НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
____комп'ютерних наук_____
(назва кафедри)

_____ __Голуб Б.Л._
(підпис) (ПІБ)

"___" _____ 20 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему

«Програмне забезпечення моніторингу інформаційних даних комп'ютерної системи»

Спеціальність 121 – «Інженерія програмного забезпечення»

Гарант освітньої програми
_____к.т.н., доцент_____ _____ ____Голуб Б.Л._____
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи
_____к.т.н., доцент_____ _____ ____Пархоменко І.І._____
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ _____Журавльов Микита Олександрович_____
(підпис) (ПІБ студента)

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ

Завідувач кафедри

___комп'ютерних наук___
___к.т.н., доцент___ _____ _Голуб Б.Л._
(науковий ступінь, вчене звання)(підпис) (ПІБ)
« 06 » _травня___ 2025р.

З А В Д А Н Н Я

на виконання бакалаврської кваліфікаційної роботи студенту

_____Журавльову Микиті Олександровицчу_____
(прізвище, ім'я, по батькові)

Спеціальність 121 – «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи Програмне забезпечення моніторингу інформаційних даних комп'ютерної системи_____
_____
_____

затверджена наказом ректора НУБіП України від «06» травня 2025р. № 758-С

Термін подання завершеної роботи на кафедру __2025.05.25_____
(рік, місяць, число)

Вихідні дані до бакалаврської кваліфікаційної роботи
_____
_____
_____

Перелік питань, які потрібно розробити:
Аналіз предметної області, огляд існуючих рішень, визначення вимог, моделювання системи, проєктування програмного забезпечення, логічна модель даних, діаграми класів і компонентів, вибір інструментів розробки, розробка програмного забезпечення, впровадження та експлуатація системи
Перелік графічних документів (за потреби)_____
_____
_____

Дата видачі завдання "_22_"_____жовтня_____2024_ р.

Керівник бакалаврської кваліфікаційної роботи _____ _____
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання _____ _____
(підпис) (прізвище та ініціали студента)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Системний адміністратор - Людина, відповідальна за керування та обслуговування комп'ютерних систем та мереж

.NET Framework - Фреймворк розробки програмного забезпечення для Windows.

WPF - Фреймворк графічного інтерфейсу користувача (GUI) для Windows.

API - Набір правил і специфікацій, які визначають, як програмний додаток може взаємодіяти з іншими програмними додатками.

UI - Графічні елементи програмного додатку, з якими взаємодіє користувач.

UX - Загальний досвід, який має користувач при використанні програмного додатку.

ОС – Операційна система

ВСТУП

Робота системних адміністраторів та їхніх помічників має надзвичайне значення для будь-якого підприємства, де наявні два або більше комп'ютерів. Це стає ще більш важливим у випадках, коли мова йде про сервери чи мережеві сховища.

Системні адміністратори відповідають за забезпечення справної роботи обладнання, його взаємодію в мережевій структурі, а також надання працівникам повного спектра необхідного програмного забезпечення для виконання службових обов'язків. Вони також займаються реінвентаризацією всього обладнання у разі зміни складу працівників компанії або придбання нового обладнання.

Важливо зазначити, що більшість робочих станцій працюють на операційних системах Windows 10 та 11, що визначає вибір інструментів розробки для оптимізації їхньої роботи.

Фахівці IT-відділів компаній стикаються з численними викликами, пов'язаними з використанням різноманітних окремих інструментів для виконання щоденних завдань. Ці завдання можуть варіюватися від інвентаризації обладнання до встановлення програмного забезпечення на робочі станції.

Хоча окремі інструменти для кожного з цих завдань є високоякісними та ефективними, їх одночасне використання часто призводить до розпорошення уваги, неефективного використання часу та ускладнення робочих процесів. Наприклад, зберігання необхідних файлів і програм на флешках, їх розсилка через хмарні сховища та месенджери для подальшого запуску на віддалених ПК є не тільки незручними, але й часто призводять до втрати даних або зниження безпеки.

Мета цієї роботи полягає в розробці універсального програмного рішення для оптимізації ефективності роботи працівників IT-відділу та їх інструментарію у підприємствах з великою кількістю робочих станцій. Це програмне

забезпечення має стати своєрідним "армійським швейцарським ножем", який об'єднає в собі всі необхідні функції та інструменти. Враховуючи, що більшість робочих станцій працюють на Windows 10 та 11, головним інструментом розробки буде обрано Windows Presentation Foundation (WPF) на мові програмування C#.

Програмне забезпечення повинно бути простим, але водночас функціональним. Воно має містити у власній директорії всі необхідні файли та програми, забезпечуючи зручний та дружній інтерфейс для користувачів. Набір автоматизованих рішень дозволить виконувати більшість рутинних операцій набагато швидше та ефективніше.

Завдяки такому підходу вже не потрібно буде шукати конкретну програму чи файл в іншому місці, переглядати безліч вкладок з необхідною інформацією, або витрачати час на налаштування кожного окремого інструменту.

Асинхронне підхід програмування коду дозволить додатку залишатися чутливим до дій користувача, навіть коли виконуються тривалі операції, такі як доступ до мережевих ресурсів або обробка великих обсягів даних.

У підсумку, завантажені різноманітними інструментами для простих завдань, IT-адміністратори потребують єдиного рішення. Це універсальне програмне забезпечення не тільки інвентаризуватиме обладнання та встановлюватиме програмне забезпечення, але й зберігатиме всі необхідні файли та програми в одному місці. Це забезпечить централізоване управління ресурсами, що значно спростить робочі процеси. Зручний інтерфейс програми дозволить користувачам швидко і легко знаходити потрібні інструменти та виконувати завдання.

Таке програмне рішення заощаджує час, підвищує продуктивність, мінімізує помилки та спрощує робочі процеси. Воно стане ефективним інструментом для оптимізації IT-інфраструктури підприємства.

За рахунок централізації ресурсів та автоматизації багатьох рутинних завдань, IT-відділ зможе зосередитися на більш складних та стратегічних питаннях, що сприятиме загальному покращенню ефективності роботи

підприємства.

Розробка універсального програмного рішення для IT-відділів, яке об'єднає всі необхідні інструменти та функції в одному додатку, є надзвичайно важливою для оптимізації роботи в умовах великої кількості робочих станцій.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка завдання

Ця робота пропонує розробку універсального програмного рішення для оптимізації роботи з мережевим обладнанням, комп'ютерними компонентами та програмним забезпеченням.

Програма буде централізувати інструменти для інвентаризації обладнання, управління ПЗ та спрощення робочих процесів. Вона буде зручною, надійною та сумісною з Windows 10 та 11. Очікується, що програма заощадить час, підвищить продуктивність, мінімізує помилки та оптимізує роботу системних адміністраторів.

Програма буде поділена на три головні секції:

Мережеве обладнання - модуль програми дозволятиме отримувати детальну інформацію про всі мережеві адаптери. Інформація збиратиметься автоматично та відображатиметься у зручному інтерфейсі, що зробить її легко доступною для аналізу та прийняття рішень. Збір відбуватиметься відносно найголовніших параметрів, таких як: MAC-адреса, назва, адреса в мережі, статус підключення та шлюз.

Інше обладнання - модуль має відображати детальну інформацію про кожен компонент обладнання, включаючи його характеристики, поточний стан та динамічні показники. Це дасть чітке і просте відображення про роботу комп'ютера та його параметрів для інвентаризації. При виборі комп’ютера для працівника, здебільшого звертається увага на такі компоненти, як: центральний процесор, графічний процесор, об’єм пам’яті і параметри екрану, якщо це ноутбук.

Для зручності ці дві секції міститимуть кнопки, які дозволяють швидко перейти до вбудованих сервісів Windows для управління конкретним обладнанням. Це може бути корисно для виконання більш складних налаштувань або діагностики більшого спектру проблем.

Також має бути можливість до створення звітів зібраної інформації в максимально простому і кроссплатформленому форматі, для подальшої передачі чи експортуванні до баз даних інвентаризації.

Програми - модуль управління програмним забезпеченням, який значно полегшить роботу з стороннім (додатковим) програмним забезпеченням. Функціонал буде поділено на дві категорії:

- портативні програми - категорія включатиме програми, які не потребують інсталяції та можуть запускатися з визначеної директорії. Це зробить їх зручними для використання на різних комп'ютерах без необхідності встановлення на кожному з них;
- встановлювані програми - до цієї категорії належатимуть програми, які потребують інсталяції на комп'ютер. Програма надає зручний інтерфейс для перегляду та запуску файлів інсталяції безпосередньо з інтерфейсу програми.

Розробка цієї програми сприятиме поліпшенню ефективності управління IT-інфраструктурою, забезпечуючи користувачам інструмент для моніторингу мережевими та апаратними ресурсами, а також програмним забезпеченням.

1.2 Огляд інформаційних джерел та існуючих рішень

Сучасні інформаційні системи є складними екосистемами, що складаються з різноманітних компонентів, включаючи мережеве обладнання, обчислювальні ресурси та програмне забезпечення. Забезпечення їх ефективної роботи та надійності вимагає наявності інструментів для діагностики та управління, які, в свою чергу, забезпечують оперативну інформацію про стан системи та дозволяють здійснювати необхідні корекції.

Для системних адміністраторів, які відповідають за налагодження та оптимізацію роботи інформаційних систем, діагностика та управління стають основними завданнями. Ці процеси дозволяють виявляти проблеми та недоліки у роботі системи, а також здійснювати відповідні корекції для забезпечення її оптимального функціонування.

Програмне забезпечення для діагностики та управління системами відіграє ключову роль у роботі системного адміністратора, забезпечуючи ефективний моніторинг і управління різноманітними аспектами IT-інфраструктури. Це ПЗ дозволяє адміністратору здійснювати збір, аналіз та обробку даних про різні компоненти мережі та обладнання, що є критично важливим для підтримки стабільності та продуктивності систем.

Однією з основних функцій діагностичного програмного забезпечення є робота з мережевим обладнанням. Це включає збір та аналіз інформації про мережеві адаптери, таких як статус їх підключення, MAC-адреси, IP-адреси та інші параметри. Таке ПЗ дозволяє системному адміністратору отримувати детальні дані про поточну мережеву активність, виявляти потенційні проблеми з підключенням та забезпечувати безперебійну роботу мережі.

Крім мережевого обладнання, діагностичне ПЗ також збирає інформацію про обчислювальні ресурси, такі як процесори, оперативна пам'ять і системи зберігання даних. Це включає в себе аналіз продуктивності процесорів, рівень використання пам'яті та дискових ресурсів. Системний адміністратор може використовувати ці дані для оптимізації роботи системи, виявлення вузьких місць у продуктивності та планування майбутніх оновлень обладнання.

Програмне забезпечення для діагностики може бути як портативним, так і встановлюваним на робочі станції. Портативні програми дозволяють системному адміністратору швидко запускати діагностику на будь-якому комп'ютері без необхідності інсталяції, що є дуже зручним у разі аварійних ситуацій або при діагностиці великої кількості систем. Встановлювані програми, у свою чергу, надають більш глибокі можливості для аналізу та моніторингу, оскільки можуть працювати постійно і збирати дані у реальному часі. Вибір між портативним і встановлюваним ПЗ залежить від конкретних потреб і завдань системного адміністратора.

На просторах інтернету є основні лідери в галузі діагностики і управлінням системами. Програмне забезпечення, таке як AIDA64, HWMonitor та CPU-Z, займає важливе місце у роботі системних адміністраторів, проте кожна з цих

систем має свої переваги та недоліки.

AIDA64 (рис. 1) від FinalWire Ltd. надає широкий спектр функцій, включаючи діагностику, тестування продуктивності та моніторинг системи. Однак, вона є платною і може бути складною для початківців.

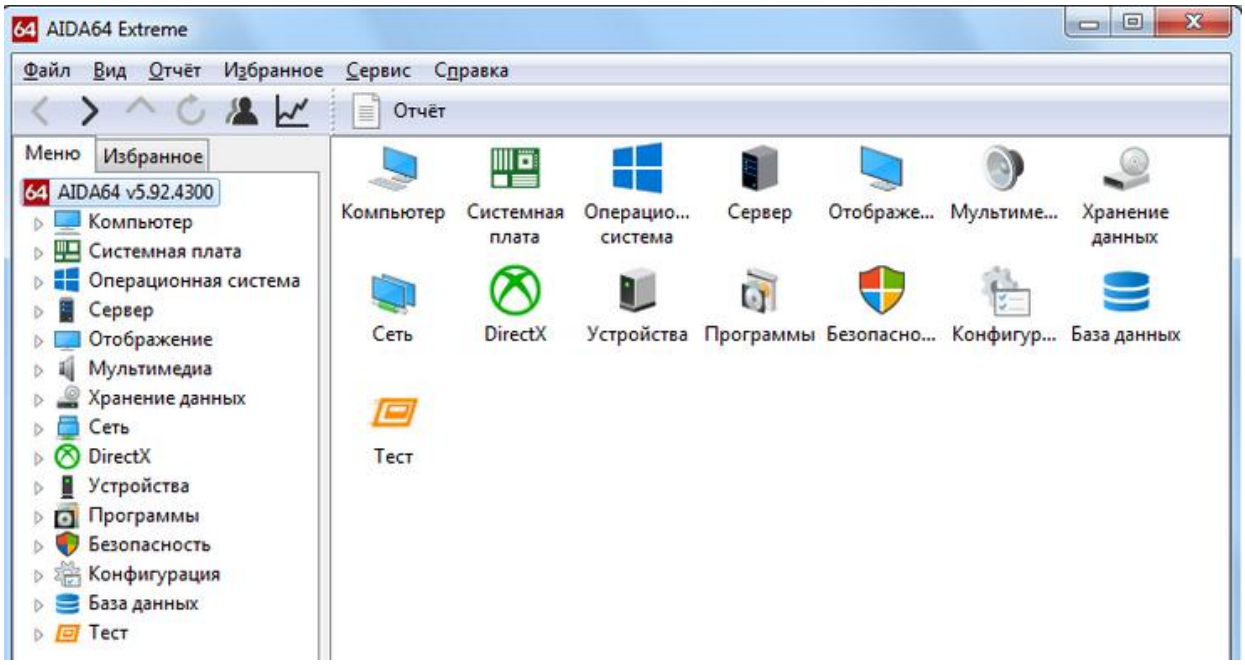

Рис.1 Інтерфейс програми AIDA64

HWMonitor (рис. 2) від CPUID, хоча є безкоштовною і має простий у використанні інтерфейс, але має менший функціонал порівняно з AIDA64, також відсутня можливість створення звітів про стан системи.

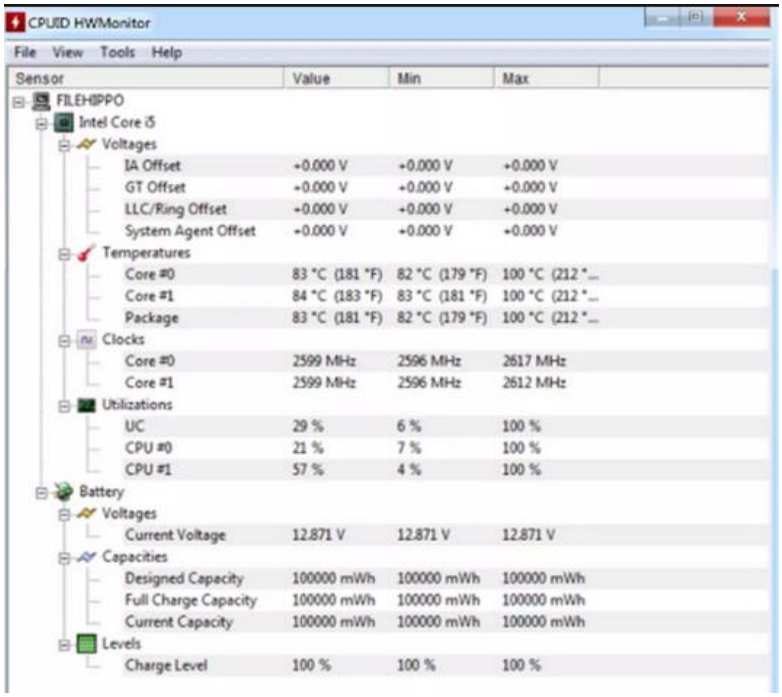

Рис. 2 Інтерфейс програми HWMonitor

CPU-Z (рис. 3) також від CPUID, безкоштовна та не вибаглива до ресурсів, проте має обмежений функціонал у порівнянні з іншими програмами. Вона дозволяє діагностувати та відображувати інформацію про апаратне забезпечення, але не має звітності про стан системи та обмежену підтримку деяких датчиків.

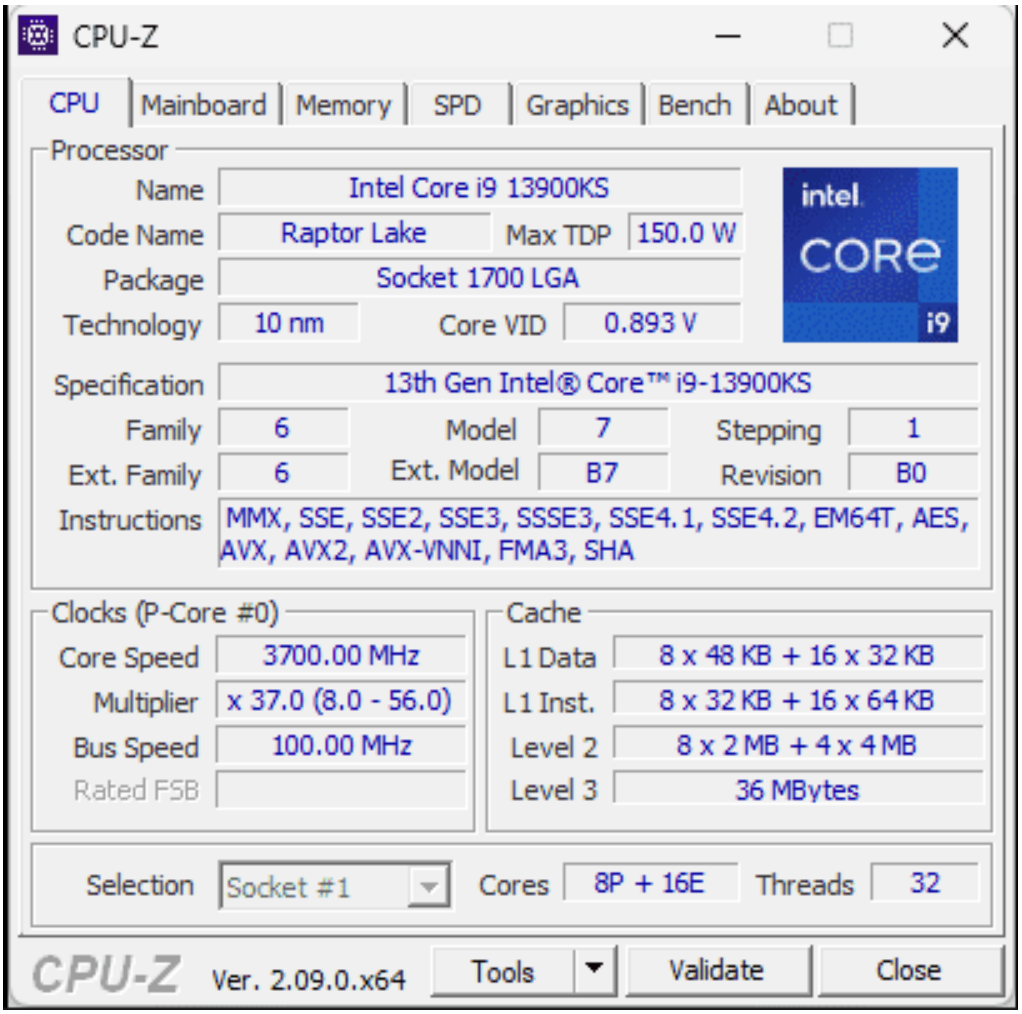

Рис. 3 Інтерфейс програми CPU-Z

Так, ці програми не надають можливості взаємодіяти з іншим програмним забезпеченням безпосередньо. Зазвичай для такої взаємодії потрібне додаткове програмне забезпечення, яке може запускати або встановлювати інші програми. Це можуть бути окремі програми або компоненти, які входять до складу інших програм або операційних систем.

Табличне відображення порівняння цих програмних рішень представлено нижче в таблиці 1:

Таблиця 1

Порівняння систем

Параметр	AIDA64	HWMonitor	CPU-Z
Відображення показань з датчиків	+	+	+
Моніторинг в режимі реального часу	+	-	+
Простий інтерфейс	+	+	-
Створення звітів	+	-	-
Підтримка широкого спектру апаратних платформ	+	+	+
Безкоштовна ліцензія	-	+	+

1.3 Визначення основних вимог

У процесі планування і розробки програмного забезпечення важливо враховувати як функціональні, так і нефункціональні вимоги.

Функціональні вимоги визначають конкретні завдання, які повинна виконувати програма, наприклад, збір інформації з датчиків чи створення звіту. Нефункціональні вимоги стосуються таких якостей системи, як швидкодія, надійність, зручність використання, що є критичними для забезпечення ефективної роботи та взаємодії з самою програмою.

Функціональні вимоги (окремо для кожної секції)

- ❖ Мережеве обладнання:
  - отримання та відображення інформації про мережеві адаптери;
  - виведення параметрів:

- пінг;
- mac-адреса;
- назва;
- мережева адреса;
- статус підключення;
- шлюз.

- кнопка для швидкого доступу до вбудованих сервісів управління мережевими підключеннями;
- можливість створення та експорту звітів.

❖ Обладнання:

- поділ на три групи: процесори (gpu і cpu), пам'ять (диски та озу) та екран;
- отримання і виведення статичної інформації про обладнання: назви моделей, кількість ядер, слотів, об'єми пам’яті, частоти, роздільчої здатності;
- динамічне відображення завантаження компонентів;
- кнопки для швидкого доступу до вбудованих сервісів управління конкретним обладнанням;
- можливість створення та експорту звітів.

❖ Програми:

- відображення та запуск портативних;
- відображення та запуск інсталяційних файлів;
- автоматичне підключення до директорій з ПЗ.

Нефункціональні вимоги:

- інтуїтивний і простий інтерфейс для сприйняття та швидкого доступу до всіх функцій програми;
- стабільна робота програми без збоїв та помилок;
- підтримка операційних систем windows 10 та 11.

Для системних адміністраторів, які відповідають за налагодження, інвентаризацію та оптимізацію роботи обладнання, взаємозв'язок між

функціональними і нефункціональними вимогами є ключовим, адже ефективність їх роботи є добутком зручності і швидкості з функціональністю.

1.4. Моделювання предметної області

Моделювання предметної області – це процес створення абстрактного представлення реальної системи, що відображає її основні аспекти та взаємозв’язки. Це дозволяє краще зрозуміти складні системи, описати їхні основні компоненти, взаємодії між ними, а також важливо визначити вимоги до кінцевого продукту.

Нижче буде наведено декілька основних діаграм, що допомагають зрозуміти загальну концепцію розроблюваного програмного забезпечення в цій роботі:

На рис 4 відображено діаграму прецедентів до розроблюваної системи. Вона є важливим інструментом у процесі моделювання вимог для системи. Вона допомагає візуалізувати взаємодію між користувачами та системою, а також визначити основні функціональні можливості програмного забезпечення. Ця діаграма надає зрозумілу картину того, які конкретні дії може виконувати користувач у системі та які результати він очікує.

Кожен прецедент у діаграмі представляє окремий функціональний елемент системи, що може бути викликаний або ініційований користувачем. Ці прецеденти можуть описувати різні сценарії взаємодії з системою, включаючи створення, зміну або видалення даних, отримання звітів або виконання інших дій.

Кожен прецедент також містить список акторів, які беруть участь у взаємодії з системою. Актори можуть бути різними видами користувачів, зовнішніми системами або навіть іншими компонентами програмного забезпечення.

Діаграма прецедентів створює зрозумілу та структуровану модель взаємодії між користувачами та системою, що допомагає уникнути непорозумінь та недоліків у сприйнятті вимог.

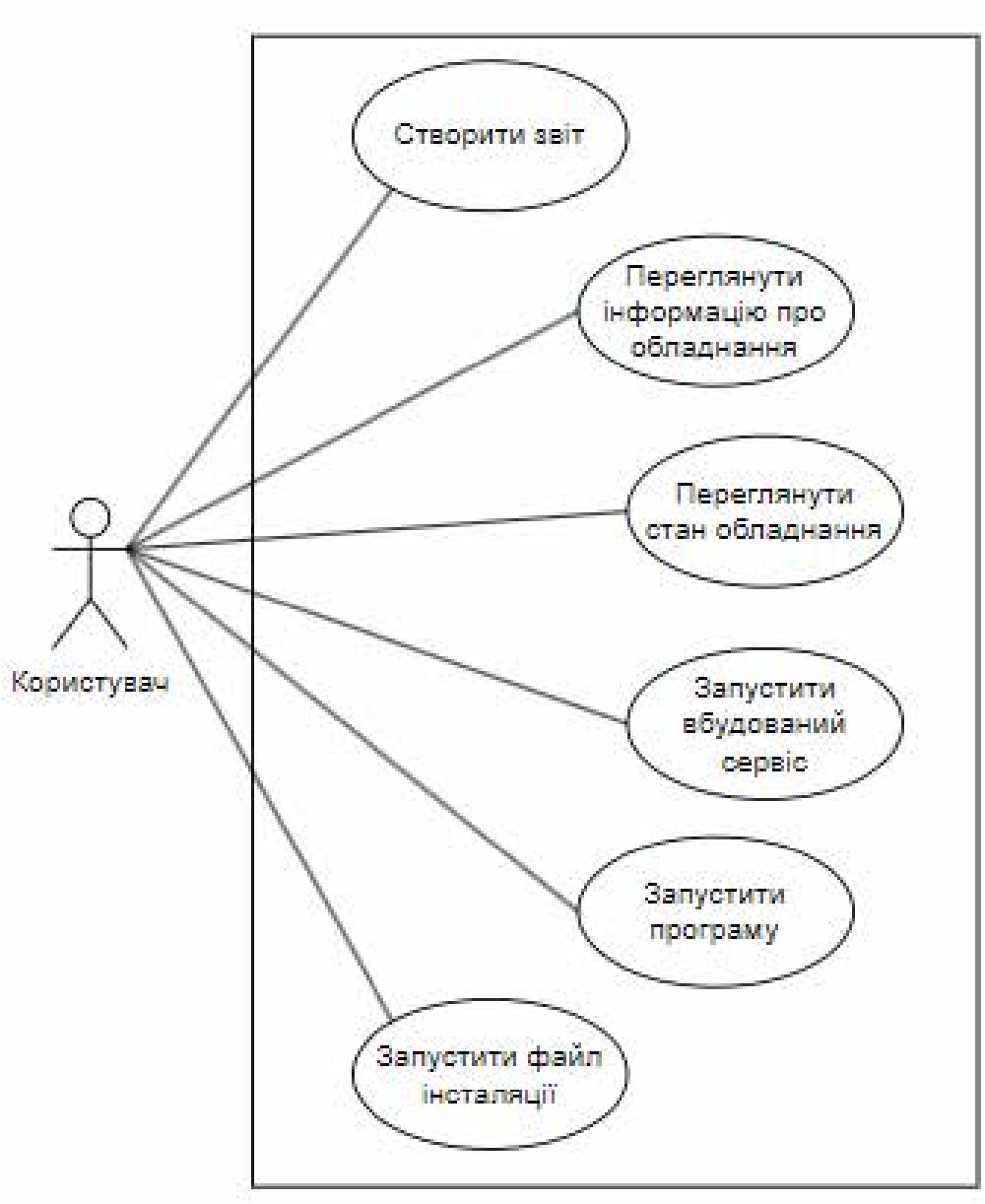

Рис. 4 Діаграма прецендентів

Діаграма послідовності - це інструмент в аналізі та проектуванні систем, призначений для візуалізації послідовності взаємодії між об'єктами або компонентами системи у часі. Вона дозволяє чітко уявити, як об'єкти взаємодіють один з одним у системі, подаючи послідовність подій або операцій.

Цей тип діаграми стає особливо корисним для розуміння та аналізу логіки роботи системи перед її реалізацією або під час вдосконалення. Вона допомагає ідентифікувати, які об'єкти взаємодіють між собою, в якому порядку ця взаємодія відбувається та які дані чи повідомлення передаються між ними.

На діаграмі послідовності зображуються об'єкти, які беруть участь у

взаємодії, та стрілки, що показують послідовність викликів методів або обмін повідомленнями між ними. Час відображається по горизонталі, а взаємодія між об'єктами - по вертикалі.

Отже, діаграма послідовності створює зрозумілу та легко відстежувану модель взаємодії, яка допомагає аналізувати та розуміти функціональні аспекти системи, що розробляється або вдосконалюється. На рис 5 відображено діаграму прецедентів до розроблюваної системи.

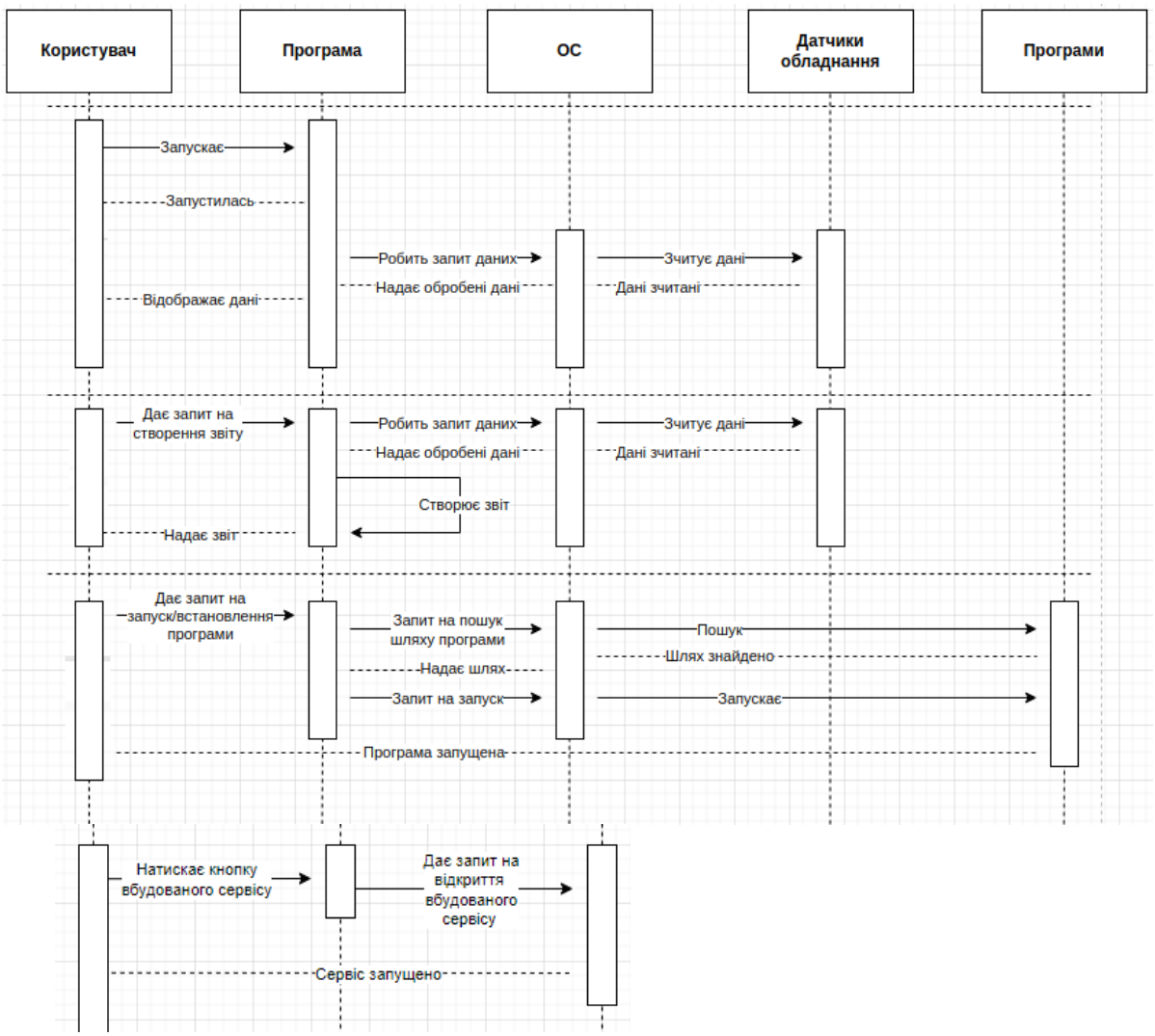

Рис. 5 Діаграма послідовності

Використання діаграм послідовності допомагає уникнути непорозумінь та помилок у специфікації системи, допомагає збільшити продуктивність у процесі розробки.

Вона також може бути використана для навчання нових користувачів або

співробітників щодо правильного використання системи.

1.5. Висновки до першого розділу

Створення постановки завдання та огляд інформаційних джерел та існуючих рішень є важливим етапом у розробці програмного забезпечення, оскільки вони надають чітке уявлення про мету та функціонал майбутнього продукту. Основна мета цього програмного рішення полягає в створенні універсального інструменту, спрямованого на централізацію інвентаризації обладнання, управління програмним забезпеченням та спрощення робочих процесів. Програма буде розділена на три основні секції та надаватиме можливість генерації звітів

Для досягнення поставленої мети було проведено визначення основних вимог до програмного забезпечення та моделювання предметної області. Моделювання предметної області включало створення абстрактного представлення реальної системи з її ключовими аспектами та взаємозв'язками. Це дозволило краще зрозуміти складність системи, описати її ключові компоненти та взаємодії між ними, а також визначити вимоги до кінцевого продукту.

Проведення аналізу і визначення вимог є важливим етапом у розробці програмного забезпечення, бо забезпечує відповідність розробленого продукту потребам та очікуванням у користуванні. В результаті цього аналізу можна побачити, що програма має великий потенціал для полегшення робочих процесів та оптимізації управління обладнанням та програмним забезпеченням.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Логічна модель даних

Логічна модель даних є ключовим елементом будь-якого програмного забезпечення, оскільки вона визначає структуру даних, їх взаємозв'язки та правила обробки.

Ця структурована організація даних дозволяє ефективно керувати великим обсягом різноманітної інформації, отриманої від компонентів, що використовуються програмою, полегшуючи їх подальшу обробку та аналіз. Завдяки цьому, розробники можуть забезпечити зручний доступ до даних, їхню цілісність та узгодженість, а також оптимізувати запити до бази даних для підвищення продуктивності системи. На рис 6 зображена логічна модель даних розроблюваної системи.

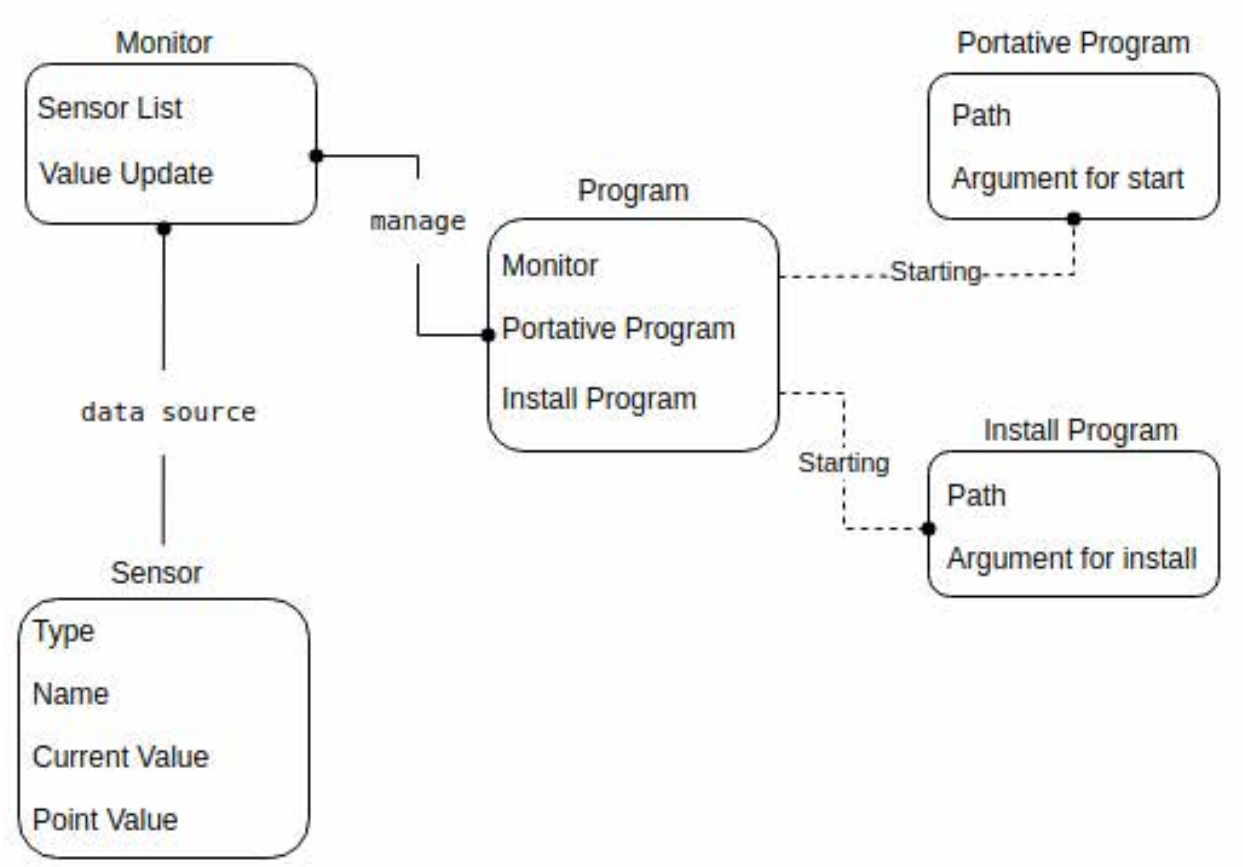

Рис. 6 Модель даних

Модель даних розроблюваного програмного забезпечення складається з п'яти основних елементів: датчиків, монітора, програми, портативної програми та файлу інсталяції програми. Датчики відповідають за збір інформації з різних джерел, що забезпечує наявність актуальних даних для подальшої обробки. Монітор обробляє зібрані датчиками дані, здійснює їх аналіз і передає результати до основної програми. Програма виступає головним елементом системи, об'єднуючи всі інші компоненти та керуючи їхньою роботою.

Портативна програма та файл інсталяції програми є спеціалізованими компонентами, які запускаються основною програмою для виконання різних завдань. Портативна програма забезпечує можливість запуску додаткових функцій без потреби в інсталяції, що зручно для користувачів, які працюють на різних пристроях. Файл інсталяції програми дозволяє встановлювати необхідні додатки або оновлення, інтегруючи їх у загальну систему. Уся ця структура сприяє ефективній роботі програмного забезпечення, забезпечуючи зручне

управління даними та взаємодію між різними компонентами.

2.2. Діаграма класів

Діаграма класів є одним із основних типів діаграм у мові моделювання UML. Вона використовується для опису статичної структури системи, показуючи класи системи, їх атрибути, методи, а також взаємозв'язки між класами. Діаграма класів є важливим інструментом для розробників, оскільки вона забезпечує візуальне уявлення про структуру програми та взаємодію між її елементами.

Основні зв'язки між класами

- ❖ Асоціація:

Опис: Вказує на зв'язок між двома класами.

Відображення: Відображається у вигляді лінії між класами, іноді з маркерами ролей та множинності. Маркери ролей показують роль, яку кожен клас відіграє у зв'язку, а множинність вказує на кількість екземплярів класів, які можуть брати участь у зв'язку.

- ❖ Агрегація:

Опис: Спеціальний тип асоціації, що представляє відношення "ціле-частина".

Відображення: Відображається лінією з порожнім ромбом на кінці, що примикає до класу-цілого. Це означає, що частини можуть існувати незалежно від цілого.

- ❖ Композиція:

Опис: Більш жорстка форма агрегації, де частина не може існувати без цілого.

Відображення: Відображається лінією з заповненим ромбом на кінці, що примикає до класу-цілого. Це означає, що частини знищуються разом із цілим.

- ❖ Наслідування:

Опис: Вказує на відношення "є-нащадком" між класами.

Відображення: Відображається суцільною лінією зі стрілкою з відкритим трикутником, яка вказує на базовий клас. Це означає, що похідний клас успадковує атрибути та методи базового класу.

- ❖ Залежність:

Опис: Вказує, що один клас використовує інший.

Відображення: Відображається пунктирною лінією зі стрілкою, яка вказує на клас, від якого залежить інший. Це означає, що зміни в класі, від якого залежить інший, можуть вплинути на клас, який залежить.

Діаграма класів є важливим інструментом для аналізу та проєктування системи. Вона допомагає розробникам визначити основні об'єкти системи, їх атрибути та методи, а також взаємозв'язки між ними. Завдяки діаграмі класів можна виявити потенційні проблеми в дизайні на ранніх етапах розробки та внести необхідні корективи. Рис 7 демонструє діаграму класів розроблюваної системи.

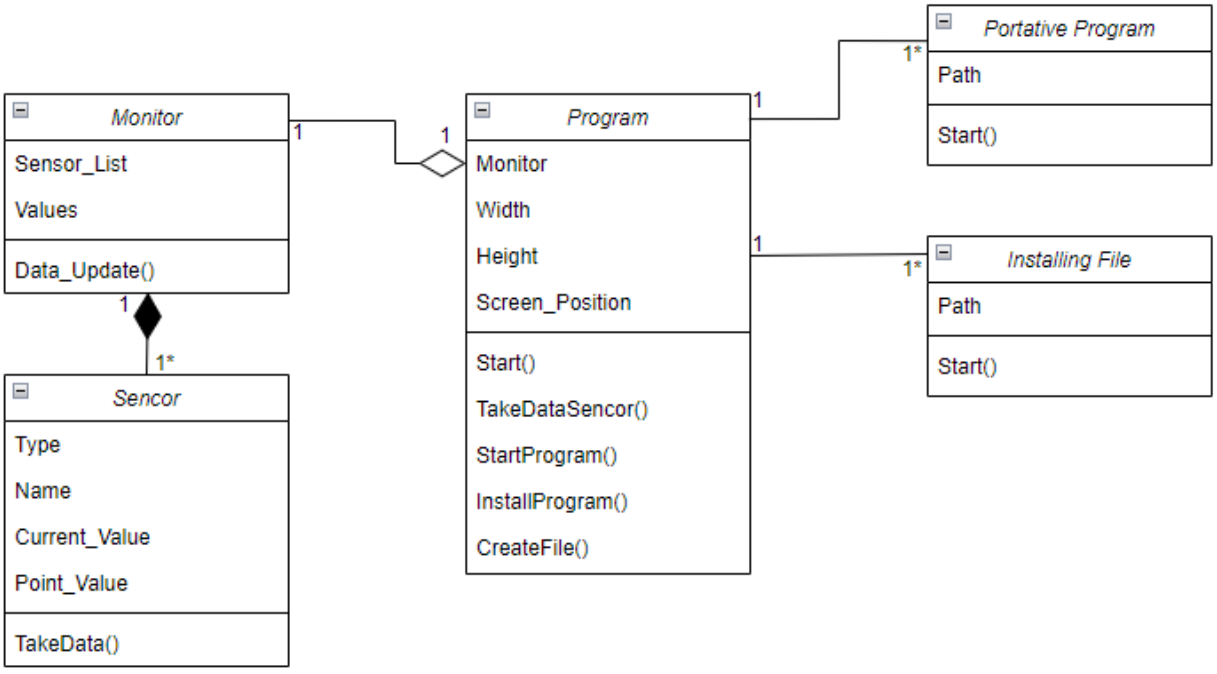

Рис.7 Діаграма класів

Центральний клас Program є головним компонентом і відповідає за запуск, управління та взаємодію з іншими класами системи. Він включає в себе наступні методи:

Start(): Основний метод, який ініціалізує та запускає всю програму;

TakeDataSensor(): Метод, який збирає дані з класу Monitor;

StartProgramm(): Метод для запуску портативної програми;

InstallProgramm(): Метод для встановлення програми;

CreateFile(): Метод для створення звіту.

Опис інших класів

Portative та InstallProgram:

Відповідають за виконавчі файли у відповідних директоріях.

Асоційовані з класом Program, надаючи функціональність для запуску та встановлення програм. Ці класи забезпечують виконання необхідних дій для портативних і встановлюваних програм, інтегруючись з основним процесом керування.

Monitor:

Відповідає за отримання даних від екземплярів класу Sensor.

Оновлює показники датчиків і агрегаціюється з класом Program, забезпечуючи функціональність збору даних. Monitor збирає, обробляє та оновлює інформацію, яку надають датчики, дозволяючи програмі отримувати актуальні дані для подальшої обробки.

Sensor:

Містить атрибути та методи для зберігання та оновлення показників датчиків.

Є частиною композиції з класом Monitor, що означає, що датчики існують лише в контексті монітора. Sensor надає конкретні дані про стан або параметри, які Monitor збирає та передає програмі.

Взаємозв'язки між класами

Асоціація між Program і Portative та InstallProgram:

Ці класи співпрацюють для забезпечення функціональності запуску та встановлення програм.

Агрегація між Program і Monitor:

Monitor взаємодіє з Program для надання даних, зібраних від Sensor.

Композиція між Monitor і Sensor:

Датчики є невід'ємною частиною монітора і не можуть існувати окремо від

нього.

Функціональність класів

Program:

Є центральною точкою запуску та управління програмою.

Використовує методи для ініціалізації програми, збору даних, запуску та встановлення програм, а також створення звітів.

Portative:

Відповідає за запуск портативних програм.

Залежить від Program для забезпечення своєї функціональності.

InstallProgram:

Відповідає за встановлення програм.

Інтегрується з Program для управління процесом інсталяції.

Monitor:

Збирає дані від Sensor.

Взаємодіє з Program для надання актуальних даних.

Sensor:

Надає показники та інформацію для Monitor.

Є компонентом, що оновлюється та підтримується Monitor.

Ця структура забезпечує чіткий розподіл відповідальності між класами та їх методами, дозволяючи програмі ефективно виконувати свої завдання. Взаємозв'язки між класами відображають логічну структуру системи, сприяючи кращому розумінню та легшому підтриманню коду.

2.3. Діаграма пакетів

У тій же мові моделювання UML є діаграма, що використовується для відображення організації та структуризації системи на рівні пакетів. Це полегшує управління класами та іншими елементами у програмних системах, забезпечуючи впорядкованість і логічну ієрархію. Пакети можуть містити класи,

інтерфейси, підпакети та інші компоненти, що допомагає організувати та ієрархічно структурувати модель. Вони забезпечують чітке розмежування відповідальності між різними частинами системи, що сприяє кращому розумінню взаємодій між модулями та полегшує їхнє обслуговування і розвиток.

Основні елементи діаграми пакетів – це пакети і залежності між ними. Пакети виступають основними контейнерами, що групують елементи моделі, і представлені у вигляді прямокутників із вкладеною вкладкою зверху ліворуч, яка нагадує папку. Ці вкладки символізують вміст пакетів та їхні підпакети, надаючи зручну візуалізацію для групування схожих елементів. Залежності відображають відношення між пакетами, показуючи, що один пакет використовує або залежить від іншого, що є важливим для розуміння потоків інформації та взаємозв'язків у системі. Залежності представляються пунктирними стрілками, що вказують напрямок залежності та взаємодії між пакетами.

Діаграма пакетів є важливим інструментом для планування і організації архітектури програмного забезпечення. Вона дозволяє розробникам чітко бачити, як різні частини системи взаємодіють одна з одною, що сприяє кращому розподілу завдань і відповідальності серед членів команди. Завдяки чітко визначеним межам і взаємозв'язкам між пакетами, розробники можуть ефективніше управляти залежностями та уникати складних і заплутаних структур.

Окрім цього, діаграма пакетів спрощує процес документування архітектури програмного забезпечення, полегшуючи передачу знань між командою та новими розробниками. Вона також допомагає визначити модулі системи, що можуть бути розроблені та протестовані незалежно один від одного, що підвищує ефективність та гнучкість розробки. Завдяки цьому, діаграма пакетів сприяє створенню добре структурованих, підтримуваних та масштабованих програмних систем.

Таким чином, діаграма пакетів у UML є надзвичайно корисним інструментом для візуалізації та організації системи на високому рівні абстракції. Вона забезпечує чітке уявлення про структуру і взаємозв'язки між

різними частинами системи, сприяючи кращому розумінню, управлінню та розвитку програмного забезпечення. На рис 8 зображено діаграму пакетів розроблюваної системи.

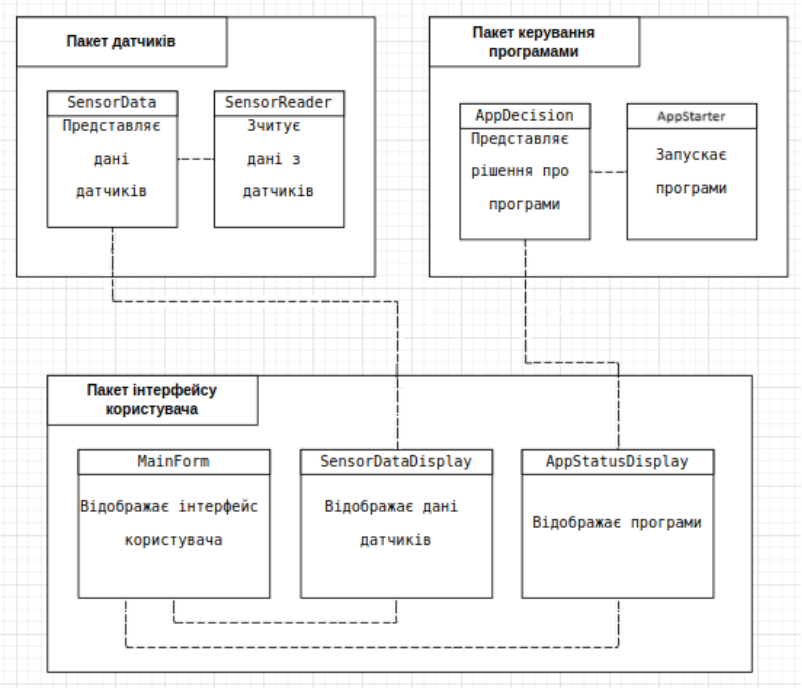

Рис. 8 Діаграма пакетів

Для опису структури системи використовується діаграма пакетів, яка складається з трьох основних пакетів: Пакет датчиків, Пакет керування програмами та Пакет інтерфейсу.

Пакет датчиків відповідає за збір та первинну обробку даних з різних джерел. Він містить класи та компоненти, що безпосередньо працюють з датчиками, забезпечуючи актуальну інформацію про стан системи.

Пакет керування програмами включає компоненти, відповідальні за управління програмним забезпеченням. Це охоплює портативні програми, які можна запускати без попередньої інсталяції, та інсталяційні файли програм, які потребують встановлення. Цей пакет забезпечує функціональність для інсталяції, запуску та управління додатками, інтегруючи їх у загальну систему.

Пакет інтерфейсу містить компоненти, що відповідають за взаємодію з користувачем. Він включає графічний інтерфейс користувача (GUI), що забезпечує зручний доступ до функціоналу системи, дозволяючи користувачам

легко керувати датчиками та програмами. Інтерфейс надає засоби для візуалізації даних, зібраних датчиками, та можливість взаємодії з іншими компонентами системи через інтуїтивно зрозумілі елементи управління.

Такий розподіл на пакети забезпечує чітку структуру та організацію системи, що полегшує розробку, підтримку та розширення програмного забезпечення, забезпечуючи ефективну взаємодію між різними його частинами.

2.4 Діаграма компонентів

За графічне зображення фізичних аспектів інформаційної системи в UML відповідає діаграма компонентів. Вона показує організацію і залежності між програмними компонентами, такими як модулі, бібліотеки або сервіси, що складають систему. Діаграма компонентів відображає, як ці частини взаємодіють між собою та які інтерфейси використовують для комунікації. Це дозволяє розробникам і архітекторам програмного забезпечення отримати чітке уявлення про структуру системи і зрозуміти, як різні елементи співпрацюють для досягнення загальної функціональності.

Структура діаграми компонентів включає кілька ключових елементів. Основні з них - це компоненти, які представляють окремі частини системи. Компоненти зображуються у вигляді прямокутників з двома маленькими прямокутниками зверху, що символізує їх модульний характер. Залежності між компонентами зображуються пунктирними стрілками, що вказують на напрямок залежності, що дозволяє легко ідентифікувати, які компоненти залежать від інших. Такі залежності можуть включати використання одного компоненту іншими, зв'язки між інтерфейсами та реалізацію функціональності.

Діаграма компонентів є важливим інструментом для побудови архітектури програмного забезпечення. Вона допомагає зрозуміти, як різні частини системи пов'язані між собою, що сприяє ефективній розробці та підтримці програмного забезпечення. Завдяки модульності, яку відображає діаграма, можна розробляти і тестувати окремі компоненти незалежно один від одного. Це значно підвищує гнучкість і масштабованість системи, оскільки кожен компонент можна

оновлювати або замінювати без значного впливу на інші частини системи.

Крім того, діаграма компонентів є корисною для документування архітектури системи, що полегшує передачу знань між членами команди та новими розробниками. Вона також може служити основою для визначення завдань і відповідальності в команді, оскільки кожен компонент може бути призначений конкретному розробнику або групі розробників. Це сприяє організованості роботи та ефективному управлінню проектами.

Таким чином, діаграма компонентів у UML є потужним засобом для візуалізації і планування архітектури програмного забезпечення. Вона забезпечує чітке уявлення про структуру системи, допомагає зрозуміти взаємозв'язки між її частинами і сприяє модульному підходу до розробки, що в свою чергу забезпечує ефективність, гнучкість і масштабованість програмного рішення. На рис 9 зображена діаграма компонентів проекту.

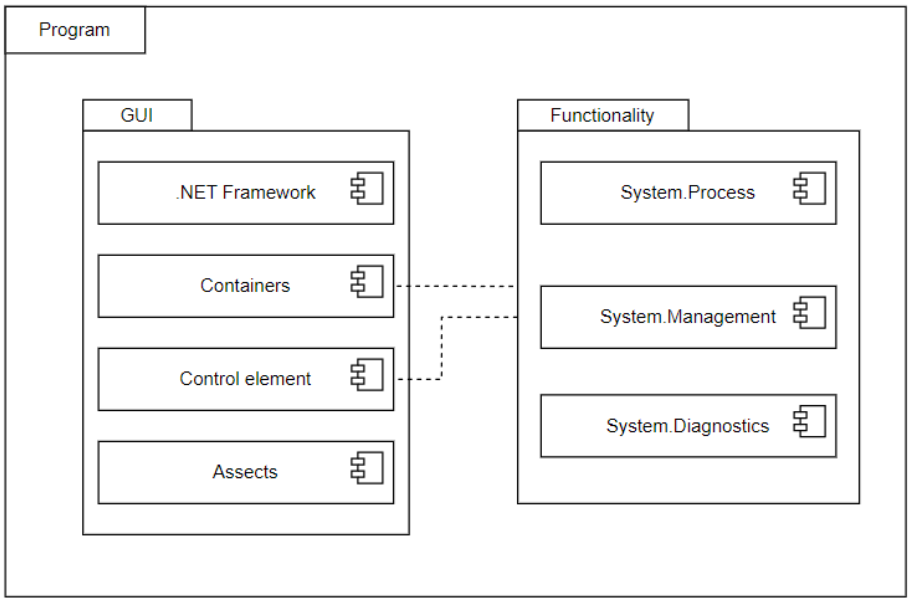

Рис. 9 Діаграма компонентів

Діаграма представляє структуру програми, яка складається з двох основних контейнерів: GUI та Functionality. Ці контейнери взаємодіють між собою, забезпечуючи функціональність програми та її інтерфейс. Кожен контейнер містить певні компоненти, що виконують конкретні завдання, які разом формують цілісну архітектуру програми.

Структура контейнера GUI

Контейнер GUI (графічний інтерфейс користувача) включає чотири основні компоненти:

.Net Framework - це фундаментальна частина, яка забезпечує основні функціональні можливості та середовище виконання для інших компонентів інтерфейсу. .Net Framework включає в себе бібліотеки класів, що підтримують розробку різноманітних застосунків, і дозволяє використовувати єдине середовище виконання для різних типів додатків, забезпечуючи при цьому високу продуктивність та сумісність.

Containers - це структурні елементи інтерфейсу, які організовують інші елементи. Вони служать для групування та розміщення контрольних елементів, що робить інтерфейс більш логічним і зручним для користувача. Containers можуть включати в себе панелі, вкладки та інші контейнери, які допомагають розділити інтерфейс на окремі логічні частини.

Control Element - це елементи інтерфейсу, з якими користувач безпосередньо взаємодіє. Вони включають кнопки, текстові поля, чекбокси, випадаючі списки та інші елементи, які дозволяють користувачу вводити дані та керувати програмою. Control Elements є ключовими для забезпечення інтерактивності та зручності використання програми.

Assets - це ресурси, які використовуються у графічному інтерфейсі. Вони включають зображення, іконки, шрифти, стилі та інші візуальні елементи, які покращують вигляд і сприйняття програми. Assets забезпечують єдиний стиль і дизайн інтерфейсу, що робить програму привабливою та легкою у використанні.

Структура контейнера Functionality

Контейнер Functionality містить компоненти, відповідальні за основну функціональність програми:

System.Process - цей компонент забезпечує управління та моніторинг процесів. Він дозволяє програмі запускати, зупиняти і контролювати різні процеси, а також отримувати інформацію про їхній стан та використання

ресурсів. System.Process є важливим для виконання завдань, пов'язаних з керуванням іншими програмами та системними процесами.

System.Management - цей компонент надає інтерфейси для управління та доступу до інформації про систему. Він включає інструменти для роботи з Windows Management Instrumentation (WMI), що дозволяє отримувати детальну інформацію про апаратне забезпечення, операційну систему, налаштування мережі та інші аспекти системи. System.Management є ключовим для моніторингу та керування системою.

System.Diagnostics - цей компонент містить інструменти для діагностики, ведення журналів та трасування роботи програми. Він дозволяє розробникам відслідковувати помилки, аналізувати продуктивність та вести журнал подій, що відбуваються в програмі. System.Diagnostics допомагає забезпечити надійність та стабільність роботи програми, спрощуючи процеси налагодження та оптимізації.

Взаємодія між контейнерами

Контейнер Functionality прив'язаний до Control Element в контейнері GUI. Це означає, що елементи інтерфейсу взаємодіють з основною функціональністю програми, забезпечуючи інтерактивний та функціональний досвід користувача. Наприклад, коли користувач натискає кнопку або вводить дані в текстове поле, відповідні Control Elements передають ці дії компонентам у контейнері Functionality, які обробляють запити і виконують необхідні операції.

Такий підхід дозволяє розділити логіку інтерфейсу та основну функціональність програми, що сприяє кращій організації коду, його масштабованості та підтримуваності. Це також дозволяє легше оновлювати або змінювати окремі частини системи без значного впливу на інші компоненти, що забезпечує більшу гнучкість та адаптивність програми до змінних вимог.

2.5 Інструмент WMI

Основним джерелом інформації для програми є WMI (Windows Management Instrumentation) - потужний інструментарій від Microsoft, вбудований у Windows. WMI базується на моделі Common Information Model

(CIM), яка забезпечує стандартизований спосіб доступу до інформації про комп'ютерні системи та пристрої. ЇЇ схематика зображена на рис 10.

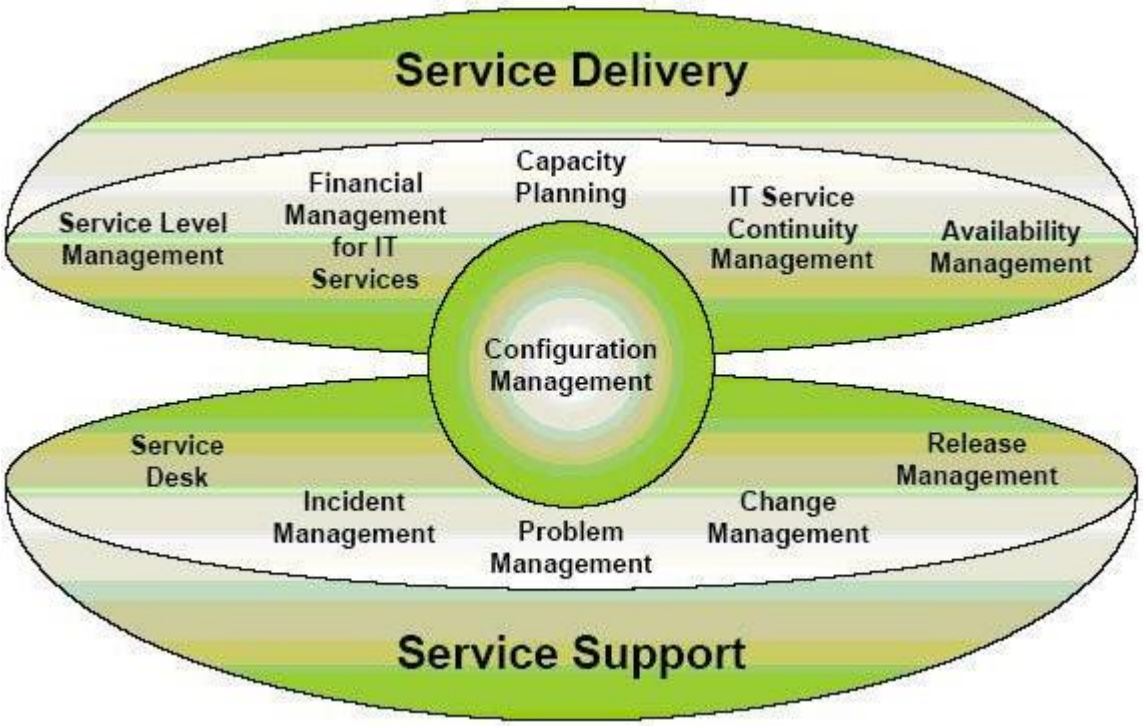

Рис. 10 Схематика CIM

Цей інструмент надає можливість отримувати та керувати детальною інформацією про різні аспекти операційної системи та апаратного забезпечення, що робить його надзвичайно корисним для розробки та адміністрування програмного забезпечення.

WMI є надзвичайно потужним інструментом, і його можливості обширні. Наприклад, для отримання інформації про систему WMI може надавати дані про різні аспекти системи, такі як апаратне забезпечення, операційну систему, програмне забезпечення, користувачів та мережеві налаштування. Це дозволяє розробникам отримувати всю необхідну інформацію для аналізу та управління системою. Крім того, WMI можна використовувати для керування системою: запуску та зупинки служб, зміни налаштувань системи та оновлення драйверів, що значно спрощує процес адміністрування.

Моніторинг системи за допомогою WMI дозволяє відстежувати продуктивність системи, виявляти проблеми та отримувати сповіщення про події, що сприяє швидкому реагуванню на будь-які неполадки. Автоматизація

завдань є ще однією важливою функцією WMI, яка дозволяє автоматизувати завдання адміністрування системи, наприклад, інсталяцію програмного забезпечення, налаштування користувачів та резервне копіювання даних. Це значно знижує ручну працю та підвищує ефективність роботи системних адміністраторів.

WMI, не зважаючи на свою вбудованість у Windows, є найкращим вибором для вирішення задач розроблюваного програмного рішення, адже має ключові переваги.

По-перше, стандартизованість: WMI ґрунтується на CIM, що робить його стандартизованим способом доступу до інформації про системи Windows. Це означає, що будь-який додаток, який використовує WMI, може отримувати узгоджену та структуровану інформацію незалежно від версії операційної системи або апаратної платформи.

По-друге, потужність: WMI надає широкий спектр можливостей для управління та моніторингу систем Windows. Його функціональні можливості включають доступ до детальної інформації про систему, управління ключовими аспектами операційної системи, а також моніторинг її стану і продуктивності.

По-третє, гнучкість: WMI можна використовувати з різними інструментами та мовами програмування, що робить його надзвичайно гнучким. Наприклад, його можна використовувати в PowerShell для створення скриптів адміністрування або в таких мовах програмування як C# для розробки потужних додатків. Використання функціоналу WMI доволі просте – звернення до вбудованої бази даних можна здійснити вже у дві строки коду. Наприклад, на C# це виглядає як зображено на рис 11:

```
ManagementObjectSearcher searcher =
new ManagementObjectSearcher("select * from Win32_Processor");
foreach (ManagementObject obj in searcher.Get())
{
    // перегляд зібраних даних
}
```

Рис. 11 Використання функціоналу WMI

Використання WMI як основного джерела інформації для програми є стратегічним вибором, враховуючи його вбудованість, потужність та гнучкість. Вміле застосування можливостей WMI дозволяє створювати ефективні рішення для моніторингу, управління та автоматизації систем Windows. Завдяки стандартизованому підходу та широким можливостям, WMI залишається ключовим інструментом для роботи з системами Windows. Це забезпечує надійну та ефективну роботу програмного забезпечення, оптимізуючи процеси адміністрування та управління IT-інфраструктурою.

2.6 Висновки до другого розділу

Результатом другого розділу роботи є детальне пояснення інформаційного забезпечення програмного рішення, включаючи логічну модель даних, діаграми класів, пакетів та компонентів.

Логічна модель даних визначає структуру та взаємозв'язки між ключовими елементами системи. Вона описує, як дані організовані та як вони взаємодіють один з одним, що забезпечує фундаментальне розуміння системи та її функціонування. Це є важливим етапом у процесі розробки, оскільки дозволяє чітко визначити основні елементи даних і їхні взаємозв'язки, що необхідно для побудови ефективної та надійної системи.

Діаграма класів описує статичну структуру системи, показуючи взаємодію між основними класами та їх методами. Вона надає зрозуміле уявлення про роботу системи, відображаючи основні класи, їх атрибути, методи та зв'язки між ними. Це допомагає зрозуміти, як окремі частини системи взаємодіють між

собою, і дозволяє виявити потенційні проблеми та невідповідності на ранньому етапі розробки.

Діаграма пакетів відображає організацію класів на рівні пакетів, що допомагає структурувати та ієрархічно організувати модель. Це сприяє кращій організації коду, робить його більш зрозумілим та підтримуваним. Організація класів у пакети дозволяє чітко розмежувати різні частини системи, що спрощує їх обслуговування та розвиток.

Діаграма компонентів ілюструє фізичну організацію та залежності між програмними компонентами, визначаючи, як GUI та функціональні компоненти взаємодіють для забезпечення повної функціональності програми. Ця діаграма відображає, як окремі модулі системи з'єднані між собою, що допомагає краще зрозуміти архітектуру програми та оптимізувати її продуктивність.

Цей аналіз показує, що розробка інформаційного забезпечення охоплює всі основні аспекти проектування програмного забезпечення, від логічного моделювання даних до фізичного розташування компонентів. Такий підхід забезпечує цілісність і функціональність системи, дозволяючи створити добре структуроване, ефективне та надійне програмне забезпечення

Загалом, такий підхід до розробки інформаційного забезпечення забезпечує цілісність і функціональність системи, дозволяючи створити добре структуроване, ефективне та надійне програмне забезпечення. Кожен аспект проектування, від моделювання даних до фізичного розташування компонентів, відіграє важливу роль у забезпеченні того, щоб кінцевий продукт був не лише функціональним, але й легко підтримуваним і гнучким до змін. Цей всебічний підхід гарантує, що програмне забезпечення буде відповідати високим стандартам якості та задовольняти всі вимоги користувачів.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Орієнтована операційна система

Операційна система Windows, розроблена компанією Microsoft, є однією з найбільш відомих і поширених операційних систем у світі. Її популярність та універсальність зробили Windows ключовим гравцем у сфері інформаційних технологій. Логотипи ОС зображено на рис 12.

Рис. 12 Логотипи Windows

Windows залишається найбільш поширеною операційною системою для персональних комп'ютерів. За даними різних досліджень, Windows займає приблизно 75-80% ринку операційних систем для настільних ПК.

Одним із ключових чинників її популярності є широка підтримка програмного забезпечення. Windows підтримує величезну кількість програм і ігор, що робить її ідеальною платформою для різних користувачів — від геймерів до професіоналів у різних галузях.

Більшість комерційних програмних продуктів, таких як Microsoft Office, Adobe Creative Suite та інші, мають версії для Windows, що забезпечує широкий

спектр можливостей для роботи та розваг. Доля ринку, що спадає на операційну систему Windows, зображена діаграмою на рис 13.

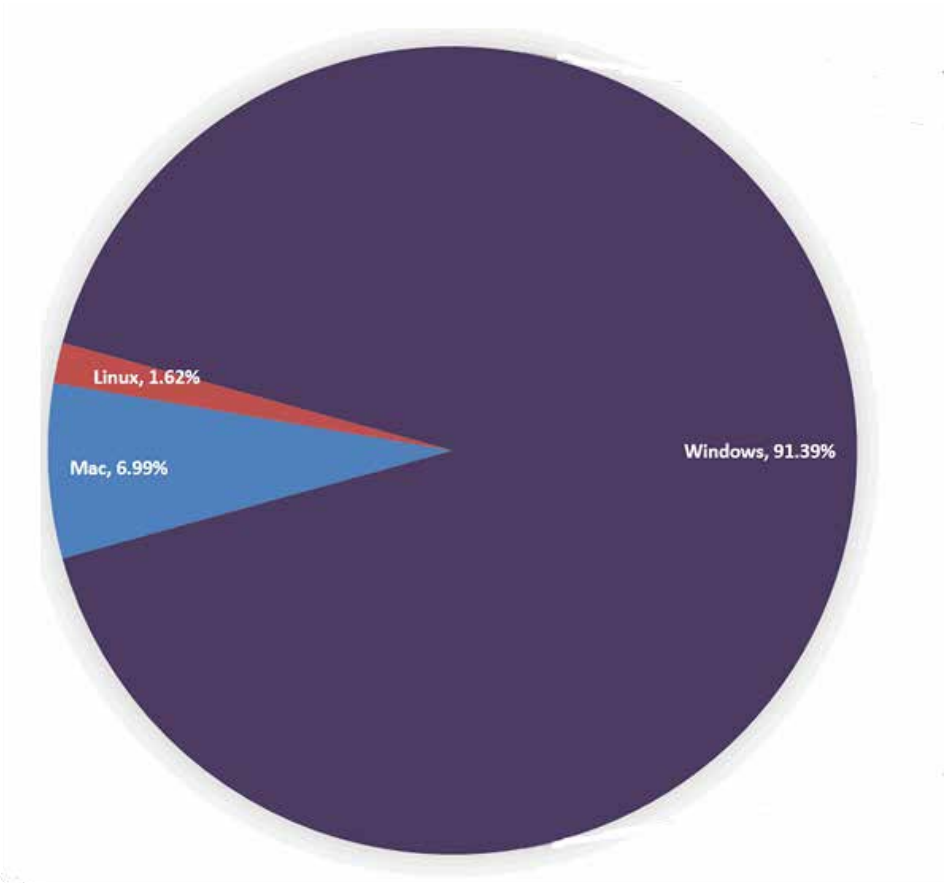

Рис. 13 Доля ринку Windows

Ще одним важливим фактором є сумісність з апаратним забезпеченням. Windows має відмінну підтримку широкого спектру апаратного забезпечення, включаючи як старі, так і нові пристрої. Це дозволяє користувачам оновлювати свої комп’ютери та використовувати нові технології без потреби змінювати операційну систему. Важливо зазначити, що Windows чудово інтегрується з іншими продуктами Microsoft, такими як Office 365, OneDrive, Azure та Xbox. Це створює єдину екосистему, що робить роботу з різними пристроями та програмами більш зручною і ефективною.

Операційна система Windows відома своїм зручним і інтуїтивно зрозумілим інтерфейсом користувача. Вона пропонує знайомий вигляд з вікнами, іконками, меню "Пуск" та панеллю завдань, що робить її простою у використанні як для новачків, так і для досвідчених користувачів. Одним із

ключових аспектів її універсальності є налаштовуваність.

Користувачі Windows можуть легко налаштувати систему відповідно до своїх потреб, від зовнішнього вигляду інтерфейсу до функціональності. Це включає зміни теми, встановлення різноманітних додатків і налаштувань безпеки. Видозмінена і мінімалістична Windows зображена на рис 14.

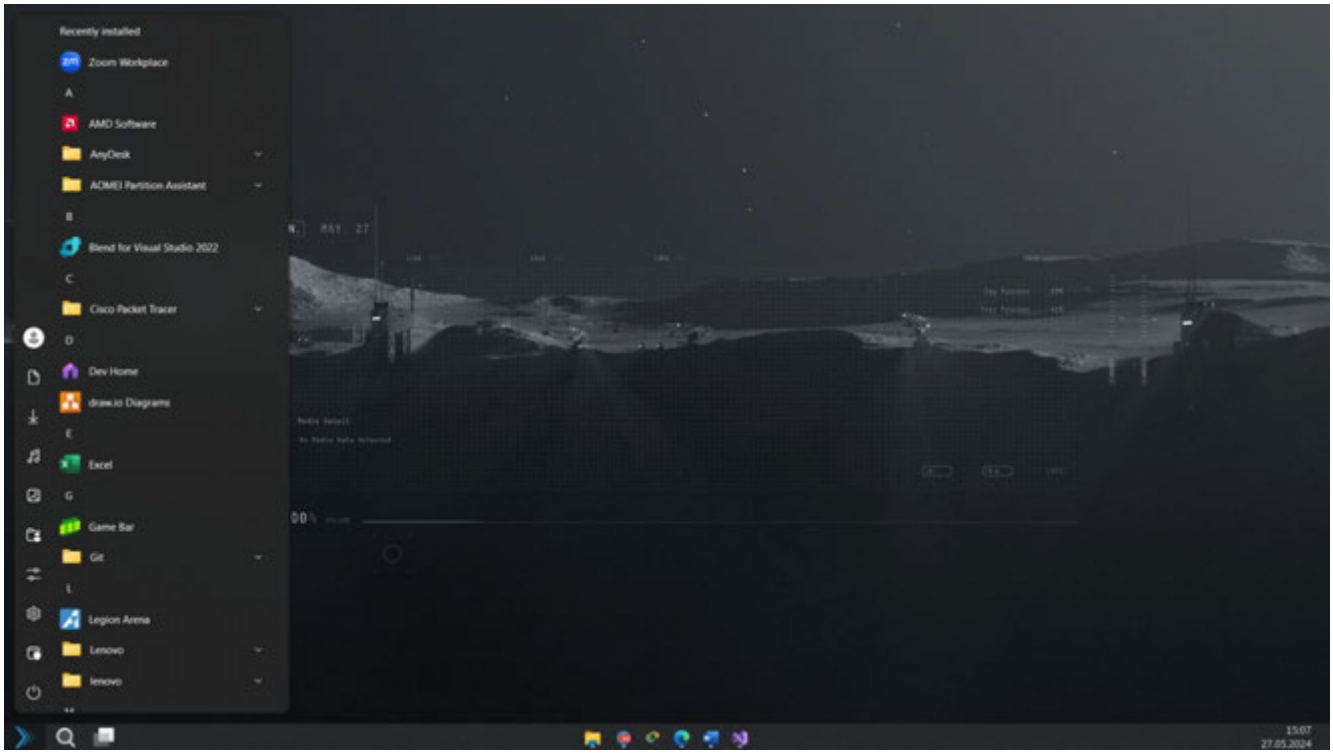

Рис. 14 Видозмінена і мінімалістична Windows

Windows використовується в різних сферах, включаючи освіту, медицину, науку, розваги та багато іншого. Вона підтримує як прості програми для повсякденного використання, так і складні спеціалізовані програми для професійної роботи. Це робить її надзвичайно гнучкою платформою, яка може задовольнити потреби найрізноманітніших користувачів.

Ще одним важливим аспектом універсальності Windows є підтримка багатьох мов. Windows підтримує більше 100 мов, що робить її доступною для користувачів з усього світу. Це важливо для компаній, які працюють на міжнародному рівні, та для користувачів, які хочуть працювати рідною мовою.

Windows також є привабливою платформою для розробників. Вона надає

потужні інструменти, такі як Visual Studio, і підтримку різних мов програмування, включаючи C#, C++, Python, JavaScript та інші. Це дозволяє розробникам створювати широкий спектр програмного забезпечення, від простих додатків до складних систем.

Нарешті, варто зазначити, що Windows підтримує сучасні технології, такі як віртуалізація та хмарні сервіси. Windows Server і Azure пропонують потужні інструменти для побудови та управління віртуальними середовищами та хмарними додатками. Це забезпечує додаткові можливості для підприємств і розробників, роблячи Windows ще більш універсальною платформою.

Операційна система Windows залишається незамінною завдяки своїй популярності, широкій підтримці програмного забезпечення та апаратного забезпечення, зручному інтерфейсу та широким можливостям для налаштування.

Вона підходить як для домашніх користувачів, так і для корпоративного сектору, пропонуючи потужні інструменти для роботи, розваг та розвитку технологій. Завдяки постійним оновленням та інноваціям, Windows продовжує залишатися однією з найпопулярніших і універсальних операційних систем у світі.

3.2 Інструментальні засоби та платформа для розробка ПЗ

Оскільки ціль розробки програмного забезпечення саме під Windows полягає у створенні інтерактивного додатку, було обрано мову програмування C# та платформу WPF як ідеальну комбінацію для досягнення цієї мети. Логотипи технологій зображено на рис 15

Рис. 15 Логотипи C# і WPF

C# є сучасною об'єктно-орієнтованою мовою програмування, розробленою компанією Microsoft як частину платформи .NET. Вона поєднує у собі найкращі риси кількох мов програмування, що робить її потужним, зручним і гнучким інструментом.

Основними характеристиками C# є підтримка об'єктно-орієнтованого підходу, включаючи класи, об'єкти, успадкування, поліморфізм та інкапсуляцію.

Строгість типізації мови допомагає уникнути багатьох помилок на етапі компіляції, а багатий набір класів та методів дозволяє вирішувати широкий спектр завдань у .NET.

Приклад стандартного коду C# з WPF зображено на рис 16

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace SimpleApp_2._0
{
    3 references
    public partial class Window2 : Window
    {
        0 references
        public Window2()
        {
            InitializeComponent();
        }
    }
}
```

Рис. 16 Приклад стандартного коду C# з WPF

C# також пропонує інші цікаві можливості. Завдяки вбудованому збирачу сміття (Garbage Collector), C# зменшує ризики витоків пам'яті та спрощує управління ресурсами.

Ключові слова async і await дозволяють легко писати асинхронний код, що підвищує продуктивність та відгук програм.

Поєднання C# саме з WPF є ідеальним вибором для розробки програмного забезпечення для Windows завдяки тому, що WPF є частиною .NET і бездоганно працює з C#.

WPF (Windows Presentation Foundation) - це підсистема для побудови інтерфейсів користувача у Windows, яка дозволяє створювати сучасні, багаті інтерфейси користувача з використанням декларативного підходу на основі XAML (Extensible Application Markup Language), приклад якого зображено на

рис 17

```
<DockPanel>
    <DockPanel.Background>
        <ImageBrush TileMode="Tile" ImageSource="/main_color.jp
    </DockPanel.Background>
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="40"/>
            <RowDefinition/>
            <RowDefinition/>
            <RowDefinition/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>

        <Border Grid.ColumnSpan="2" Background="DarkGreen">

        </Border>
        <Label Grid.Column="0" Foreground="White" FontFamily=
               VerticalAlignment="Center" HorizontalAlignment="
               Content="Портативні програми" Background="Dark
        <Label Grid.Column="1" Foreground="White" FontFamily=
               VerticalAlignment="Center" HorizontalAlignment="
               Content="Встановлювані програми" Background="D
```

Рис. 17 Приклад коду XAML

Основні характеристики і можливості WPF включають:

- ❖ підтримка складних графічних елементів, анімацій, мультимедійних компонентів та векторної графіки. Це дозволяє створювати візуально привабливі та інтерактивні інтерфейси, що можуть включати багаті мультимедійні елементи та графічні ефекти;
- ❖ потужний механізм прив'язки даних, що дозволяє легко синхронізувати UI та логіку програми. Це спрощує роботу з даними, дозволяючи автоматично оновлювати інтерфейс при зміні даних і навпаки;
- ❖ застосування стилів та шаблонів для гнучкої зміни зовнішнього вигляду елементів. Завдяки цій можливості можна легко змінювати зовнішній вигляд додатків, застосовуючи різні стилі та шаблони, що підвищує

зручність та гнучкість у дизайні UI;

- ❖ розширення функціональності через створення користувацьких контролів та інтеграцію з іншими технологіями .NET. Це дозволяє створювати нові, спеціалізовані елементи управління, що підвищує функціональність додатків та спрощує їх розробку.

З цього можна зробити висновок, що Microsoft сама надала розробникам готовий інструментарій для поєднання backend і frontend у розробці програмного забезпечення на їхній системі. Вибір C# і WPF забезпечує ефективну, надійну і зручну розробку додатків, що повністю відповідають сучасним вимогам до програмного забезпечення для Windows.

3.3 Середовище розробки

У контексті розробки програмного забезпечення для Windows, Visual Studio виступає як ідеальне інтегроване середовище розробки (IDE), що забезпечує ефективну та зручну роботу з мовою програмування C# та платформою WPF. Логотип зображено на рис 18.

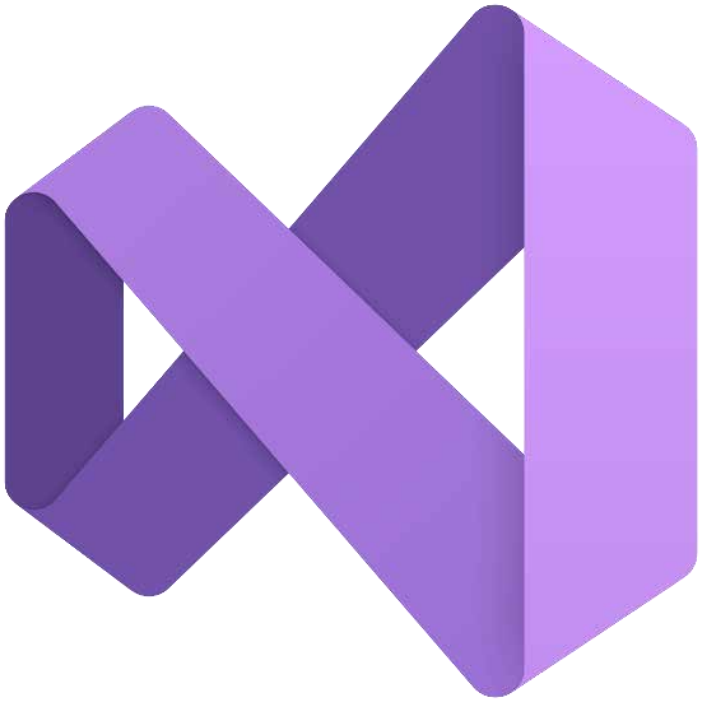

Рис. 18 Логотип Visual Studio

Visual Studio, розроблена Microsoft, має найкращу інтеграцію з платформою .NET, що робить її найбільш підходящим інструментом для створення додатків, орієнтованих на Windows. Вона забезпечує відмінну підтримку для C# та WPF, надаючи розробці всі необхідні інструменти для повного циклу розробки, від написання коду до налагодження та тестування.

Visual Studio пропонує потужний редактор коду, який підтримує підсвітку синтаксису, автодоповнення, рефакторинг та інтегровані інструменти для аналізу коду. Ці функції значно підвищують продуктивність розробки, дозволяючи швидко писати та редагувати код, мінімізуючи кількість помилок. Підсвітка синтаксису та автодоповнення допомагають уникати синтаксичних помилок, тоді як інструменти для рефакторингу забезпечують легке та безпечне внесення змін у код, зберігаючи його чистим та підтримуваним.

Для роботи з WPF, Visual Studio надає зручний візуальний дизайнер, що дозволяє створювати інтерфейси користувача у графічному режимі (зображено на рис 19). Це значно спрощує процес розробки складних інтерфейсів, роблячи його більш інтуїтивно зрозумілим та доступним і це корисно якщо не мати глибоких знань у графічному програмуванні. Візуальний дизайнер дозволяє швидко компонувати елементи інтерфейсу, налаштовувати їх властивості та бачити результат у реальному часі, що знижує ймовірність помилок та прискорює процес розробки.

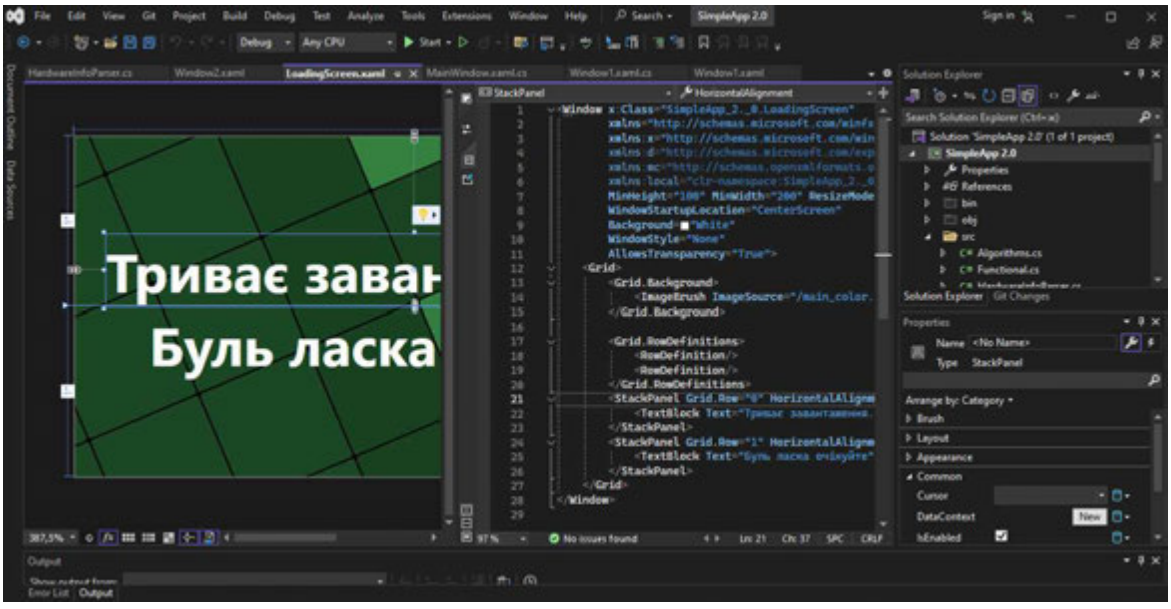

Рис. 19 Візуальний дизайнер Visual Studio

Visual Studio також підтримує інтеграцію з різними системами контролю версій, такими як Git, що дозволяє ефективно керувати версіями коду та відстежувати всі зміни у проекті. Вбудовані інструменти для роботи з Git дозволяють легко виконувати операції коміту, злиття та вирішення конфліктів безпосередньо з середовища розробки.

Ще однією важливою перевагою Visual Studio є велика кількість документації, навчальних матеріалів та прикладів коду. Це дозволяє швидко знаходити відповіді на свої питання, вивчати нові можливості платформи та підвищувати свій професійний рівень. Документація Visual Studio є однією з найбільш детальних та добре структурованих, що робить її надзвичайно корисною.

Переваги Visual Studio для розробки на C# та WPF можна підсумувати наступним чином:

Комплексне середовище: Visual Studio об'єднує всі необхідні інструменти для розробки, налагодження та тестування додатків в одному місці, забезпечуючи безшовний робочий процес

Ефективність розробки: Інтелектуальні функції автодоповнення, підсвітки синтаксису та рефакторингу значно прискорюють написання коду та знижують ймовірність помилок, підвищуючи загальну продуктивність в розробці.

Візуальний дизайн: Інструменти для візуального дизайну інтерфейсів спрощують створення привабливих та функціональних UI, роблячи цей процес інтуїтивно зрозумілим та доступним.

Підтримка та ресурси: Широка документація, навчальні матеріали та активна спільнота підтримки допомагають швидко вирішувати проблеми та освоювати нові можливості.

У підсумку, Visual Studio є потужним і зручним інструментом, який забезпечує ефективну розробку програмного забезпечення на C# та WPF, сприяючи створенню надійних і функціональних додатків для Windows.

3.4 Організаційна структура програмного забезпечення

Як було вже зазначено в першому розділі, програма розбивається, згідно свого функціоналу, на три основні форми: мережеве обладнання, інше обладнання і програми. Додавання ще двох форм до програми, які доповнюють основні форми мережевого обладнання, іншого обладнання і програм, може значно поліпшити користувацький досвід і спростити взаємодію з програмою.

Головна форма (Main Form), що зображена на рис 20 зображена - форма є центральною точкою взаємодії з програмою. Вона вітає користувача, надає короткий огляд можливостей програми та навігаційні елементи для переходу до різних розділів програми. Головна форма може містити швидкий доступ до найважливіших функцій програми, таких як кнопки запуску або пошуку.

Проміжна форма (Intermediate Form), що на рис 21 зображена - форма використовується як перехідна сторінка для користувача, коли він очікує завантаження важливих даних або результатів. Вона може містить підказки для користувача для відображення стану завантаження. Проміжна форма допомагає підтримувати користувача в курсі процесу та зменшує відчуття очікування.

Додавання цих додаткових форм покращить узгодженість та інтуїтивність користувацького інтерфейсу програми. Головна форма надасть користувачеві зручну точку входу, а проміжна форма забезпечить плавний перехід та підтримку користувача під час очікування.

Рис. 20 Головна форма (курсор наведено на 2-гу кнопку)

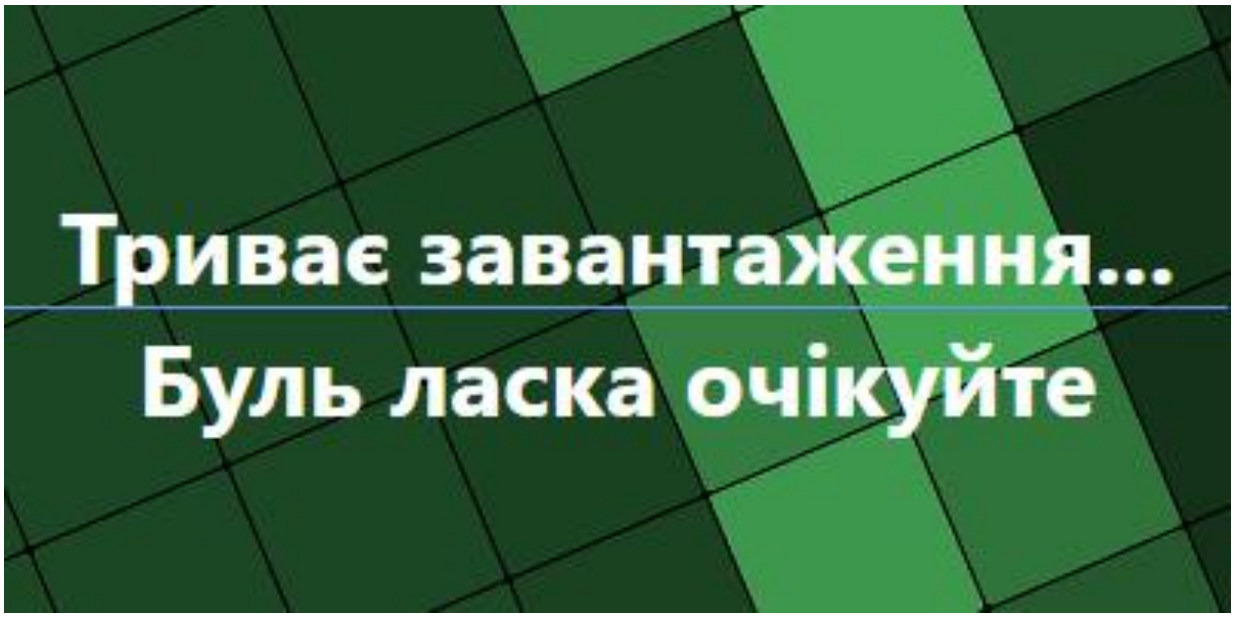

Рис. 21 Проміжна форма

Вибір білого і зеленого кольорів для загального дизайну програми відображає впевнений стиль та сприймається як частина корпоративної ідентичності компанії "Клевер". Цей кольоровий підхід також відповідає сучасним тенденціям у дизайні, де мінімалізм і чистота форм високо цінуються. Принцип Flat Design, що використовується в дизайні, підкреслює простоту та

лаконічність, забезпечуючи зручність як для розробників, так і для користувачів. Приклад дизайну програми зображено на рис 23.

Особливо важливою є спадщина дизайну WPF від Windows Forms, що надає знайомий інтерфейс для користувачів, які працювали з попередніми версіями Windows. Старий дизайн Windows Forms зображено на рис 22.

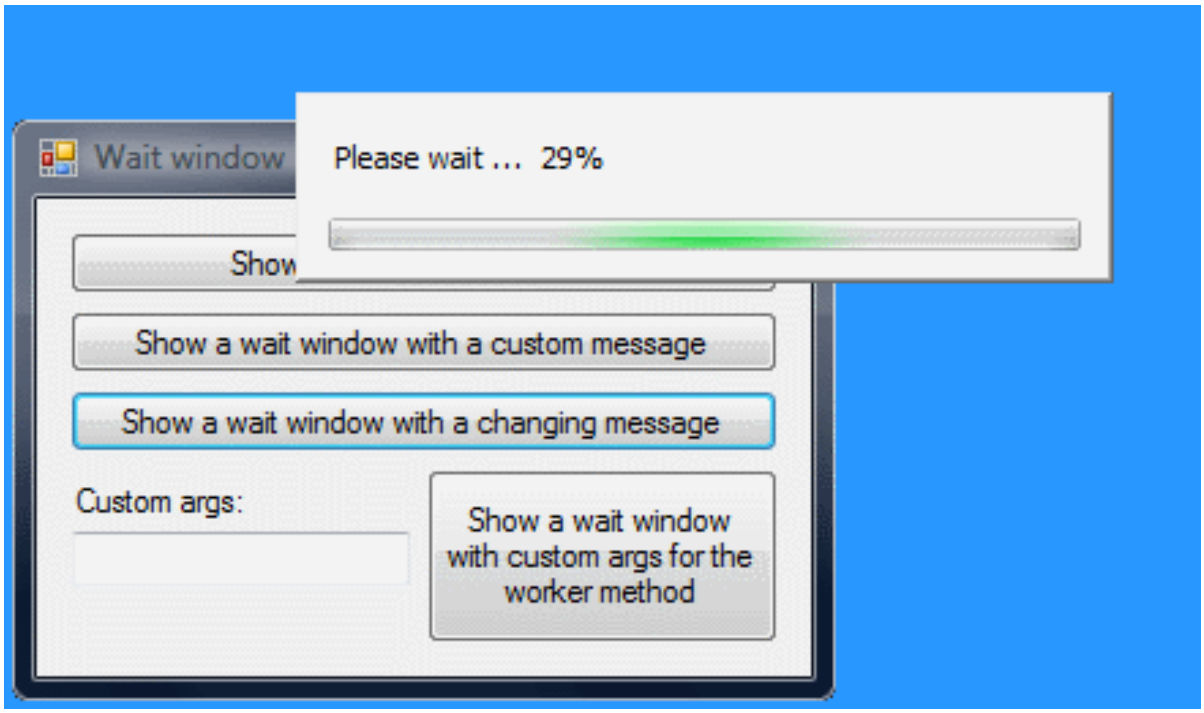

Рис. 22 Старий дизайн Windows Forms

Це створює відчуття зручності та легкості в оволодінні програмою, оскільки вона має відомий і зрозумілий інтерфейс. Такий підхід сприяє підвищенню прийняття програми користувачами та спрощує її використання, що важливо для успіху будь-якого програмного забезпечення.

Рис. 23 Приклад загального дизайну

Рішення з винесенням потрібних функцій для стягування інформації з датчиків в окремий клас (рис 24) та роблення їх асинхронними є дуже розумним кроком (виклик на рис 25). Це дозволяє відокремити логіку отримання даних від основної програми, що полегшує розуміння та підтримку коду. Асинхронність дозволяє програмі не блокувати свій основний потік виконання під час очікування відповіді від датчиків, що робить її більш продуктивною та відзначається більшою відзивчивістю.

Ініціалізація цих функцій під час завантаження програми також додає до швидкодії, оскільки вони готові до використання в момент, коли користувач взаємодіє з програмою. Це особливо корисно на менш потужних комп'ютерах, де ресурси обмежені, і дозволяє покращити загальну продуктивність програми без значного навантаження на систему.

Такий підхід дозволяє оптимізувати використання ресурсів комп'ютера та покращити користувацький досвід, забезпечуючи швидке та ефективне функціонування програмного забезпечення, незалежно від характеристик системи, на якій воно запущене.

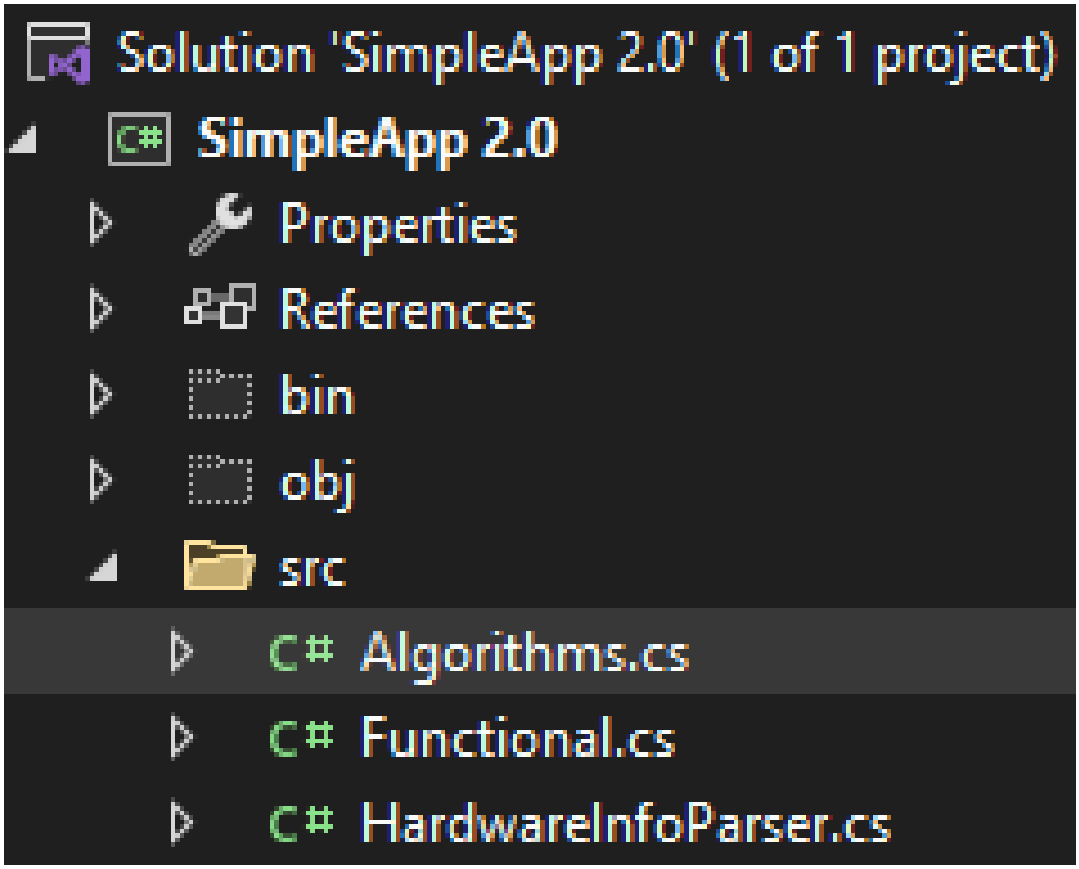

Рис. 24 Структура проекту

```
public async Task initComponents()
{
    Task[] tasks = new Task[] {
        HardwareInfoParser.GetProcessorInfo(),
        HardwareInfoParser.GetGraphicInfo(),
        HardwareInfoParser.GetPhysicalDiskInfo(),
        HardwareInfoParser.GetRAM(),
        HardwareInfoParser.GetDisk(),
        HardwareInfoParser.GetMonitor()
    };
    await Task.WhenAll(tasks);
```

Рис. 25 Виклик асинхроних функцій

Це логічний та зручний підхід до організації програмного інтерфейсу, оскільки він дозволяє зберігати цілісність та структурованість програми, одночасно забезпечуючи зручний користувацький досвід.

Коли кожна форма працює незалежно, але їх запуск відбувається через

головну форму, це спрощує навігацію користувача та допомагає йому зберігати контекст взаємодії з програмою. Користувач може легко повернутися на головну форму, якщо потрібно запустити іншу форму або виконати іншу дію.

Цей підхід також допомагає зберігати логічну структуру програми, оскільки він відображає ієрархію взаємодії між різними частинами програми. Головна форма служить входом до програми, в якій користувач може знайти доступ до різних функцій та опцій.

Загалом, цей підхід сприяє як зручності користувача, так і структурованості програмного коду, роблячи взаємодію з програмою більш зрозумілою та ефективною.

3.5 Висновки до третього розділу

Можна зробити висновок, що використання мови програмування C#, платформи WPF та середовища розробки Visual Studio є оптимальним вибором для створення інтерактивного програмного забезпечення під Windows. Цей вибір забезпечує високу продуктивність, гнучкість, зручність у розробці та підтримку сучасних вимог до програмного забезпечення.

Поєднання C# і WPF дозволяє створювати додатки, які легко розширювати та підтримувати завдяки глибокій інтеграції з платформою .NET, а Visual Studio надає комплексне середовище для розробки, налагодження та тестування додатків, що робить її оптимальним вибором для цього комплексу технологій.

Структура програмного забезпечення, описана у цьому розділі, демонструє чіткий та організований підхід до розробки, з розбиттям функціоналу на окремі форми, кожна з яких виконує конкретні завдання.

Головна форма забезпечує користувачам зрозумілий стартовий пункт для взаємодії з додатком, проміжна форма використовується для очікування відкриття інших сторінок, підвищуючи користувацький досвід, а асинхронні функції винесено в окремому класі для покращення продуктивності програми, особливо на менш потужних комп'ютерах.

4 ВПРОВАДЖЕННЯ СИСТЕМИ ТА ЕКСПЛУАТАЦІЯ

4.1 Вимоги до програмного забезпечення

Для успішного функціонування інтерактивного додатку, розробленого на C# з використанням WPF, необхідно забезпечити відповідність певним вимогам до програмного забезпечення. Ці вимоги охоплюють операційну систему, платформу .NET, графічні драйвери та інші компоненти.

Операційна система (ОС) як було сказано від початку розділу є Windows від компанії Microsoft. Підтримувані версії цієї операційної системи для роботи з додатком:

- Windows 11 (в будь якому вигляді);
- Windows 10 (рекомендується остання версія з усіма оновленнями);
- Windows 8.1(рекомендується остання версія з усіма оновленнями);
- Windows Server 2012 R2 або пізніші версії для серверних застосувань.

Важливо додати, що на старіших версіях Windows, програма може запуститись проте деякі функції будуть некоректно працювати.

Саме 64-бітна версія ОС рекомендована для кращої продуктивності та підтримки більшої кількості оперативної пам’яті.

Програма повністю заснована на .NET Framework**,** а тому важливо мати оновлені версії цього програмного забезпечення для коректної роботи всіх функцій**:**

- мінімальна підтримувана версія: .NET framework 4.7.2;
- рекомендована версія: .NET framework 4.8 або новіша для покращеної продуктивності, безпеки та доступності функцій.

Забезпечення відповідності цим вимогам є критичним для стабільної та ефективної роботи додатку на C# та WPF, розробленого у Visual Studio. Це гарантує, що програмне забезпечення буде працювати плавно, без збоїв, та забезпечить користувачеві найкращий можливий досвід взаємодії.

4.2 Вимоги до апаратного забезпечення

Оскільки програмне забезпечення засновано на стандартних інструментах Windows, як мінімум версії 10, і розроблене за допомогою C# та WPF, воно потребує певного апаратного забезпечення для забезпечення стабільної та ефективної роботи. Нижче наведено вимоги до апаратного забезпечення для такого програмного забезпечення.

Мінімальні вимоги

Процесор (CPU):

- 1 гігагерц (ггц) або швидший процесор з підтримкою pae, nx і sse2;
- бажано багатоядерний процесор для кращої продуктивності.

Оперативна пам’ять (RAM):

- 2 ГБ для 64-бітної версії Windows 10;
- 4 ГБ рекомендовано для більш комфортної роботи з додатками, що використовують WPF.

Відеокарта (GPU):

- графічний пристрій, сумісний з directx 9 або новішою версією з драйвером wddm 1.0;
- рекомендується відеокарта з підтримкою directx 11 для покращеної продуктивності.

Місце на жорсткому диску (HDD/SSD):

- 500 МБ вільного місця на диску для 64-бітної версії Windows 10.

Дисплей:

- екран з роздільною здатністю 800x600 пікселів;
- рекомендується роздільна здатність 1366x768 пікселів або вище для комфортної роботи з інтерфейсами WPF.

Рекомендовані вимоги

Процесор (CPU):

- багатоядерний процесор Intel Core i5 від шостого покоління;
- багатоядерний процесор AMD Razen 3 мінімально Zen2 покоління;

Оперативна пам’ять (RAM):

- 8 ГБ для більш комфортної роботи та можливості запуску інших додатків одночасно.

Відеокарта (GPU):

- відеокарта з підтримкою directx 11 або новішою версією;
- рекомендується також відеокарта з апаратним прискоренням графіки.

Місце на жорсткому диску (HDD/SSD):

- SSD для швидкого завантаження ОС та програми;
- 2 ГБ або більше для зберігання додаткового програмного забезпечення.

Дисплей:

- дисплей з роздільною здатністю 1920x1080 (Full HD) або вище.

Забезпечення відповідності цим вимогам до апаратного забезпечення гарантує стабільну та продуктивну роботу програмного забезпечення, розробленого на базі C# та WPF, і дозволяє користувачам ефективно взаємодіяти з додатком, використовуючи всі його функціональні можливості.

Варто зазначити цікавий факт. Простий факт того, що на комп’ютері вже встановлена Windows 10 чи 11, дає майже стовідсоткову ймовірність коректної роботи програми.

4.3 Склад інсталяційного пакету

Оскільки програма розроблена для адміністраторів, її створено у портативному форматі. Вся програма може бути перенесена в одній директорії або архіві, що значно спрощує її розповсюдження та використання.

Це означає, що адміністратори можуть легко перенести програму на будь-який комп’ютер без необхідності інсталяції, просто розпакувавши архів чи скопіювавши директорію. Такий підхід забезпечує максимальну мобільність і зручність, дозволяючи швидко розгортати програму в різних середовищах.

Відсутність інсталяційного файлу зменшує складність процесу впровадження, адже немає потреби виконувати тривалу процедуру встановлення чи налаштування. Це також мінімізує ризики, пов’язані з правами адміністратора та сумісністю з іншими програмами, які можуть виникнути під час традиційної

інсталяції.

Таким чином, портативний формат програми забезпечує гнучкість, оперативність та легкість використання, що є важливими критеріями для адміністраторів, які часто працюють в динамічних та різноманітних IT-середовищах.

4.4 Висновки до четвертого розділу

Розробка програмного забезпечення для Windows, орієнтованого на адміністраторів, потребує вибору оптимальних інструментів і технологій. В даному випадку обрання мови програмування C# та платформи WPF виявилося ідеальним рішенням.

Завдяки строгій типізації, вбудованому збирачу сміття і підтримці асинхронного програмування, C# дозволяє створювати надійні та ефективні додатки. В свою чергу, WPF надає потужні інструменти для створення сучасних і багатих інтерфейсів користувача, підтримуючи складні графічні елементи, що робить її ідеальною для розробки інтерактивних додатків під Windows.

Інтеграція C# та WPF у середовищі розробки Visual Studio забезпечує розробці усі необхідні інструменти для створення, налагодження та тестування додатків. Visual Studio пропонує потужний редактор коду, що значно підвищує продуктивність розробки. Візуальний дизайнер інтерфейсів WPF у Visual Studio дозволяє легко та інтуїтивно створювати привабливі й функціональні UI, спрощуючи роботу з графічними компонентами.

Особливо важливою особливістю є портативність розробленого програмного забезпечення. Можливість перенесення програми в одній директорії або архіві без необхідності інсталяції значно спрощує її розповсюдження та впровадження. Це дозволяє адміністраторам швидко та безпечно розгортати програму на будь-якому комп’ютері, що значно підвищує оперативність і гнучкість у роботі. Відсутність інсталяційного файлу мінімізує ризики, пов’язані з правами адміністратора та сумісністю з іншими програмами, що можуть виникнути під час традиційної інсталяції.

Крім того, апаратні вимоги до програмного забезпечення забезпечують його стабільну та продуктивну роботу. Рекомендовані параметри апаратного забезпечення включають багатоядерний процесор, достатню кількість оперативної пам'яті, сучасну відеокарту та швидкий SSD, що гарантує високу продуктивність і зручність використання. Забезпечення відповідності цим вимогам дозволяє користувачам максимально ефективно використовувати програму, незалежно від середовища її розгортання.

Загалом, обрання C#, WPF та Visual Studio для розробки програмного забезпечення під Windows забезпечує високу продуктивність, гнучкість, зручність у розробці та підтримці сучасних вимог до програмного забезпечення. Портативність і оптимальні апаратні вимоги роблять цей додаток ідеальним для адміністраторів, що працюють у різних IT-середовищах, забезпечуючи швидке і безпечне розгортання та використання програмного забезпечення.

ВИСНОВКИ

У результаті проведеної роботи було розроблено програмне забезпечення з метою оптимізації роботи IT-адміністраторів на платформі Windows. Для

досягнення цієї мети було використано мову програмування C# та платформу WPF у середовищі розробки Visual Studio.

Процес розробки розділився на чотири основних етапи, які включали в себе аналіз предметної області, проектування програмного забезпечення, розробку самого програмного продукту та впровадження системи.

Перший етап включав в себе детальне дослідження потреб користувачів та аналіз існуючих рішень. На основі цього були сформульовані вимоги до програмного забезпечення та проведено моделювання предметної області. Цей етап дозволив чітко визначити завдання проекту та його область застосування.

Другий етап був спрямований на проектування програмного забезпечення. Була створена логічна модель даних, розроблені діаграми класів, пакетів та компонентів, що визначили архітектуру програми та її компоненти. Завершивши всі чотири етапи проекту, можемо зазначити, що ця робота становить значний внесок у сферу оптимізації роботи IT-адміністраторів на платформі Windows. Адже, розроблене програмне забезпечення не лише відповідає поставленим вимогам, але й відображає глибоке розуміння предметної області та потреб користувачів.

Проведений аналіз предметної області та існуючих рішень дозволив чітко визначити проблемні питання, які мали бути вирішені. Це відкривало шлях до формулювання конкретних вимог до програмного забезпечення та планування подальших етапів роботи.

У процесі проектування програмного забезпечення було приділено особливу увагу розробці логічної моделі даних та структури програми. Це забезпечує якісну базу для подальшого розвитку та підтримки системи.

Реалізація програмного забезпечення відбувалася у відповідності з усіма сучасними стандартами програмування, що дозволяє забезпечити високу якість продукту та його легку модифікацію у майбутньому.

Впровадження системи та підготовка до експлуатації відбувалися за чіткою стратегією, що дозволило уникнути непередбачених проблем та максимально ефективно впровадити розроблене програмне забезпечення.

У підсумку, цей проект є важливим кроком у напрямку створення високоефективного інструменту для оптимізації роботи IT-адміністраторів на платформі Windows. Його успіх підтверджується не лише технічними характеристиками, але й глибоким розумінням потреб користувачів та вмінням відповідати їм.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ І ЛІТЕРАТУРИ

1) Посилання на офіційний сайт AIDA64: https://www.aida64.com/.

2) Посилання на офіційний сайт виробника програмного забезпечення CPUid: https://www.cpuid.com/.
3) Книга "UML. Посібник користувача" авторів Г. Буч, А. Якобсон, Дж. Рамбо.
4) Книга "UML 2. Руководство користувача" авторів Дж. Рамбо, Г. Буч, А. Якобсон.
5) Книга "Використання UML для моделювання бізнес-процесів" авторів А. Якобсон, І. Якобсон.
6) Microsoft Virtual Academy - курси з мови XAML.
Доступно на https://learn.microsoft.com/en-us/training/.
7) Microsoft Docs:
   - Курси з мови C#: https://learn.microsoft.com/en-us/dotnet/csharp/.
   - Курси з WPF: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/?view=netdesktop-8.0.
   - Курси з Visual Studio: https://learn.microsoft.com/en-us/visualstudio/windows/?view=vs-2022.
8) Microsoft Learn:
   - Посібник з мови C#: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/tutorials/.
   - Посібник з WPF: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/?view=netdesktop-8.0.
   - Посібник з Visual Studio: https://visualstudio.microsoft.com/vs/getting-started/.
9) Канал Microsoft Windows на YouTube: https://www.youtube.com/user/Microsoft;
10) Книга "Programming C# 10th Edition" автора Mark Hendrickson.

11) Книга "WPF with .NET 6 and .NET Core 3.1: Beginner's Guide" автора Adam Freeman.
12) Книга "Head First Programming with Visual Studio 2022" автора Paul Butcher.