVVM, що в свою чергу складається з :

2.1 View – містить 4 xaml файли з графічною розміткою і 4 класи з описом логіки роботи програми.   
Зображення, що містить текст, знімок екрана, Шрифт

Автоматично згенерований опис

2.2 ViewModel – містить 5 класів, з описом взаємодії елементів графічного інтерфейсу між собою.  
Зображення, що містить текст, знімок екрана, Шрифт, програмне забезпечення

Автоматично згенерований опис

1. Images – містить необхідні ресурси для роботи програми, а саме : словники для локалізацій, іконки для самої програми, для трею, і для внутрішнього дизайну.

Зображення, що містить текст, знімок екрана, Шрифт

Автоматично згенерований опис

1. Themes – папка в якій зберігається кастомний стиль для кнопок на сторінці MainViewView.  
   
2. Localization – папка, де зберігаються 2 мовні словники.

Зображення, що містить текст, Шрифт, знімок екрана, ряд

Автоматично згенерований опис

Опис всіх файлів і їхнього коду наведений вище у коді.

**4. Тестування програмного забезпечення та інструкції користувачеві**

**4.1 Тестування та опис інтерфейсу програми**

Для зручного користування програмою рорзоблено графічний інтерфейс за допомогою платфори WPF на мові С#.

Зображення, що містить текст, знімок екрана, програмне забезпечення, Мультимедійне програмне забезпечення

Автоматично згенерований опис

*Рис. 4.1 Основне вікно програми та Розділ Головна*

1. **Налаштування –** Вкладка Налаштування, там знаходиться інформація про проект та є кнопка для зміни мови.
2. **Головна –** Розділ програми, що відкритий відразу після запуску програми, тут можна подивитись характеристики системи, провести нове дослідження та зберегти.
3. **Тестування –** Розділ програми, в якому можна зробити, переглянути тестування пам’яті та зберегти.
4. **Історія -** Розділ програми, в якому можна побачити історію тестування пам’яті.
5. **Зберегти–** Кнопка за допомогою якої можна зберегти знимок екрана на пристрій.
6. **Почати тестування –** Кнопка для проведення тестування та оновлення даних на екрані.
7. **Вибір режиму –** Кнопка, завдякий якій можна мінімізувати вікно(згорнути).
8. **Вибір бажаних типів файлів –** Кнопка, завдякий якій можна закрити програму.

**При першому запуску програми, користувач отримає пусте вікно з даними, оскільки тестування ще не проводились і файли зі збереженими даними минулих тестувань ще пусті.**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, Мультимедійне програмне забезпечення

Автоматично згенерований опис**

*Рис. 4.2 Пусте вікно при першому запуску.*

Вікно буде пустим доки не проведуться перші тестування, далі при майбутніх запусках програма буде виводити останнє тестування, що проводилось, шоб оновити дані потрібно буде провести нове тестування.

При натисканні на кнопку Зберегти з’являється вікно в якому потрібно обрати шлях куди зберегти файл. Файл буде збережено у вказану дерикторію та з іменем Image.png.

Зображення, що містить текст, знімок екрана, програмне забезпечення, Комп’ютерна піктограма

Автоматично згенерований опис

*Рис. 4.3 Вікно для вибору шляху.*

Зображення, що містить текст, знімок екрана, програмне забезпечення, монітор

Автоматично згенерований опис

*Рис. 4.4 Збережений файл у вказаній папці.*

При завчасному закритті вікна з вибором шляху, фото збережеться у папку з проектом.

Зображення, що містить текст, знімок екрана, число, Шрифт

Автоматично згенерований опис

*Рис. 4.4 Збережений файл у папці з проектом.*

Зображення, що містить текст, знімок екрана, програмне забезпечення, Мультимедійне програмне забезпечення

Автоматично згенерований опис

*Рис. 4.5 Розділ Тестування.*

Розділ «Тестування» має аналогічні рішення як у розділі Головна, відмінність тільки у тестах, що проводяться.

*Зображення, що містить текст, знімок екрана, програмне забезпечення, Мультимедійне програмне забезпечення

Автоматично згенерований опис Рис. 4.7 Розділ Історія.*

Розділ «Історія» буде виводити усі результати досліджень, які проводились на даному пристрої, вони будуть відсортовані по даті(найновіші - найвище).

Якщо тестувань ще не було, то вікно буде пустим.

В четвертій вкладці під назвою “Налаштування”, в яку можна перейти при натиску на кнопку шестерні у лівому верхньому куті є можливість зміни мови та вивиедений текст про програму нижче :

Зображення, що містить текст, знімок екрана, програмне забезпечення, Веб-сайт

Автоматично згенерований опис

*Рис. 4.8 Розділ Налаштування.*

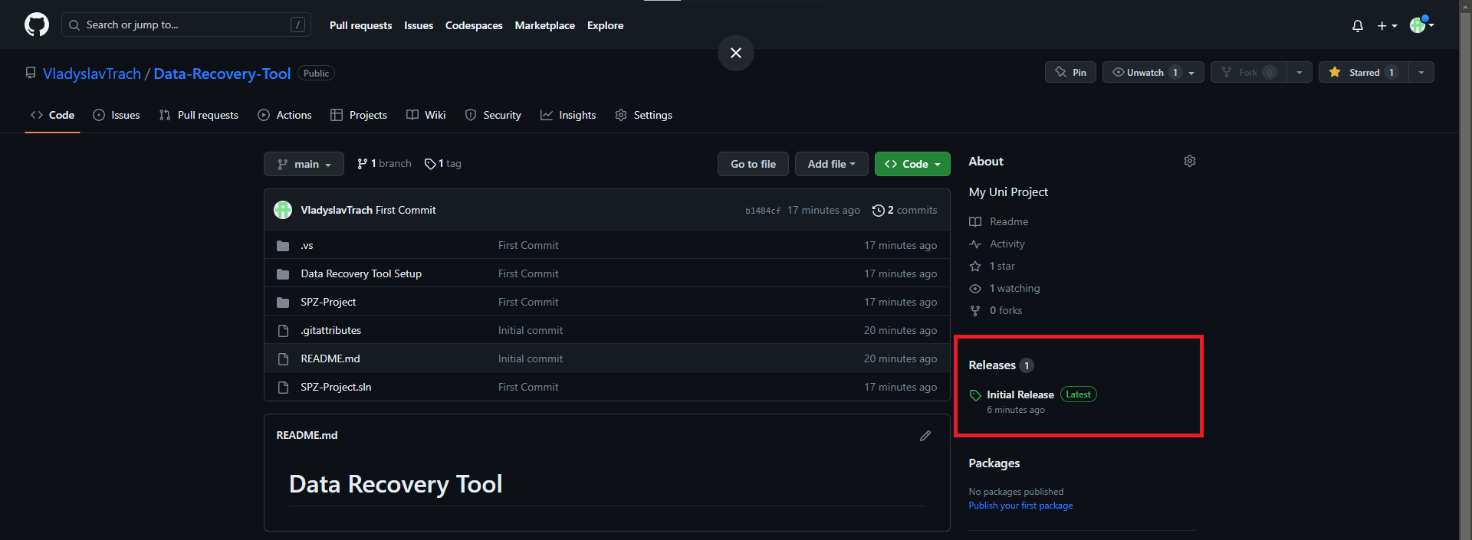
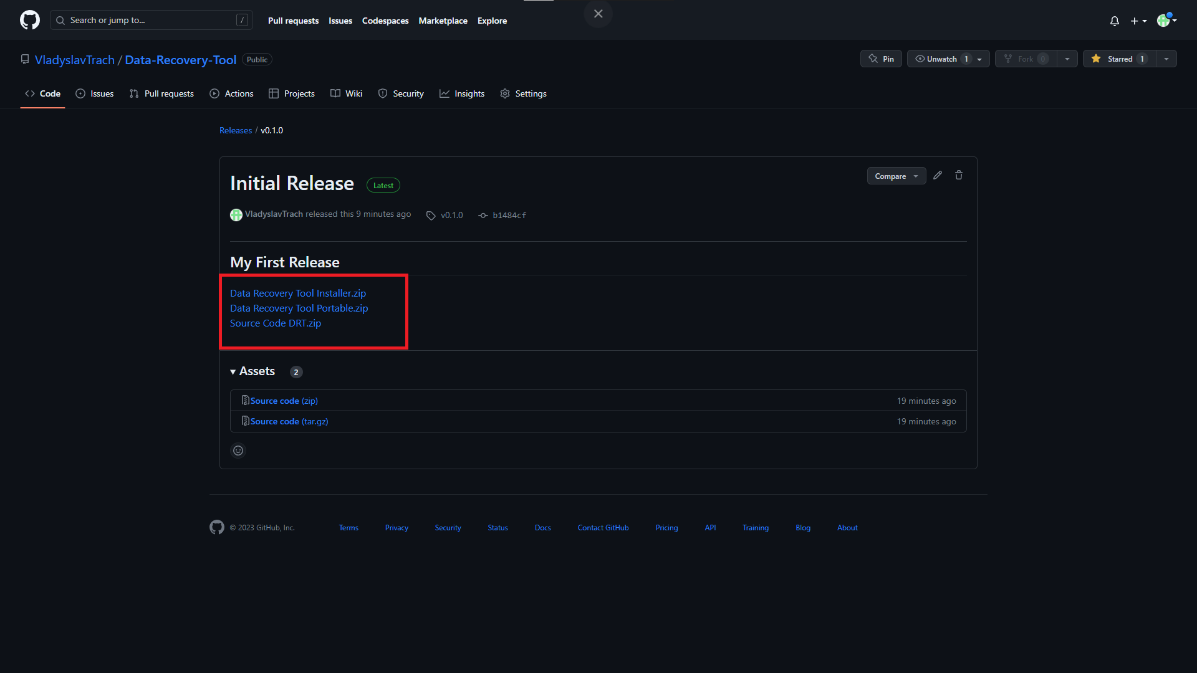
Зображення, що містить текст, знімок екрана, програмне забезпечення, Мультимедійне програмне забезпечення

Автоматично згенерований опис

*Рис. 4.9 Вигляд програми після зміни мови.*

**4.2 Інструкції для користувача**

Для завантаження додатку користувач має :

1. Перейти на гітхаб сторінку цього проєкту : <https://github.com/VladyslavTrach/Data-Recovery-Tool>
2. Натиснути на версію “Initial Release”
3. У відкритому вікні можна завантажити Інсталятор, Портативну версію і вихідний код :

**Висновок**

Під час виконання даного курсовго проєкту, я закріпив знання, які отримав під час вивчення курсу: “Системне програмне забезпечення” і які були використані на практиці при проектуванні та розробці утиліти тестування кешу та оперативної памяті комп’ютера в ОС Windows, яка отримала назву: “Cache and Memory Benchmark”. В результаті було виконано наступну роботу:

1. Проведено аналіз і огляд існуючих розробок і методів у галузі тестування пам’яті комп’ютера з ОС Windows. Описано різні підходи, програмні засоби та технології, що використовуються для тестування пам’яті на ПК.

2. Проектування програми включало розробку архітектури та вибір необхідних інструментів та бібліотек. Програма побудована на принципах Model-View-ViewModel Programming і об'єктно-орієнтованого підходу(ООП) з використанням платформи WPF для побудови графічного інтерфейсу, та .NET бібліотек для певного функціоналу.

3. Під час тестування була підтверджена працездатність системи. Програма ефективно виконує свої функції і забезпечує якісне відновлення втрачених даних з ПК.

Загалом, розробка цієї утиліти відновлення втрачених даних з ПК на ОС Windows виявилася успішною, демонструючи важливі аспекти Model-View-ViewModel Programming і об'єктно-орієнтованого підходу(ООП) та ефективного використання .NET бібліотек. Результати роботи відповідають поставленим цілям та відображають успішну реалізацію програми, яка може бути використана для відновлення втрачених даних зручним та ефективним способом. Звісно, даний додаток, ще потребує багато вдосконалень як і в плані функціональних можливостей, так і в продуктивності.

Отже, розробка утиліти відновлення втрачених даних дозволила набути практичних навичок у проектуванні програмних систем та поглибити розуміння процесу розробки програмного забезпечення.

**Список використаних джерел**

1. [The Model-View-ViewModel (MVVM) Documentation by Micrososft](https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm) .
2. [DriveInfo Class in System.IO namespace Documentation by Microsoft](https://learn.microsoft.com/ru-ru/dotnet/api/system.io.driveinfo?view=net-7.0).
3. [Visual Studio Installer Projects Extension and .NET 6.0](https://learn.microsoft.com/en-us/visualstudio/deployment/installer-projects-net-core?view=vs-2022).
4. [Recover lost files on Windows 10](https://support.microsoft.com/en-us/windows/recover-lost-files-on-windows-10-61f5b28a-f5b8-3cc2-0f8e-a63cb4e1d4c4).