Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка системи візуального моніторингу показників комп'ютерного обладнання з використанням JavaFX

Виконав: студент 4 курсу, групи СП-42
спеціальності 121 «Інженерія програмного забезпечення»
(шифр і назва спеціальності)

(підпис) Муц Н.Р. (прізвище та ініціали)

Керівник (підпис) Мудрик. І.Я. (прізвище та ініціали)
Нормоконтроль (підпис) Стоянов Ю.М. (прізвище та ініціали)
Завідувач кафедри (підпис) Петрик М.Р. (прізвище та ініціали)
Рецензент (підпис) (прізвище та ініціали)

Тернопіль 2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет Інформаційних систем та програмної інженерії
(повна назва факультету)

Кафедра Програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ _____
(підпис) (прізвище та ініціали)
« » 20___ р.

З А В Д А Н Н Я
КВАЛІФІКАЦІЙНОЇ РОБОТИ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту Муцу Назару Романовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка системи візуального моніторингу показників комп'ютерного обладнання з використанням JavaFX

Керівник роботи Мудрик Іван Ярославович, доктор філософії, старший викладач кафедри ПІ
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « » 20___ року № .
2. Термін подання студентом завершеної роботи
3. Вихідні дані до роботи Методичні вказівки; Технічна документація для JavaFX, OSHI, Java.

4. Зміст роботи (перелік питань, які потрібно розробити)
Огляд та порівняльний аналіз конкурентів, обгрунтування вибору напрямку дослідження, вибір методології розробки, формування вимог до системи, проєктування відношень між акторами та прецедентами, проєктування головних класів, визначення архітектури системи, проєктування структури ПЗ, тестування та верифікація вимог, надзвичайні ситуації, викликані пожежами, вибухами, техногенними та природними причинами, оцінка технологічного процесу, обладнання, щодо умов електробезпеки, безпеки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Діаграма класів, діаграма варіантів використання, діаграма послідовностей,

Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Лазарюк В. В., к.т.н., доц., доц. каф. МТ		
Нормоконтроль	Стоянов Ю. М., к.т.н.		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Вибір та затвердження теми		
2.	Аналіз предметної області та технічного завдання		
3.	Вибір методології розробки		
4.	Формування вимог до системи		
5.	Визначення архітектури, проєктування голосвних класів		
6.	Конструювання основного функціоналу ПЗ		
7.	Тестування та верифікація вимог		
8.	Безпека життєдіяльності, основи охорони праці		
9.	Підготовка презентації та доповіді		
10.	Нормоконтроль		
11.	Перевірка на плагіат		
12.	Попередній захист кваліфікаційної роботи		
13.	Захист кваліфікаційної роботи		

Студент _____ Муц Н.Р.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Мудрик І.Я.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Атестаційна робота бакалавра. Тернопільський національний технічний університет імені Івана Пулюя, кафедра програмної інженерії, спеціальність 121 «Інженерія програмного забезпечення». ТНТУ, 2025. Сторінок 70, таблиць 0 ,рисунків 30 , презентація.

Тема: Розробка системи візуального моніторингу показників комп'ютерного обладнання з використанням JavaFX

У даній атестаційній роботі бакалавра розроблено систему для візуального моніторингу стану комп'ютерного обладнання в режимі реального часу. Проєкт реалізований із використанням JavaFX — сучасного фреймворку для створення графічних інтерфейсів у середовищі Java.

Система надає користувачеві зручний інтерфейс для перегляду ключових апаратних показників, зокрема: завантаження процесора, обсягу доступної оперативної пам'яті, температури компонентів та стану накопичувачів. Крім того, інтегровано модуль штучного інтелекту, що дозволяє не лише аналізувати зібрані дані, але й відповідати на запитання користувача. У розробці використано багаторівневу архітектуру, реалізовано збір даних через низькорівневі API, забезпечено масштабованість і стабільність системи. Особливу увагу приділено візуалізації: всі графіки, гістограми та діаграми оновлюються в реальному часі, забезпечуючи зрозуміле представлення технічної інформації.

Система може бути корисною для IT-спеціалістів, системних адміністраторів, ентузіастів та користувачів, які хочуть контролювати стан свого комп'ютера у зручній формі. Також робота демонструє інтеграцію елементів штучного інтелекту у прикладні десктопні застосунки.

Ключові слова: моніторинг системи, JavaFX, штучний інтелект, апаратні ресурси, візуалізація, аналіз даних, системна аналітика.

ABSTRACT

Bachelor's certification work. Ternopil National Technical University named after Ivan Puluj, Department of Software Engineering, specialty 121 “Software Engineering”. TNTU, 2025. Pages 70, tables 0 , figures 30, presentation

Theme: Development of a system for visual monitoring of computer equipment indicators using JavaFX

In this bachelor`s certification work, a system for visual monitoring of computer equipment in real time has been developed. The project was implemented using JavaFX, a modern framework for creating graphical interfaces in the Java environment.

The system provides a user-friendly interface for viewing key hardware indicators, including CPU utilization, available RAM, component temperatures, and the status of drives. In addition, an artificial intelligence module is integrated, which allows not only analyzing the collected data but also answering user questions. The development used a multi-level architecture, implemented data collection via low-level APIs, and ensured scalability and stability of the system. Particular attention is paid to visualization: all graphs, histograms, and charts are updated in real time, providing a clear presentation of technical information.

The system can be useful for IT professionals, system administrators, enthusiasts, and users who want to monitor their computer in a convenient way. The work also demonstrates the integration of artificial intelligence elements into desktop applications.

Keywords: system monitoring, JavaFX, artificial intelligence, hardware resources, visualization, data analysis, system analytics.

ЗМІСТ

ВСТУП

У сучасному світі стабільна та ефективна робота комп’ютерних систем є критично важливою як для персонального використання, так і для бізнесу. Вчасне виявлення перевантаження ресурсів, перегріву, нестачі оперативної пам’яті або інших відхилень у роботі системи дозволяє уникнути збоїв, втрати даних та скорочення терміну експлуатації обладнання. Саме тому інструменти для моніторингу комп’ютерного стану набувають все більшого значення.

Однак більшість існуючих засобів моніторингу є або надто технічними, або не забезпечують достатньо інформативної візуалізації, що ускладнює їх використання для широкого кола користувачів. У зв’язку з цим постає потреба у створенні програмного продукту, який би об'єднував зрозумілий інтерфейс, гнучкі можливості аналізу та сучасні засоби візуалізації даних.

Метою цієї роботи є розробка програмного комплексу, що дозволяє здійснювати візуальний моніторинг основних показників комп’ютерного обладнання у режимі реального часу. Основою реалізації інтерфейсу виступає JavaFX — гнучка та потужна платформа для створення графічних застосунків мовою Java.

Особливістю розробленої системи є інтеграція модуля штучного інтелекту, який дозволяє не лише збирати і візуалізувати дані, але й здійснювати первинний аналіз ситуації та відповідати на запитання користувача у природній мові. Такий підхід поєднує переваги класичного моніторингу з інтелектуальними можливостями, що забезпечує зручність і розширює функціональність системи.

Актуальність теми обумовлена зростаючими вимогами до надійності комп’ютерних систем, розвитком технологій штучного інтелекту та необхідністю створення доступних інструментів для моніторингу навіть для неспеціалістів. Розроблений програмний продукт може використовуватись як у побутових умовах, так і в корпоративному середовищі.

РОЗДІЛ 1. ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис концепції розробки

Сучасні комп'ютерні системи, від персональних робочих станцій до потужних ігрових ПК, стають все більш складними та продуктивними. Водночас, ефективне управління їхньою роботою та забезпечення стабільності в деяких випадках вимагає моніторингу показників апаратного забезпечення. Без належного контролю, користувачі можуть стикатися з проблемами зниження продуктивності, перегріву компонентів, несподіваних збоїв та скорочення терміну служби обладнання. Існуючі засоби моніторингу часто є або занадто складними для звичайного користувача, або надають недостатньо вичерпну інформацію, або ж є пропрієтарними рішеннями, що обмежують їх використання на різних платформах. Актуальність розробки програмного комплексу "SystemScope" обумовлена зростаючою потребою користувачів у простому та доступному інструменті для візуального моніторингу стану комп'ютерних систем. Особливо це стосується випадків, де важливо відстежувати поведінку обладнання під навантаженням – наприклад, під час ігор, обробки графіки чи виконання ресурсоємних обчислень. Основна ціль цієї дипломної роботи полягає в проєктуванні, розробці та реалізації кросплатформного програмного комплексу "SystemScope", який може надаи користувачеві інформацію про апаратне забезпечення комп'ютера, візуалізувати динаміку показників та надавати інтелектуальну допомогу для аналізу цих даних.

Результатом розробки стане програмний комплекс, який не лише надасть користувачам контроль над своїм апаратним забезпеченням, а й спростить діагностику потенційних проблем, оптимізацію продуктивності та подовжить термін служби компонентів комп'ютера завдяки своєчасному інформуванню та інтелектуальному аналізу даних.

1.2 Огляд конкурентів

На сучасному ринку програмного забезпечення представлено велику кількість систем для моніторингу ресурсів комп’ютера. Вони розрізняються за функціональністю, інтерфейсом, можливістю аналізу та візуалізації даних. У цьому розділі розглянуто найпоширеніші рішення, які можна вважати аналогами або конкурентами розроблюваної системи.

Одним із лідерів є AIDA64 — потужний інструмент для комплексної діагностики і моніторингу апаратного забезпечення. Програма надає детальну інформацію про процесор, материнську плату, оперативну пам’ять, відеокарту, накопичувачі та інші компоненти. Вона підтримує моніторинг температур, напруг, швидкостей обертання вентиляторів, а також дозволяє створювати звіти та вести логування.

Рисунок 1.1 – Логотип компанії конкурента AIDA64

Ще одним популярним інструментом є HWMonitor. Це безкоштовна утиліта орієнтована на відображення температур, напруг і швидкості обертання вентиляторів. Програма має простий інтерфейс, не вимагає встановлення і споживає мінімальні ресурси. Проте функціональність обмежується лише відображенням поточних значень, без підтримки графіківта аналітики.

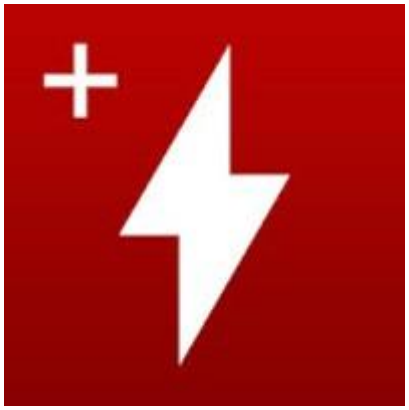

Рисунок 1.2 – Логотип компанії конкурента HWMonitor

Варто також відзначити Open Hardware Monitor — програму з відкритим кодом, яка дозволяє переглядати інформацію про температуру, використання процесора, напругу та інші параметри. Вона підтримує логування та може бути розширена за допомогою сторонніх розробників. Однак інтерфейс користувача досить складний і технічний, що ускладнює її використання широкою аудиторією.

Рисунок 1.3 – Логотип компанії конкурента Open Hardware Monitor

Окремо можна згадати MSI Afterburner, який здебільшого використовується геймерами для моніторингу та розгону графічних процесорів відеокарт. Він має зручний оверлей для виведення інформації під час ігрових сесій, проте охоплює переважно тільки GPU та не підтримує аналіз інших системних компонентів.

Рисунок 1.4 – Логотип компанії конкурента MSI Afterburner

Узагальнюючи, більшість існуючих рішень мають певні обмеження а саме: гомерційність або обмежена функціональність безкоштовних версій, відсутність гнучкої візуалізації та неможливість розширення або інтеграції зі штучним інтелектом.

Розроблювана система має на меті об'єднати переваги наявних рішень, водночас усунувши їхні недоліки. Застосування JavaFX дозволяє створити інтуїтивно зрозумілий та гнучкий інтерфейс користувача, а вбудований модуль штучного інтелекту — забезпечити новий рівень взаємодії з користувачем завдяки можливості формулювати запити природною мовою.

1.3 Обгрунтування вибору напрямку дослідження

Для реалізації функціоналу системи моніторингу та модуля штучного інтелекту була обрана мова програмування Java. Цей вибір обумовлений перевагами кросплатформеності, тому що принцип "Write Once, Run Anywhere" дозволяє розробленому застосунку працювати на різних операційних системах без значних змін у коді. Це є важливою умовою для системи моніторингу, яка має бути

доступною широкому колу користувачів незалежно від їхньої програмної платформи. Завдяки використанню віртуальної машини Java (JVM) Java додатки демонструють високу продуктивність, що також важливим для моніторингу в режимі реального часу, де необхідна швидка обробка та оновлення даних. Також важливо що Java є строго типізованою мовою, що сприяє написанню менш схильного до помилок коду. Механізм обробки винятків та автоматичне керування пам'яттю підвищують стабільність розроблених рішень та знижують ризик виникнення непередбачуваних помилок.

Для розробки графічного інтерфейсу користувача системи моніторингу було обрано сучасний фреймворк JavaFX. Цей вибір ґрунтується перевагами сучасного та потужного GUI-фреймворку JavaFX який є наступником Swing і AWT, та пропонує ширші можливості для створення високоінтерактивних інтерфейсів. JavaFX використовує апаратне прискорення для рендеригнгу графіки, що забезпечує високу продуктивність інтерфейсу. Також за допомогою використання FXML, який дозволяє розділити логіку застосунку від його візуального представлення, було спрощено процес розробки дизайну інтерфейсу.

Для ефективного та кросплатформного збору низькорівневих показників комп'ютерного обладнання була обрана бібліотека OSHI (Operating System Hardware Information). OSHI надає Java API для отримання системної інформації, відокремлюючись від особливостей конкретної операційної системи, що значно спрощує розробку, оскільки в більшості випадків не потрібно писати окремий код для кожної платформи. Також ця бібліотека дозволяє отримати доступ до широкого списку апаратних і системних показників, що є необхідним для комплексного моніторингу.

РОЗДІЛ 2. АНАЛІЗ ВИМОГ ДО СИСТЕМИ

2.1 Функціональні вимоги

Розроблений програмний комплекс “SystemScope” для візуального моніторингу показників комп'ютерного обладнання має виконувати наведений перелік основних функцій, спрямованих на надання користувачеві детальної інформації про стан апаратних ресурсів комп'ютера. Основні функціональні вимоги включають:

1) Моніторинг апаратних ресурсів: Система повинна зчитувати з сенсорів та відображати інформацію про показники основних апаратних ресурсів персонального комп’ютера в реальному часі, а саме центрального процесора (CPU), графічного процесора (GPU), оперативної пам’яті (RAM), дискового простору (HDD/SSD), температури компонентів та навантаження системи.
2) Візуалізація даних: Система повинна надавати зручний інтерфейс для графічної візуалізації зібраних даних про графіки та діаграми для подальшого відображення динаміки змін параметрів в режимі реального часу.
3) Аналіз і рекомендації: Вбудована система штучного інтелекту повинна давати можливість аналізувати зібрані дані та надавати рекомендації з оптимізації роботи системи, також має бути можливість задавати питання системі щодо її стану та отримувати текстові відповіді.
4) Оповіщення про критичні стани: у разі досягнення критичних значень показників (наприклад, висока температура ЦП або перевантаження оперативної пам’яті), система повинна автоматично сповіщати користувача
5) Збір і збереження даних: Система повинна мати можливість збереження зібраних даних для подальшого аналізу, створення звітів або для порівняння результатів на різних етапах.

6) Інтерфейс налаштувань: Користувач повинен мати можливість налаштувати параметри моніторингу, вибираючи компоненти комп’ютера для відстеження.
7) Тест продуктивності: Система повинна мати вбудовану функцію для проведення бенчмарку ПК для обраного процесу який буде навантажувати систему, що дозволяє оцінити загальну швидкодію системи під час значних навантажень.

Функціональні вимоги можна ілюструвати в вигляді діаграми варіантів використання (Use Case Diagram), ця діаграма розробленої системи моніторингу визначає функціональні вимоги та взаємодію користувача з програмним комплексом. Вона розроблена у вигляді діаграми, яка ілюструє основні дії, які є доступними для користувача. Для проєктування моделі було обрано програмне забезпечення IBM Rational Software Architect. На рисунку 2.1 наведено діаграму варіантів використання системи SystemScope.

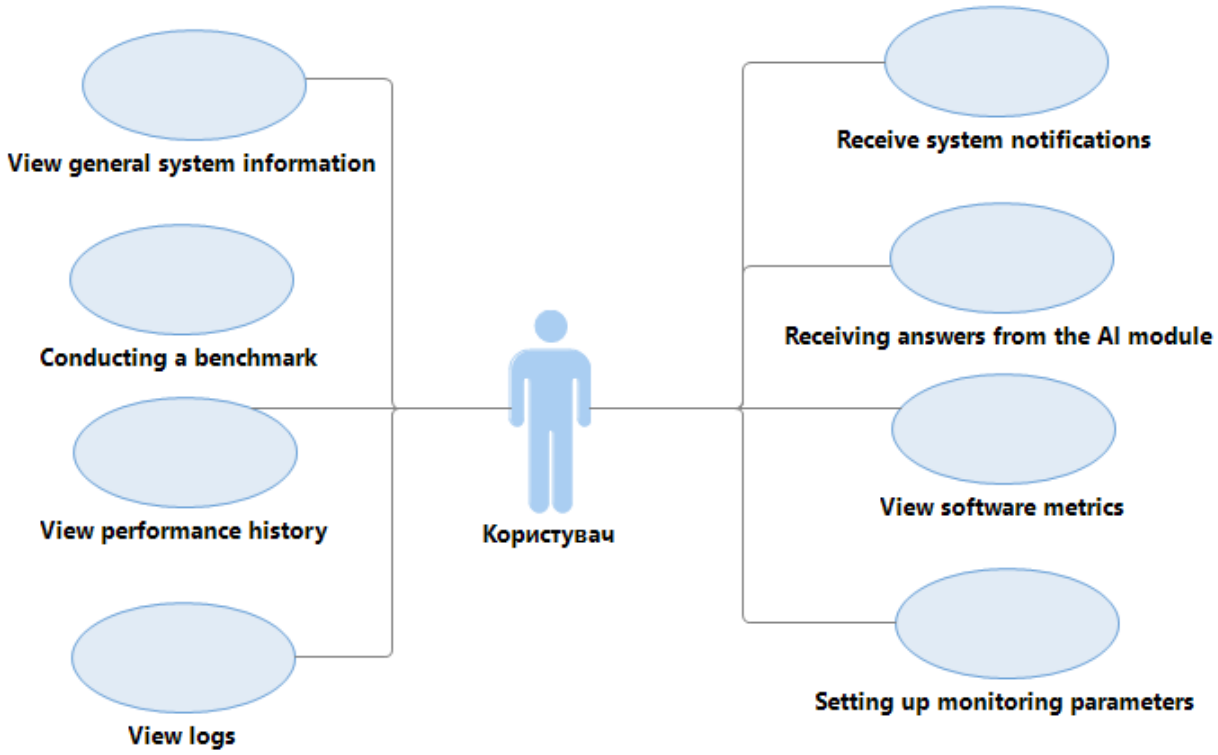

Рисунок 2.1 – Діаграма варіантів використання

Основним актором розробленої системи моніторингу є Користувач, який взаємодіє з її функціональними можливостями. Ключові варіанти використання включають:

Перегляд показників (View Software Metrics): Дозволяє користувачеві переглядати актуальні дані про апаратне обладнання а саме завантаження процесора, обсяг оперативної пам'яті, обсяг використаної оперативної пам’яті, показники відеокарт різних виробників з підтримкою декількох відеокарт в комп’ютері, стан накопичувачів, кількість обертів системи охолодження, температурні показники програмного забезпечення та стан заряду батареї.

- Основний потік подій: Користувач запускає додаток → Система автоматично збирає необхідні доступні дані про апаратне забезпечення комп’ютера → Дані відображаються у відповідних секціях графічного інтерфейсу.

Отримання відповідей від AI-модуля (Receiving answers from the AI module): Надає користувачеві можливість задавати питання системі щодо зібраних показників або загального стану обладнання та отримувати обгрунтовані відповіді з поясненнями та порадами.

- Основний потік подій: Користувач переходить до модуля чату → Вводить необхідний текстовий запит → Система відправляє запит до інтегрованого модуля штучного інтелекту → AI обробляє запит та формує відповідь → Система відображає відповідь AI в інтерфейсі чату.

Перегляд історії показників апаратного обладнання (View Software Metrics): Користувач може переглядати графіки які відображають зміни показників апаратного забезпечення центрального процесору та графічного відеоядра за певний період часу, що дозволяє відстежувати динаміку температур та навантаження на обладнання і виявляти високі показники комплектуючих в режимі реального часу. Користувач може отримувати доступ до графіків в будь який момент і вони динамічно оновлюються навіть коли вікно з графіками є неактивним або прихованим.

Отримання системних сповіщень від системи моніторингу(Receive System

Notification): Система автоматично генерує та відображає сповіщення про критичні події в системі або перевищення порогових значень комп’ютерного обладнання.

Перегляд загальної системної інформації (View General System Infromation): Надає користувачеві можливість ознайомитися з основними характеристиками поточної операційної системи,персонального комп'ютера та його апаратним забезпеченням.

Налаштування параметрів моніторингу (Setting up monitoring parameters): Користувач може змінювати інтервал оновлення даних, встановлювати порогові значення для сповіщень, а також налаштовувати інші параметри функціонування системи.

Перегляд логів (View Logs): Надає доступ до журналу подій який записує хід проведення бенчмарку та його фінальні показники, відображає помилки системи для діагностики та аналізу, відображають кінцевий висновок за допомогою штучного інтелекту.

Проведення бенчмарку (Conducting a benchmark): Дозволяє проводити тестові вимірювання продуктивності певних компонентів системи під час навантаження в обраному процесі (більше використовується для ігор).

2.2 Нефункціональні вимоги

Окрім того, що саме має робити система не менш важливими є і нефункціональні вимоги. Це ті якості програми, які не є її прямою функцією, але які досить сильно впливають на те, наскільки зручною, надійною та швидкою вона буде для кінцевого користувача. Ці вимоги визначають, як система працює: її продуктивність, безпечність, здатність гнучко масштабуватися, простоту у використанні та підтримці, а також сумісність з різними необхідними середовищами. Для виконання проекту мною було поставлено такі нефункціональні вимоги:

1) Продуктивність: Система повинна працювати з мінімальним впливом на продуктивність ПК, щоб не перевантажувати ресурси. Також моніторинг повинен здіснювати в режимі реального часу без значної затримки в оновленні даних.
2) Кросплатформеність: програмне рішення має бути сумісне з основними операційними системами (Windows, Linux, MacOS) та має бути адаптоване до специфіки кожної з платформ.
3) Інтуїтивно зрозумілий інтерфейс: Інтерфейс користувача який повинен бути простим та зрозумілим, щоб навіть недосвічений користувач міг досить швидко розібратись в інтерфейсі системи.
4) Масштабованість: Система має бути спроектована так, щоб забезпечити можливість розширення функціоналу в майбутньому.
5) Безпека та конфіденційність: Програмне рішення не повинне збирати або зберігати особисті дані користувача.

2.3 Вимоги до інтерфейсу

Створюючи програмне рішенн "SystemScope", було поставлено за мету, щоб програма була максимально зручною та зрозумілою для користувача. Це означає, що інтерфейс мав бути не просто функціональним, а й приємним у роботі, адже саме через нього людина взаємодіє з усіма можливостями моніторингу. Для цього було обрано технологію JavaFX. Вона дає чудові інструменти для реалізації всіх потрібних елементів: від звичних кнопок та таблиць до складних динамічних графіків, які показують, як змінюються показники персонального комп'ютера. Також важливо щоб користувач завжди бачив актуальну інформацію. Тому однією з головних вимог стала можливість динамічного оновлення даних. Також має бути передбачено, що користувач зможе сам обирати, які показники відображатимуться у вікні бенчмарку.

2.4 Підсумки аналізу вимог

Отже, функціональність системи повинна мати широкі можливості для моніторингу апаратного обладнання комп'ютера, їх візуалізації, а також надання рекомендацій на основі аналізу даних. Бенчмарк для тестування продуктивності також є важливим доповненням, що дозволить користувачу оцінювати роботу своєї системи під обраним процесом який буде навантажувати систему. Інтерфейс користувача має бути інтуїтивно зрозумілим, гнучким і адаптивним до різних потреб користувача. Також мають бути реалізовані можливості налаштування відображення даних. Система повинна працювати на різних операційних системах (Windows, Linux, MacOS), та забезпечувати коректну роботу на обраній операційній системі.

РОЗДІЛ 3. РОЗРОБКА МОДЕЛІ ТА ПРОГРАМНОГО КОМПЛЕКСУ

3.1 Розробка моделі предметної області

Модель предметної області відображає основні сутності та їх взаємозв'язки, з якими взаємодіє розроблена система візуального моніторингу комп'ютерного обладнання “SystemScope”. Вона охоплює апаратні компоненти, їхні функціональні показники, а також елементи взаємодії з користувачем та інтелектуального аналізу.

Основними сутностями предметної області є:

- Комп’ютер (PC): Центральний об'єкт моніторингу, що об'єднує всі апаратні компоненти.
- Процесор (CPU): Характеризується показниками завантаження, температури, тактової частоти та кількістю фізичних та логічних ядер, тобто ядра та їх потоки.
- Оперативна пам’ять (RAM): Включає параметри загального обсягу, використовуваного та доступного обсягу.
- Відеокарта (GPU): Представлена різними типами (NVIDIA, AMD, Intel) з показниками завантаження, температури та обсягу відеопам'яті.
- Накопичувач (HDD, SSD): Відображає обсяг пам'яті, використаний простір та загальний стан.

Взаємозв'язки між сутностями визначаються ієрархією комп'ютерного обладнання: Персональний комп'ютер містить центральний процесор, оперативну пам'ять, одну або декілька відеокарт та накопичувачів, дискретної відеокарти може не бути, в такому випадку повинне бути інтегроване графічне ядро в центральний процесор. Кожен з цих компонентів генерує відповідні показники для діагностики, які є основою для візуалізації даних та аналізу навантаження загалом на систему.

3.2 Діаграма послідовностей

Діаграма послідовностей є UML-діаграмою, що відображає взаємодію об'єктів у системі в хронологічному порядку для виконання певного варіанту використання. Для деталізації ключових функціональних можливостей програмного комплексу SystemScope, на рисунку 3.1 представлена діаграма послідовностей для варіанту використання "Перегляд показників апаратного забезпечення".

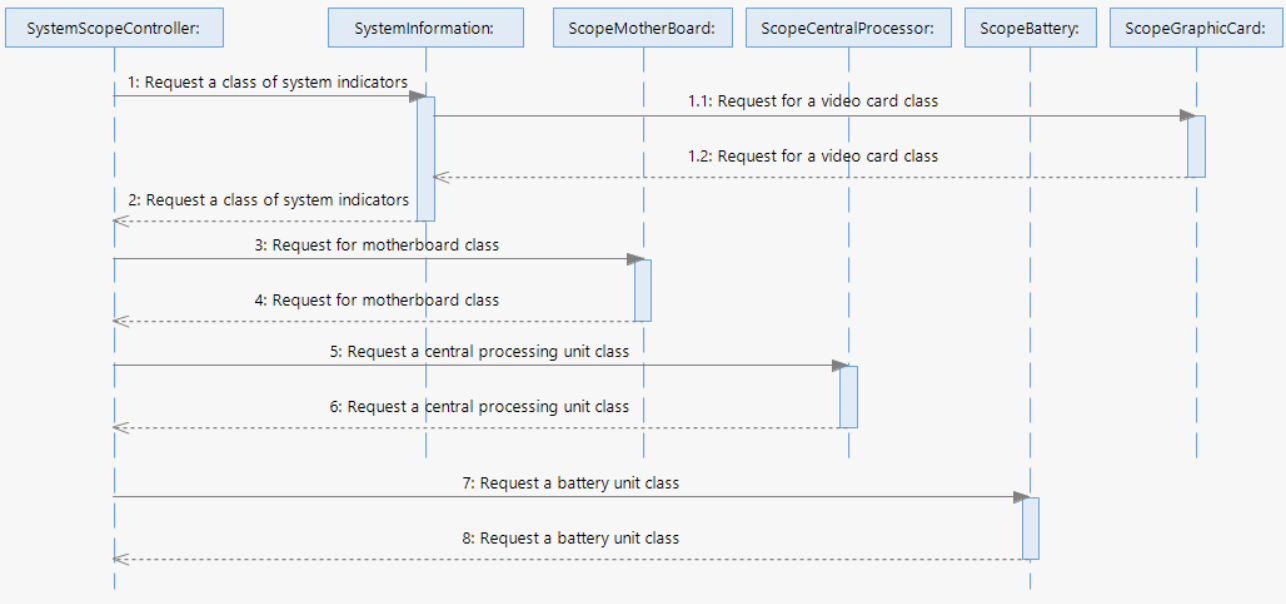

Рисунок 3.1 Діаграма послідовностей Sequence Diagram перегляду показників апаратного забезпечення

Ця діаграма послідовностей фокусується на взаємодії між основними контролерами та модулями збору інформації для отримання показників апаратного забезпечення. Основними учасниками цієї взаємодії є:

- SystemScopeController: Контролер графічного інтерфейсу, який ініціює запити на отримання даних про систему.
- SystemInformation: Клас, що надає загальну інформацію про систему та може направляти запити до інших спеціалізованих модулів.

- ScopeMotherBoard: Клас, відповідальний за збір даних про материнську плату.
- ScopeCentralProcessor: Клас для отримання даних про центральний процесор (CPU).
- ScopeBattery: Клас для отримання даних про стан акумулятора.
- ScopeGraphicCard: Інтерфейс для класів, що відповідають за отримання даних про відеокарти.

3.3 Проєктування архітектури

Проектування архітектури системи візуального моніторингу базується на принципах розподілу відповідальностей, модульності та розширюваності, що забезпечує гнучкість, легкість та масштабованість підтримки програмного комплексу. Архітектура системи є багаторівневою і складається з логічно відокремлених розроблених компонентів, кожен з яких виконує специфічні функції.

Розробка програмного комплексу візуального моніторингу показників комп'ютерного обладнання базується на принципах об'єктно-орієнтованого проєктування (ООП) та застосуванні архітектурних патернів проєктування, що забезпечують чітке розділення відповідальності між компонентами. Враховуючи вимоги до інтерактивності інтерфейсу та необхідності обробки даних, обрано архітектуру Model-View-Controller (MVC).

Модель (Model): Представляє бізнес-логіку та дані. У даному випадку це класи, що відповідають за збір системних показників (наприклад, SystemInformation, ScopeCentralProcessor, ScopeGraphicCard, ScopeBattery, ScopeMotherBoard, ScopeUsbDevice), їх обробку та збереження (DataStorage).

Представлення (View): Відповідає за відображення даних користувачеві. Це FXML-файли та відповідні класи графічних елементів.

Контролер (Controller): Здійснює взаємодію між моделлю та представленням, обробляє події користувача та оновлює представлення на основі змін у розробленій моделі. У даній архітектурі це класи, що успадковуються від інтерфейсу BaseScopeController та реалізують клас SceneController (наприклад, SystemScopeController – контроллер основного вікна, ScopeChartsController – контроллер відображення графіків, ScopeLogsViewerController – контроллер відображення файлів логів, SettingsViewController – контроллер відображення вікна параметрів системи, AiChatController – контроллер взаємодії користувача з штучним інтелектом, BenchSelectorController – контроллер вікна вибору процесу для проведення бенчмарку, SystemTrayMenuWindowController – контроллер який взаємодіє з системним треєм для відображення активності програми). Вони відповідають за логіку візуального відображення даних, обробку вводу користувача та взаємодію з модулями збору даних.

Обраний мною такий підхід забезпечує гнучкість, спрощує тестування окремих розроблених компонентів системи та дозволяє легко модифікувати графічний інтерфейс системи без суттєвих змін у логіці роботи інших модулів.

Програмний комплекс складається з декількох ключових компонентів, які взаємодіють між собою для забезпечення необхідного повноцінного функціонування системи моніторингу. Загальна структура комплексу відображена на діаграмі класів (див. рис. 3.2).

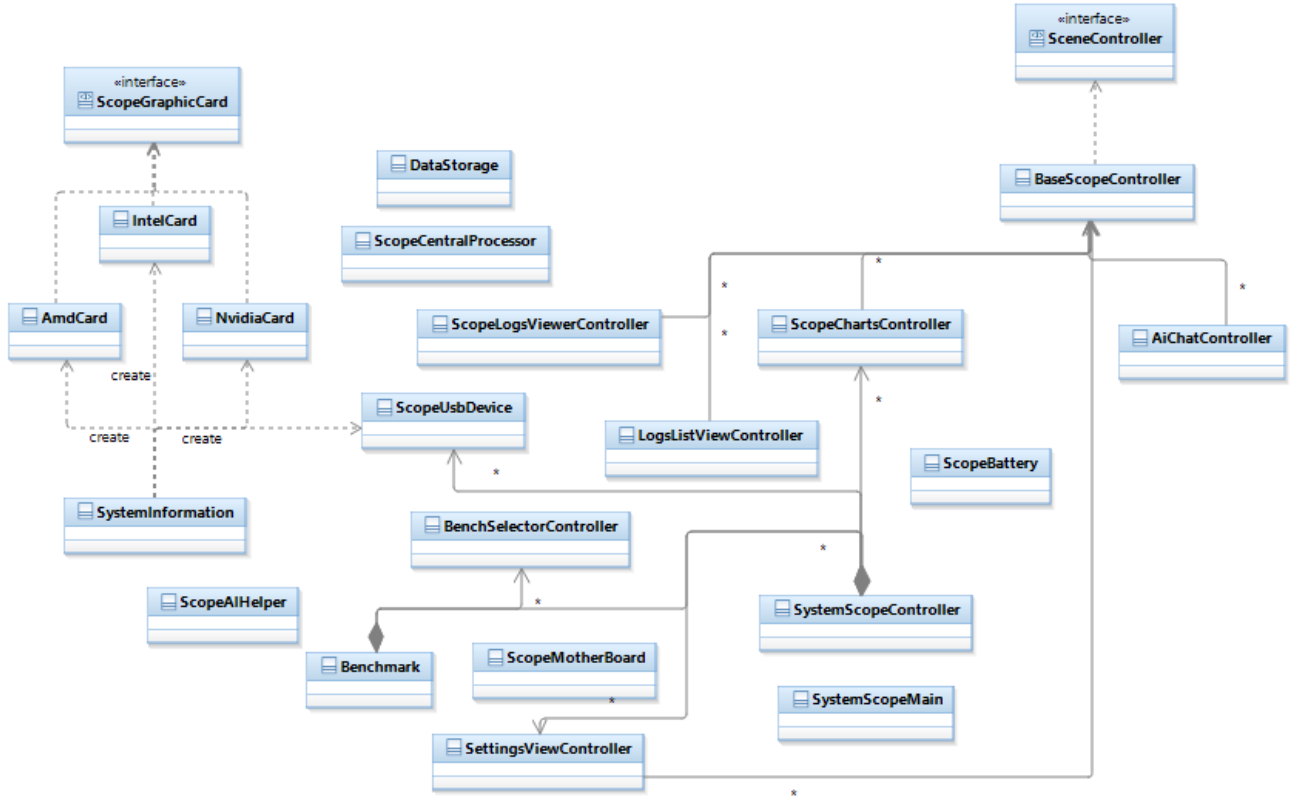

Рисунок 3.2 – Діаграма класів Class Diagram для системи візуального моніторингу апаратного забезпечення “SystemScope”

Модуль збору системних даних: це модуль який відповідає за класи які необхідні для загального збору показників апаратного забезпечення.

SystemInformation це базовий клас який дозволяє отримувати загальну інформацію про систему та її показники.

ScopeCentralProcessor це клас, який відповідає за моніторинг показників центрального процесора (навантаження, температура).

ScopeGraphicCard (інтерфейс), IntelCard, AmdCard, NvidiaCard це розроблені програмні модулі для роботи з графічними картами різних виробників, демонструючи використання патерну "Абстрактна фабрика" для створення об'єктів відеокарт на основі системної інформації.

ScopeUsbDevice це клас для моніторингу підключених до персонального комп’ютера USB-пристроїв.

ScopeMotherBoard: Моніторинг технічних показників материнської плати персонального комп’ютера.

ScopeBattery: Моніторинг стану та характеристик підключеного акумулятора (в загальному лише для ноутбуків).

Модуль управління представленнями (UI Controllers) це модуль який містить в собі класи необхідні для візуального подання інформації кінцевому користувачеві.

SceneController (інтерфейс) та BaseScopeController це класи які визначають загальний інтерфейс для контролерів сцен та базову реалізацію.

SystemScopeController це основний контролер для відображення загальних системних показників.

ScopeChartsController це розроблений програмний модуль який відповідає за відображення даних у вигляді графіків (ScopeLineChart є елементом візуалізації).

ScopeLogsViewerController, LogListViewController, ScopeLogger: це розроблені компоненти для логування системних помилок, їх перегляд та керування логами бенчмарку.

SettingsViewController це програмний модуль який дозволяє керувати налаштуваннями розробленої програми моніторингу системи.

AiChatController, ScopeAIHelper, ChatMessage, ChatMassageAdapter це розроблені програмні модулі , що відповідають за інтеграцію функціоналу чат-бота або AI-помічника, дозволяє користувачу інтегрувати свого помічника використовуючи налаштування системи та змінивши параметри API ключа.

BenchSelectorController, Benchmark, BenchWindow, ScopeBenchLog це розроблені програмні модулі для проведення бенчмарків , контролю їх відображення та запис і аналіз результатів бенчмарку.

SystemTrayMenuWindowController це модуль який керує функціоналом розробленої системи коли вона прихована і відображається в вигляді іконки в системному треї, тобто на панелі задач.

ScopeAlert, ScopeToast є розробленими класами для виведення графічних сповіщень користувачеві в необхідних ситуаціях.

Модуль управління даними та конфігурацією є розробленим програмним модулем для гручкого керування відображенням показників та збереження

необхідних записів і результатів.

DataStorage є модулем який відповідає за збереження та завантаження даних такі як налаштування конфігів, збереження логів та запис даних про системні показники.

ScopeConfigManager це модуль який керує конфігураційними файлами та параметрами розробленної системи.

Базові та службові класи:

- SystemScopeMain: Точка входу в програму, що ініціалізує основні компоненти та запускає графічний інтерфейс.
- ScopeListView, ScopeTheme, Theme: Допоміжні класи для UI елементів та управління темами системи.
- ScopeLoaderFXML: Завантажувач графічних FXML-файлів для завантаження графічного інтерфейсу.
- ProcessInfoService: Сервіс для отримання інформації про надані системні процеси.
- Sender: Клас для ідентифікації відправника повідомлення в чаті з штучним інтелектом.

3.4 Реалізація ключових класів

В цьому розділі наведено ключові фрагменти коду реалізації основних класів розробленого програмного комплексу, що забезпечують збір, обробку та візуальне представлення системних показників. Фокус зроблено на демонстрації конкретних підходів до програмування, використання бібліотек та взаємодії компонентів між собою.

Клас збору інформації SystemInformation: Даний клас SystemInformation (nm.sc.systemscope.modules.SystemInformation) є одним з найважливіших

розроблених центральних компонентів розробленого програмного комплексу. Він використовує розроблену на Java бібліотеку OSHI для взаємодії з операційною системою.

Призначення та роль у системі: Клас SystemInformation інкапсулює логіку доступу до системної інформації, надаючи статичні методи для зручного отримання даних про:

- Графічні карти (GPU).
- Оперативну пам'ять (RAM), включаючи загальний обсяг та технічні деталі про фізичні модулі.
- Дискові накопичувачі.
- Температуру та навантаження дискретної та інтегрованих GPU.
- Модель комп'ютера.
- Швидкість обертання вентиляторів (RPM).
- Підключені USB-пристрої.

Цей клас виконує роль провайдера даних для інших компонентів системи, зокрема для контролерів візуального представлення. Його функція це збирати інформації, отриманої з низькорівневих джерел (як-от бібліотека OSHI або системні утиліти), та переводити її в формат, зручний для подальшої обробки та відображення.

Цей клас забезпечує точку доступу до системних показників, що значно спрощує розробку та підтримку інших модулів. Контролери та компоненти інтерфейсу користувача не потребують знання про внутрішню логіку збору даних чи специфіку взаємодії з апаратною частиною. Нижче наведено ключові методи та їх реалізацію:

getRamInfo: Метод отримання інформації про оперативну пам’ять (враховує кросплатформеність, Windows, MacOS, Linux), надає детальну інформацію про кожен фізичний модуль пам'яті, метод дозволяє отримати такі параметри як виробник, тип, частота та об'єм пам’яті. Метод обробляє можливі винятки, якщо інформація про фізичну пам'ять недоступна через обмеження прав доступу. У разі

помилки використовується ScopeLogger.logError для запису події. Приклад коду наведено на рис. 3.3.

```java
/**
 * Retrieves detailed information about the physical memory (RAM) in the system.
 * The method may not work if you do not have the appropriate access rights to system folders.
 * @return a string with detailed information about each physical memory module.
 */
public static String getRamInfo(){  1 usage  Nazar
    try {
        List<PhysicalMemory> memoryList = layer.getMemory().getPhysicalMemory();
        if (memoryList.isEmpty()) {
            ScopeLogger.logError( message: "Physical memory information is unavailable");
            return "Physical memory information is unavailable";
        }

        StringBuilder ramInfo = new StringBuilder();

        for (PhysicalMemory memory : memoryList) {
            ramInfo.append(String.format("Виробник: %s\n", memory.getManufacturer()));
            ramInfo.append(String.format("Тип пам'яті: %s\n", memory.getMemoryType()));
            ramInfo.append(String.format("Частота: %.2f GHz\n", memory.getClockSpeed() / 1000.0));
            ramInfo.append(String.format("Об'єм: %s\n", formatMemory(memory.getCapacity())));
        }

        return ramInfo.toString();
    }
    catch(Exception e){
        ScopeLogger.logError( message: "Error while retrieving physical memory information: {}", e.getMessage());
        return "Помилка при отриманні інформації про фізичну пам'ять: " + e.getMessage();
    }
}
```

Рисунок 3.3 – Реалізація методу отримання інформації про оперативну пам’ять

getTemperatureGPU: метод який повертає температуру відеокарти різних виробників. Він динамічно створює об'єкти NvidiaCard, AmdCard або IntelCard які є реалізацією класу ScopeGraphicCard, залежно від виробника відеокарти. Кожен з цих класів має свій метод getTemperature().

В основі реалізації getTemperatureGPU є використання патерну "Фабричний метод", де вибір конкретного механізму отримання температури залежить від ідентифікованого виробника GPU. Метод спочатку опитує систему для виявлення всіх встановлених графічних карт. Для кожної виявленої карти, на основі її виробника (вендора), динамічно створюється екземпляр відповідного класу: NvidiaCard, AmdCard або IntelCard. Приклад коду наведено на рис 3.4.

```java
/**
 * Retrieves the temperature of all GPUs in the system.
 *
 * @return a string with the temperatures of all GPUs, or "No data" if not available.
 */
public static String getTemperatureGPU() {  2 usages  Nazar +1
    List<GraphicsCard> gpus = layer.getGraphicsCards();

    StringBuilder temperatures = new StringBuilder();

    for (GraphicsCard gpu : gpus) {
        String vendor = gpu.getVendor().toLowerCase();

        if (vendor.contains("nvidia")) {
            NvidiaCard nvidia = new NvidiaCard();
            temperatures.append(nvidia.getTemperature());
        } else if (vendor.contains("amd")) {
            AmdCard amd = new AmdCard();
            temperatures.append(amd.getTemperature());
        } else if (vendor.contains("intel")) {
            IntelCard intel = new IntelCard();
            temperatures.append(intel.getTemperature());
        } else {
            temperatures.append("Невідомий виробник відеокарти: ").append(gpu.getName()).append("\n");
        }
    }

    return !temperatures.isEmpty() ? temperatures.toString() : "Немає даних";
}
```

Рисунок 3.4 – Реалізація методу отримання температур графічного процесора

getGPUUsage: Аналогічно до отримання температури GPU, цей метод визначає список дискретних відеокарт та викликає відповідний метод getGPULoad() для отримання показника завантаження GPU.

Як і у випадку з температурою, getGPUUsage використовує поліморфізм та концепцію спеціалізованих класів-реалізацій (NvidiaCard, AmdCard, IntelCard), що імплементують інтерфейс ScopeGraphicCard. Після виявлення дискретної відеокарти для кожної з них викликається уніфікований метод getGPULoad(). Цей метод, реалізований у відповідних класах-нащадках, інкапсулює специфічну логіку для отримання завантаження GPU конкретного виробника. Це забезпечує гнучкість та дозволяє системі відображати завантаження GPU, незалежно від її виробника та способу отримання даних. Приклад коду наведено на рис. 3.5.

```java
/**
 * Retrieves the GPU usage percentage for the discrete GPU.
 *
 * @return a string with the GPU usage, or a message indicating no GPU is found.
 */
public static String getGPUUsage(){  3 usages  Nazar +1
    GraphicsCard gpu = getDiscreteGPU();

    if (gpu == null) {
        return "Дискретна відеокарта не знайдена";
    }

    String vendor = gpu.getVendor().toLowerCase();
    StringBuilder usage = new StringBuilder();

    if (vendor.contains("nvidia")) {
        NvidiaCard nvidia = new NvidiaCard();
        usage.append(nvidia.getGPULoad());
    } else if (vendor.contains("amd")) {
        AmdCard amd = new AmdCard();
        usage.append(amd.getGPULoad());
    } else if (vendor.contains("intel")) {
        IntelCard intel = new IntelCard();
        usage.append(intel.getGPULoad());
    } else {
        usage.append("Не підтримується виробником");
    }

    return !usage.isEmpty() ? usage.toString() : "Немає даних";
}
```

Рисунок 3.5 – Реалізація методу отримання відсотку завантаження графічного процесору

GetCPUUsage: Метод який дозволяє отримати поточне завантаження процесора використовуючи розроблену на Java бібліотеку OSHI.

Для отримання завантаження CPU метод викликає функціональність з OSHI. Цей підхід дозволяє виміряти середнє завантаження процесора за інтервал часу, забезпечуючи точний показник його активності. Результат від OSHI (значення у діапазоні від 0.0 до 1.0), конвертується у відсотковий формат, що є зручним для відображення користувачеві. Приклад коду наведено на рис. 3.6.

```java
/**
 * Retrieves the current CPU usage percentage.
 *
 * @return a string with the CPU usage as a percentage.
 */
public static String getCPUUsage(){
    CentralProcessor processor = layer.getProcessor();
    long delay = 1000;
    double loadCPU = processor.getSystemCpuLoad(delay) * 100;
    return String.valueOf((int) Math.round(loadCPU));
}
```

Рисунок 3.6 – Реалізація методу отримання відсотку завантаження процесора

GetTemperatureCPU: метод який дозволяє отримати температуру процесора в градусах Цельсія, також якщо в процесорі є влаштоване відеоядро то переважно їх температури співпадають що дозволяє визначати температуру відеоядра без надлишкових викликів інших методів. Приклад коду наведено на рис. 3.7.

```java
/**
 * Retrieves the current CPU temperature.
 *
 * @return a string with the current CPU temperature in Celsius.
 */
public static String getTemperatureCPU() { return String.valueOf(layer.getSensors().getCpuTemperature()); }
```

Рисунок 3.7 – Реалізація методу отримання температури процесора та влаштованого відеоядра

Оскільки шляхи для отримання показників відеокарт, відрізняються в залежності від обраної операційної системи та від виробника відеокарти, то мною було вирішено врахувати більшість випадків для операційних систем Windows, MacOS та Linux, на відеокарти Nvidia, Intel та AMD. Варто зазначити що між поняттям вендор та виробник відеокарти досить велика різниця. В загальному відеокарти є тільки трьох видів а саме Intel, Nvidia та AMD, тобто це виробники. Але також відеокарта може бути від вендора, наприклад Palit, MSI, Gigabyte,

Sapphire. Вендор це виробник який використовує готове графічне ядро від виробників Intel, Nvidia та AMD і на основі їх ядра виробляє повноцінну відеокарту, тому фактично якщо відеокарта на основі графічного ядра від Nvidia, то шлях отримання показників буде такий самий як для відеокарти від Nvidia.

Клас IntelCard: призначений для збору інформації про відеоркарту від виробника Intel.

Призначення та роль у системі: призначений для збору даних графічних процесорів Intel. Оскільки відеокарти Intel не випускають дискретні версії відеокарт, лише інтегровані, тобто вбудовані в самий центральний процесор то їх температури будуть співпадати.

Особливості реалізації: Для відеокарт Intel, температурні показники GPU зазвичай ідентичні показникам температури центрального процесора. Це означає, що метод отримання температури в класі IntelCard звертається до того ж джерела даних, що і метод getTemperatureCPU() у класі ScopeCentralProcessor, оскільки вбудована графіка та CPU є частиною одного кристала. Така реалізація дозволяє уникнути дублювання запитів. Нижче наведено ключові методи та їх реалізація:

getTemperature: метод який дозволяє отримати температуру вбудованого графічного ядра від виробника Intel використовуючи ту саму температуру що і центральний процесор. Приклад коду наведено на рис. 3.8.

```
/**
 * Retrieves the temperature for an Intel GPU.
 * This method calls the {@link ScopeCentralProcessor#getTemperatureCPU()} method to get the
 * CPU temperature, which may reflect the temperature of Intel's integrated GPU.
 *
 * @return a string with the temperature of the Intel GPU.
 */
@Override public String getTemperature() { return ScopeCentralProcessor.getTemperatureCPU(); }
```

Рисунок 3.8 – Реалізація методу отримання температури інтегрованого відеоядра

getGPULoad: метод який дозволяє отримати відсоток завантаження вбудованого відеоядра від Intel в залежності від обраної операційної системи. Приклад коду наведено на рис. 3.9.

```java
 * Retrieves the GPU load for an Intel GPU.
 * This method checks the operating system to determine the appropriate command to execute:
 * - On Windows, GPU load retrieval is not supported, and a message "Not supported." is returned.
 * - On Linux and Unix-like systems (including macOS), it uses the `intel_gpu_top` command to get the load.
 *
 * @return a string with the load percentage for the Intel GPU, or an error message if not available.
 */
@Override public String getGPULoad() {  3 usages  nazar
    String os = System.getProperty("os.name").toLowerCase();
    StringBuilder result = new StringBuilder();

    try {
        if (os.contains("win")) {
            result.append("Not supported.");
        } else if (os.contains("nix") || os.contains("nux") || os.contains("mac")) {
            Process process = Runtime.getRuntime().exec( command: "sudo intel_gpu_top -d 1 -n 1");
            BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream()));
            String line;

            while ((line = reader.readLine()) != null) {
                if (line.contains("Gpu") || line.contains("Render") || line.contains("Active")) {
                    result.append(line.trim()).append("\n");
                }
            }
        }
    } catch (Exception e) {
        ScopeLogger.logError( message: "Error while retrieving load for Intel GPU");
        result.append("Помилка при отриманні завантаження для Intel GPU");
    }

    return !result.isEmpty() ? result.toString() : "Не вдалося отримати завантаження Intel GPU";
```

Рисунок 3.9 – Реалізація методу отримання відсотку завантаження інтегрованого відеоядра від Intel

Клас NvidiaCard: клас який реалізує інтерфейс ScopeGraphicCard та розроблений для взаємодії з більшістю відеокарт від виробника Nvidia, зазвичай в більшості випадків в користувачів використовуються саме відеокарти від виробника Nvidia.

Призначення та роль у системі: NvidiaCard відповідає за збір показників відеоядра, таких як температура, завантаження, використання пам'яті та частоти для карт NVIDIA. Це вимагає використання специфічних API або утиліт, оскільки NVIDIA надає власні інструменти для моніторингу своїх GPU.

Особливості реалізації: Для отримання даних про графічне відеоядро GPU NVIDIA, клас NvidiaCard може використовувати як мінімум два різних підходи:

- Використовувати NVIDIA Management Library (NVML): Це пропрієтарна бібліотека від NVIDIA, яка надає API для моніторингу та керування GPU NVIDIA. Для доступу до NVML з Java може бути використаний JNI (Java Native Interface) або спеціальні Java-обгортки для NVML. Це дозволяє отримувати точні показники температури, завантаження, використання пам'яті та інших параметрів.
- Парсити вивід командної утиліти nvidia-smi: Якщо прямий доступ до NVML ускладнений, клас може запускати зовнішній процес nvidia-smi та парсити його вивід для отримання необхідної інформації. Цей підхід є простішим у реалізації, але може бути менш ефективним та більш чутливим до змін у виводі nvidia-smi.

В даній реалізації було використано підхід використання nvidia-smi з можливістю майбутнього розширення до використання обох підходів якщо один з них не дає необхідного результату. Також аргументуючи вибір варто зазначити що даних підхід з використанням утиліти nvidia-smi працює на всіх операційних системах, що дозволяє уникнути надлишкового коду для ідентифікації використовуваної користувачем операційної системи. Нижче наведено ключові методи та їх реалізація:

getTemperature: метод який дозволяє отримати температуру дискретного графічного ядра від виробника Nvidia використовуючи nvidia-smi за допомогою команди nvidia-smi --query-gpu=temperature.gpu --format=csv,noheader. Приклад коду наведено на рис. 3.10.

```java
/**
 * Retrieves the temperature for an NVIDIA GPU.
 *
 * @return a string with the temperature of the NVIDIA GPU.
 */
@Override public String getTemperature(){  nazar
    try {
        Process process = Runtime.getRuntime().exec( command: "nvidia-smi --query-gpu=temperature.gpu --format=csv,noheader");
        BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream()));
        String line;
        StringBuilder result = new StringBuilder();
        while ((line = reader.readLine()) != null) {
            result.append("NVIDIA GPU: ").append(line.trim()).append(" °C\n");
        }
        return result.toString();
    } catch (Exception e) {
        ScopeLogger.logError( message: "Error while retrieving temperature for NVIDIA GPU");
        return "Помилка при отриманні температури для NVIDIA GPU";
    }
```

Рисунок 3.10 – Метод для отримання температури дискретної відеокарти від виробника Nvidia

getGPULoad: метод який дозволяє отримати відсоток завантаження дискретного графічного відеоядра від виробника Nvidia за допомогою nvidia-smi та команди nvidia-smi --query-gpu=utilization.gpu --format=csv,noheader,nounits на різних операційних системах. Приклад коду наведено на рис. 3.11.

```java
/**
 * Retrieves the GPU load for an NVIDIA GPU.
 *
 * @return a string with the load percentage for the NVIDIA GPU.
 */
@Override public String getGPULoad(){  3 usages  nazar
    try {
        Process process;
        process = Runtime.getRuntime().exec( command: "nvidia-smi --query-gpu=utilization.gpu --format=csv,noheader,nounits");

        BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream()));
        return reader.readLine();
    } catch (IOException e) {
        return "Не вдалося отримати використання для NVIDIA";
    }
}
```

Рисунок 3.11. Метод для отримання відсотку завантаження дискретної відеокарти від виробника Nvidia

Клас AmdCard: є ще однією реалізацією інтерфейсу ScopeGraphicCard, призначеною для моніторингу дискретних графічних карт від виробника AMD. Цей клас, подібно до попередніх класів NvidiaCard та IntelCard, адаптує свій підхід до

збору даних залежно від операційної системи, використовуючи різні доступні системні команди.

Призначення та роль у системі: AmdCard відповідає за отримання ключових показників температури та завантаження для графічних процесорів AMD. Його інтеграція в загальну архітектуру через інтерфейс ScopeGraphicCard забезпечує уніфікований спосіб доступу до даних GPU, незалежно від виробника.

Особливості реалізації: Як і інші специфічні для вендорів класи, AmdCard імплементує інтерфейс ScopeGraphicCard, надаючи конкретні реалізації методів getTemperature() та getGPULoad() для отримання системних показників відеокарти, що дозволяє досягти поліморфізму та гнучкості в системі моніторингу. Нижче наведено ключові методи та їх реалізація:

getTemperature: Цей метод відповідає за отримання поточної температури графічного процесора (GPU) для відеокарт AMD. Його реалізація є кросплатформною та адаптується до особливостей операційної системи, на якій запущено програмний комплекс. На операційних системах сімейства Windows для отримання температурних показників використовується команда wmic /namespace:\\\\root\\wmi PATH MSAcpi_ThermalZoneTemperature get CurrentTemperature. Ця команда взаємодіє з WMI (Windows Management Instrumentation) – ключовою технологією Microsoft для керування та моніторингу системних компонентів. Вона дозволяє отримати загальні показники температури з різних термозон системи, які можуть включати температуру GPU, особливо для дискретних карт, що відображаються у загальній системі моніторингу температури.

На Unix-подібних платформах для отримання температури GPU використовується утиліта sensors. Ця утиліта є частиною пакета lm-sensors, який є фактичним стандартом для моніторингу апаратних сенсорів на Unix-подібних системах. Вивід команди sensors фільтрується за ключовим словом "gpu" (за допомогою grep -i 'gpu'), щоб вилучити лише необхідні дані, що стосуються графічного процесора. Такий підхід дозволяє отримати прямі показники температури GPU. Приклад коду наведено на рис 3.12.

```java
 * Retrieves the temperature for an AMD GPU.
 * This method executes platform-specific commands to retrieve the temperature of the AMD GPU.
 * On Windows, it uses WMIC to get the temperature.
 * On Linux or Unix-based systems, it uses the `sensors` command to extract GPU temperature.
 *
 * @return a string containing the temperature of the AMD GPU, in degrees Celsius.
 *         If an error occurs, it returns an error message.
 */
@Override public String getTemperature() {  nazar
    try {
        Process process;
        String os = System.getProperty("os.name").toLowerCase();

        if (os.contains("win")) {
            process = Runtime.getRuntime().exec( command: "wmic /namespace:\\\\root\\wmi PATH MSAcpi_ThermalZoneTemperature get CurrentTemperature");
        } else {
            process = Runtime.getRuntime().exec( command: "sensors | grep -i 'gpu'");
        }

        BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream()));
        String line;
        StringBuilder result = new StringBuilder();
        while ((line = reader.readLine()) != null) {
            result.append("AMD GPU: ").append(line.trim()).append(" °C\n");
        }
        return result.toString();
    } catch (Exception e) {
        ScopeLogger.logError( message: "Error while retrieving temperature for AMD GPU");
        return "Помилка при отриманні температури для AMD GPU";
    }
}
```

Рисунок 3.12 – Реалізація методу для отримання температури відеокарт від виробника AMD

getGPULoad: На Windows використовується wmic path Win32_VideoController get LoadPercentage для отримання відсотка завантаження відеоконтролера. На Linux та MacOS спочатку спроба отримати завантаження через sensors | grep -i 'gpu'. Якщо sensors не повертає дані про завантаження GPU (вивід виявляється порожнім), здійснюється спроба використовувати утиліту radeontop (якщо вона встановлена в системі) з параметрами radeontop -d 1 -n 1 для отримання інформації про завантаження. Вивід radeontop також фільтрується за ключовим словом "Load". Приклад коду наведено на рис. 3.13.

```
@Override public String getGPULoad() {  3 usages   nazar
    if (os.contains("win")) {
        Process process = Runtime.getRuntime().exec( command: "wmic path Win32_VideoController get LoadPercentage");
        BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream()));
        String line;

        while ((line = reader.readLine()) != null) {
            if (line.contains("%")) {
                result.append(line.trim()).append("\n");
            }
        }
    } else if (os.contains("nix") || os.contains("nux") || os.contains("mac")) {
        Process process = Runtime.getRuntime().exec( command: "sensors | grep -i 'gpu'");
        BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream()));
        String line;

        while ((line = reader.readLine()) != null) {
            result.append(line.trim()).append("\n");
        }

        if (result.isEmpty()) {
            process = Runtime.getRuntime().exec( command: "radeontop -d 1 -n 1");
            reader = new BufferedReader(new InputStreamReader(process.getInputStream()));

            while ((line = reader.readLine()) != null) {
                if (line.contains("Load")) {
                    result.append(line.trim()).append("\n");
                }
            }
        }
    }
```

Рисунок 3.13 – Реалізація методу для отримання завантаження відеокарти від виробника AMD

Клас ScopeAIHelper: Клас ScopeAIHelper є ключовим компонентом програмного комплексу (nm.sc.systemscope.modules.ScopeAIHelper), що відповідає за інтеграцію функціоналу штучного інтелекту (AI) та взаємодію з зовнішніми AI-сервісами через HTTP-запити. Цей клас забезпечує функціональність інтелектуального помічника, дозволяючи користувачеві отримувати відповіді на запити, пов'язані з системною інформацією або іншими аспектами.

Призначення та роль у системі: ScopeAIHelper виконує у системі збору даних функцію управління історією чату, що дозволяє AI-моделі підтримувати контекст розмови також ініціалізаціє AI-моделі попередніми повідомленнями, відправляє запити до AI-сервісу та оброблює отримані відповіді і забезпечує безпечну передачу даних шляхом екранування JSON-рядків.

Цей клас є мостом між інтерфейсом користувача та зовнішнім AI-сервісом,

що дозволяє надавати інтелектуальну допомогу. Нижче наведено ключові методи та їх реалізація:

Метод reques: Це основний розроблений метод який дозволяє відправити запит до AI-моделі використовуючи підготовлений http запит. Приклад коду наведено на рис. 3.14.

```
public static String request(String prompt) {  4 usages   nazar
    try {
        URL url = new URL(API_URL);
        HttpURLConnection connection = (HttpsURLConnection) url.openConnection();

        connection.setRequestMethod("POST");
        connection.setRequestProperty("Authorization", "Bearer " + API_KEY);
        connection.setRequestProperty("Content-Type", "application/json");
        connection.setDoOutput(true);

        JsonObject userMessage = new JsonObject();
        userMessage.addProperty( property: "role",  value: "user");
        userMessage.addProperty( property: "content", escapeJson(prompt));
        previousChatHistory.add(userMessage);

        String jsonRequest = String.format("""
        {
          "model": "%s",
          "messages": %s
        }
        """, MODEL, previousChatHistory.toString());

        try (OutputStream os = connection.getOutputStream()) {
            byte[] input = jsonRequest.getBytes(StandardCharsets.UTF_8);
            os.write(input,  off: 0, input.length);
        }

        int status = connection.getResponseCode();
```

Рисунок 3.14 – Реалізація методу для створення запиту до ШІ

Метод extractMessage: Допоміжний метод для парсингу JSON-відповіді від AI-сервісу, який використовує бібліотеку Gson(com.google.gson.*) для розбору Json рядка. Отримує JSON-об'єкт, потім шукає масив "choices", з якого вилучає перший

елемент, а з нього – об'єкт "message" та його "content" поле. Приклад коду наведено на рис 3.15.

```
/**
 * Extracts the content message from the AI's response.
 *
 * @param json The raw JSON response from the AI service.
 * @return The AI's response content, or an error message if the extraction fails.
 */
private static String extractMessage(String json) {  1 usage   nazar
    try {
        JsonObject jsonObject = JsonParser.parseString(json).getAsJsonObject();
        JsonArray choices = jsonObject.getAsJsonArray( memberName: "choices");
        if (choices != null && !choices.isEmpty()) {
            JsonObject message = choices.get(0).getAsJsonObject().getAsJsonObject( memberName: "message");
            return message.get("content").getAsString();
        }
    } catch (Exception e) {
        ScopeLogger.logError( message: "Помилка при розборі JSON: ", e);
    }
    return "❌ Не вдалося обробити відповідь від AI.";
}
```

Рисунок 3.15 – Реалізація методу парсингу JSON запиту

Класи BenchWindow та Benchmark це класи які є центральними компонентами, що відповідають за функціонал проведення бенчмарків та відображення ключових системних показників у реальному часі під час виконання навантажувальних тестів (наприклад, під час запуску ігор). Вони забезпечують запуск зовнішніх процесів, моніторинг їхнього життєвого циклу та динамічне відображення даних на екрані.

Призначення та роль у системі: BenchWindow: Реалізує відокремлене, вікно на екрані, яке відображає основні показники системи (температуру та завантаження CPU та GPU) в режимі реального часу під час бенчмарку. Вікно є прозорим та завжди знаходиться поверх інших вікон в лівій верхній частині монітору, що дозволяє спостерігати за показниками.

В свою чергу Benchmark є класом, що керує логікою всього процесу бенчмарку, від вибору файлу для запуску процесу до його зупинки та збереження результатів.

Основні механізми взаємодії з ОС та відображення: Клас Benchmark дозволяє

користувачеві вибрати виконуваний файл (наприклад, .exe для Windows або .sh для Linux/macOS) за допомогою інтерфейсу BenchSelectorController. Запуск файлу здійснюється через Java клас ProcessBuilder, який дозволяє виконувати зовнішні команди операційної системи. Для Linux або MacOS перед спробою запуску файлу, системою забезпечується його доступність до виконання командою chmod +x. Benchmark також фіксує час початку та завершення бенчмарку для розрахунку його тривалості.

Клас BenchWindow використовує готовий Java клас ScheduledExecutorService для регулярного оновлення показників з фіксованим інтервалом який користувач може змінити в налаштуваннях. Це забезпечує відображення даних у реальному часі. Дані про ЦП (CPU) отримуються за допомогою методів ScopeCentralProcessor.getTemperatureCPU() та ScopeCentralProcessor.getCPUUsage().

Дані про GPU а саме температура та завантаження отримуються за допомогою методів SystemInformation.getTemperatureDiscreteGPU() та SystemInformation.getGPUUsage(). Ці методи, як було описано раніше, взаємодіють з ОС специфічними утилітами (wmic, sensors, radeontop) для отримання необхідної інформації.

Вікно BenchWindow створюється за допомогою Swing javax.swing.JFrame. Важливою особливістю є його прозорість за допомогою властивості frame.setBackground(new Color(0, 0, 0, 0)) чого не вийшло досягти через JavaFX, та встановлення setUndecorated(true) для видалення стандартних рамок вікна.

Відображення кожного показника таких як температура CPU/GPU та завантаження CPU/GPU є опціональним і залежить від налаштувань користувача в конфігураційному файлі, які завантажуються зі ScopeConfigManager.

Після завершення бенчмарку, клас Benchmark збирає всі дані, розраховує середні значення за допомогою методів average() з BenchWindow. Ці дані можуть бути збережені у лог-файл за допомогою DataStorage.createLogFile(), якщо це дозволено налаштуваннями користувача ScopeConfigManager.isSaveBenchLogs(). Це забезпечує можливість подальшого аналізу та порівняння результатів.

Взаємодія цих двох класів забезпечує повноцінний цикл проведення бенчмарку, від його запуску та моніторингу до відображення актуальних даних та збереження фінальних показників, надаючи користувачеві хороший інструмент для оцінки продуктивності системи під час виконання конкретного процесу.

3.5 Розгортання програмного комплексу

Розгортання розробленого програмного забезпечення здійснюється шляхом генерації самодостатнього виконуваного JAR-файлу, що забезпечує незалежність від додаткових бібліотек у цільовому середовищі виконання. Для цього використовується система автоматичної збірки Maven з відповідними плагінами. Конфігурація процесу збірки, визначена у файлі pom.xml, передбачає компіляцію вихідного коду де проєкт компілюється з використанням Java Development Kit (JDK) версії 17. Це досягається за допомогою плагіна maven-compiler-plugin, налаштованого версії Java 17, пакування з усіма залежностями де застосовується плагін maven-assembly-plugin дозволяє створити єдиний архів. Цей архів інкапсулює весь необхідний байт-код проєкту та всі його зовнішні бібліотечні залежності, що усуває потребу в ручному управлінні залежностями під час розгортання. Також визначається точка входу у файлі маніфесту згенерованого JAR-файлу де явно вказано основний клас для запуску програми – nm.sc.systemscope.SecondMain. Це дозволяє виконувати додаток безпосередньо як стандартний виконуваний JAR-файл. На рисунку 3.16 наведено приклад плагіну для створення виконуваного jar-файлу.

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.13.0</version>
            <configuration>
                <source>17</source>
                <target>17</target>
            </configuration>
        </plugin><plugin>
            <artifactId>maven-assembly-plugin</artifactId>
            <configuration>
                <archive>
                    <manifest><mainClass>nm.sc.systemscope.SecondMain</mainClass></manifest>
                </archive>
                <descriptorRefs>
                    <descriptorRef>jar-with-dependencies</descriptorRef>
                </descriptorRefs>
            </configuration>
            <executions>
                <execution>
                    <id>make-assembly</id>
                    <phase>package</phase>
                    <goals><goal>single</goal></goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

Рисунок 3.16 – Приклад коду створення виконуваного файлу

Після виконання команди mvn package у кореневому каталозі проєкту, сформований JAR-файл буде розміщено у підкаталозі target/. Для запуску програми на будь-якій операційній системі, що має встановлену Java Runtime Environment (JRE) версії 17 або вище, достатньо виконати команду: java -jar <ім'я_файлу>.jar Саме такий підхід значно спрощує процес розгортання та забезпечує кросплатформеність розробленого програмного рішення.

3.6 Приклад роботи програмного комплексу

Після запуску програмного комплексу користувачу відображається головне вікно, яке надає інформацію про основні компоненти комп'ютерного обладнання. Інтерфейс розроблений з урахуванням вимог для забезпечення інтуїтивно зрозумілого доступу до даних.

На рисунку 3.17 показано головне вікно з відображенням поточних показників ЦП (CPU), оперативної пам'яті (RAM), інформації про відеокарту та інші ключові системні параметри. Дані оновлюються в режимі реального часу, забезпечуючи актуальну картину стану системи. Також вікно відображає температурні показники обладнання в режимі реального часу, дозволяє переглянути всі запущені системні процеси та завершити їх, переглянути підключені usb пристрої та перейти в інші модулі системи.

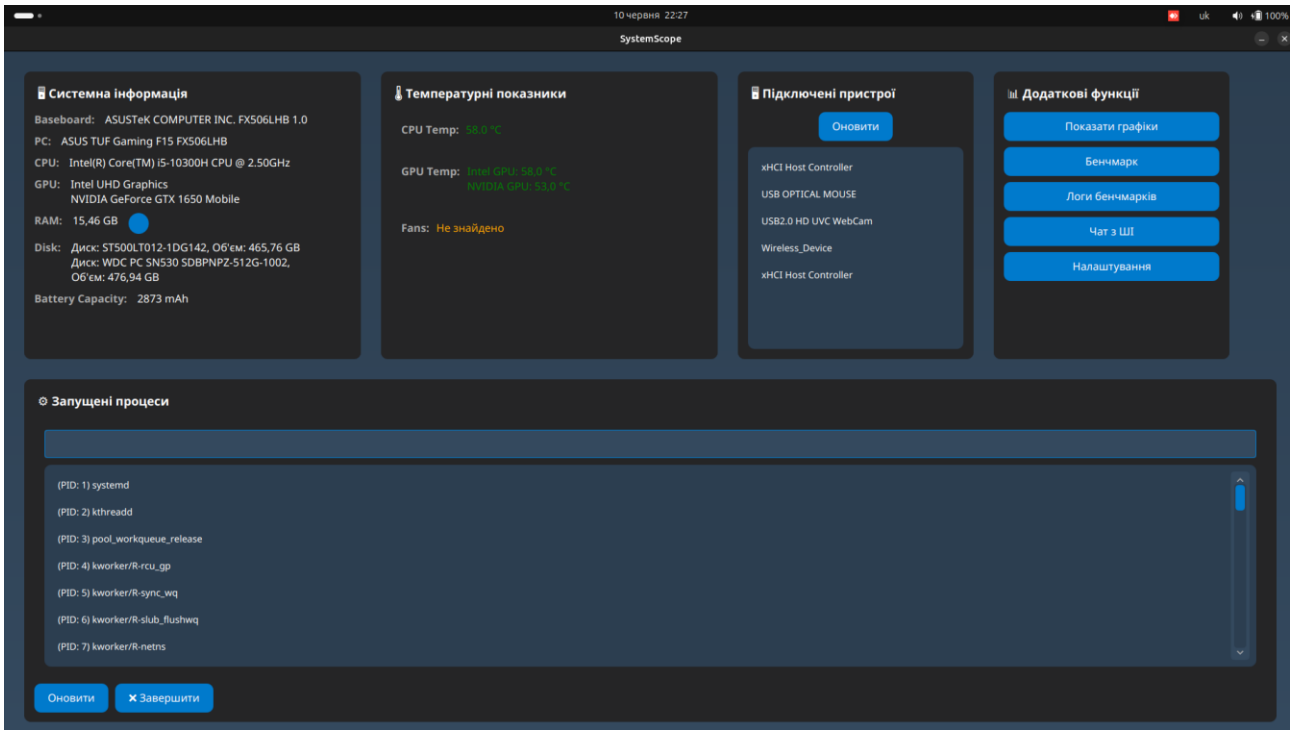

Рисунок 3.17 – Головне вікно програмного комплексу SystemScope

Для глибшого аналізу динаміки зміни показників апаратного забезпечення, програмний комплекс надає окремий модуль візуалізації даних у вигляді графіків. Це дозволяє користувачеві відстежувати зміни параметрів протягом певного періоду часу, виявляти аномалії, пікові навантаження або температурні стрибки. На рисунку 3.18 представлено приклад роботи модуля з графічним відображенням зміни системних показників.

Рисунок 3.18 – Вікно відображення графіків програмного комплексу SystemScope

Однією з інноваційних функцій програмного комплексу є інтегрований AI-модуль, який дозволяє користувачеві отримувати відповіді та поради щодо зібраних системних показників або загального стану обладнання. Цей модуль працює як чат-бот, підтримуючи контекст розмови. На рисунку 3.20 продемонстровано інтерфейс чат-бота, де користувач може ввести запит, а система надає відповідь, використовуючи можливості інтегрованої моделі штучного інтелекту.

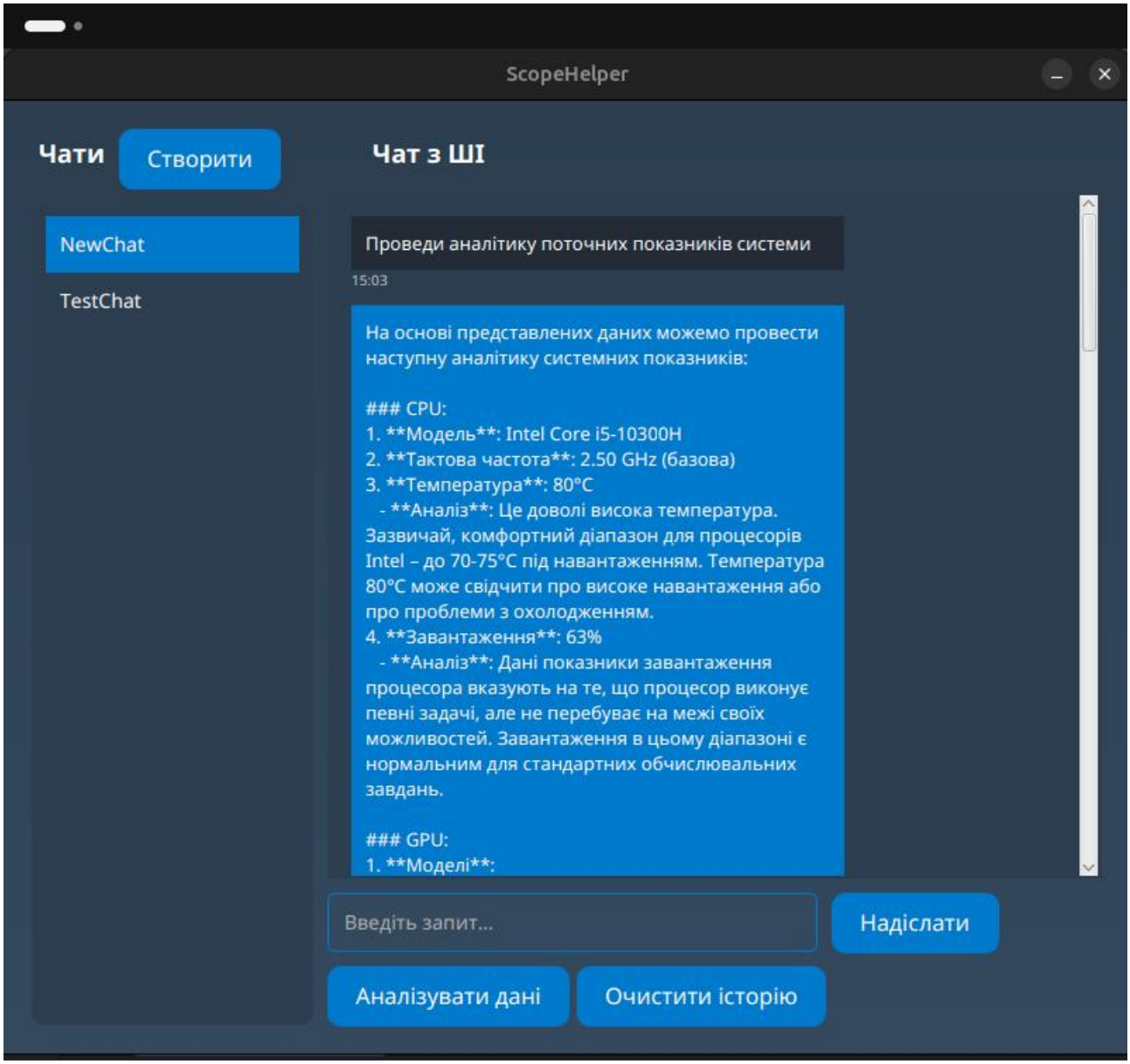

Рисунок 3.19 – Вікно чату з помічником ScopeAIHelper

Програмний комплекс надає гнучкі можливості для налаштування параметрів моніторингу та конфігурації штучного інтелекту, що дозволяє користувачеві адаптувати систему під власні потреби. Це включає зміну інтервалу оновлення даних, встановлення порогових значень для сповіщень та інші конфігураційні опції, зміна API ключа, моделі та опису щодо потреб відповіді штучного інтелекту. На рисунку 3.20 показано вікно налаштувань, де користувач може змінювати відповідні параметри моніторингу.

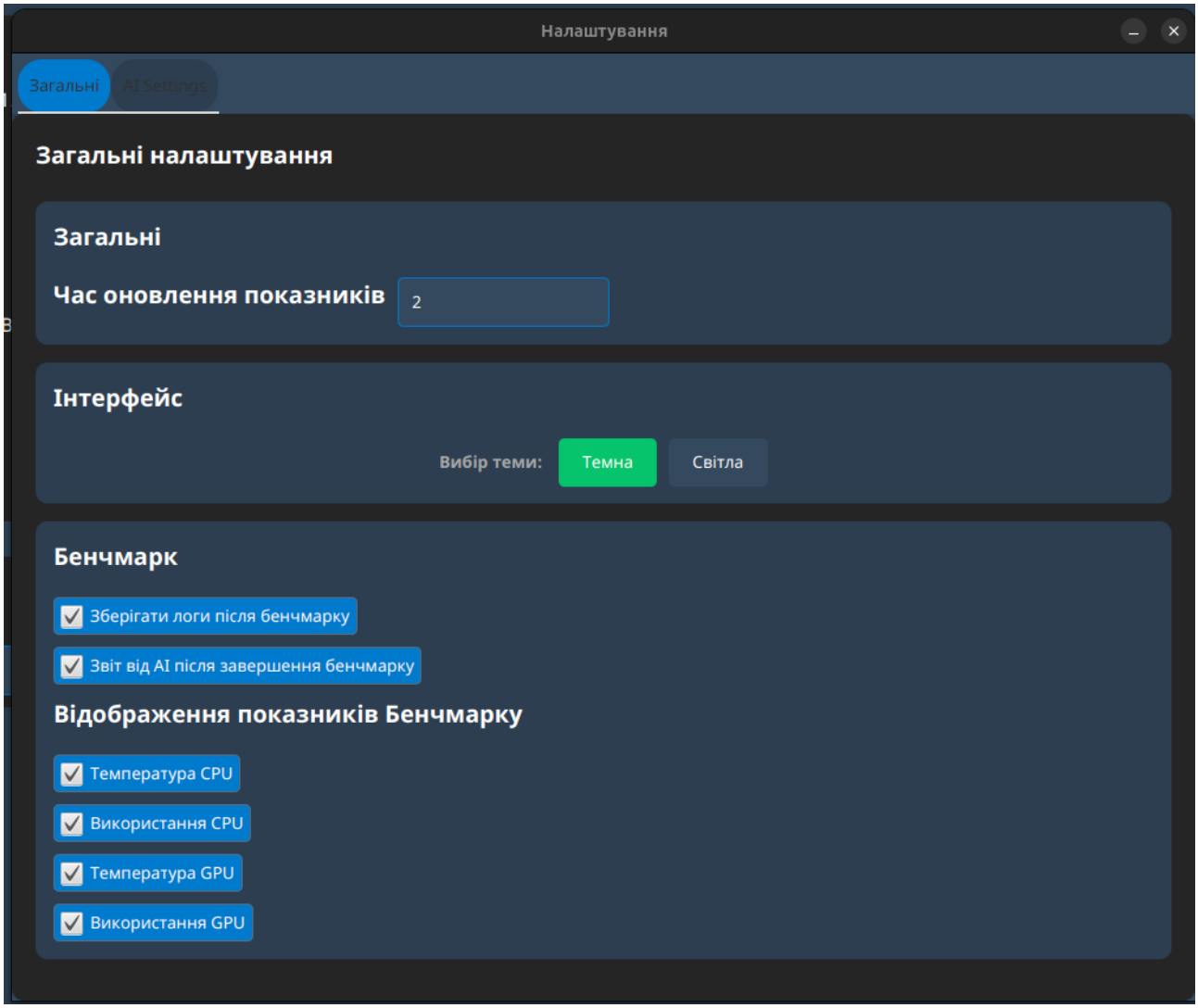

Рисунок 3.20 – Вікно налаштувань основних параметрів програмного комплексу SystemScope

На рисунку 3.21 відображено вікно налаштувань модуля штучного інтелекту, яке дозволяє користувачеві гнучко адаптувати конфігурацію AI-помічника під власні потреби. Цей функціонал є важливим для забезпечення оптимальної взаємодії з AI, дозволяючи налаштовувати такі параметри, як вибір моделі, початковий опис її ролі та, інші специфічні для AI-сервісу опції. Зміни, внесені через цей інтерфейс, впливають на те, як ScopeAIHelper взаємодіє з зовнішнім AI-сервісом, забезпечуючи налаштовану поведінку помічника.

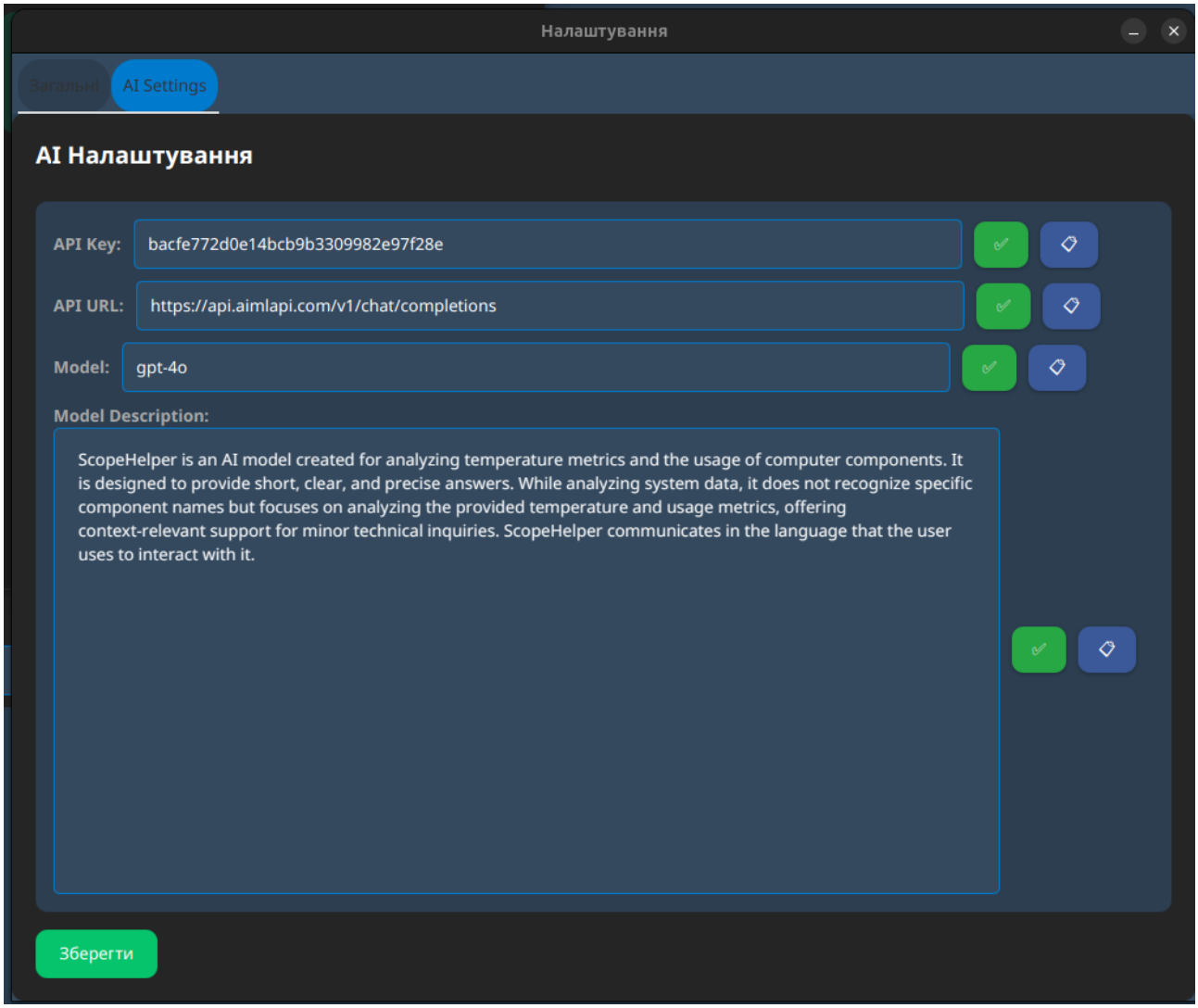

Рисунок 3.21 – Вікно налаштувань конфігураційних параметрів помічника ScopeAIHelper

Для діагностики проблем та аналізу подій, що відбуваються в системі, програмний комплекс передбачає функціонал перегляду системних логів та сповіщень. Це дозволяє відстежувати хід виконання бенчмарків, виявляти помилки та переглядати автоматичні висновки AI. На рисунку 3.22 представлено вікно перегляду логів проведення бенчмарку та відгук ШІ.

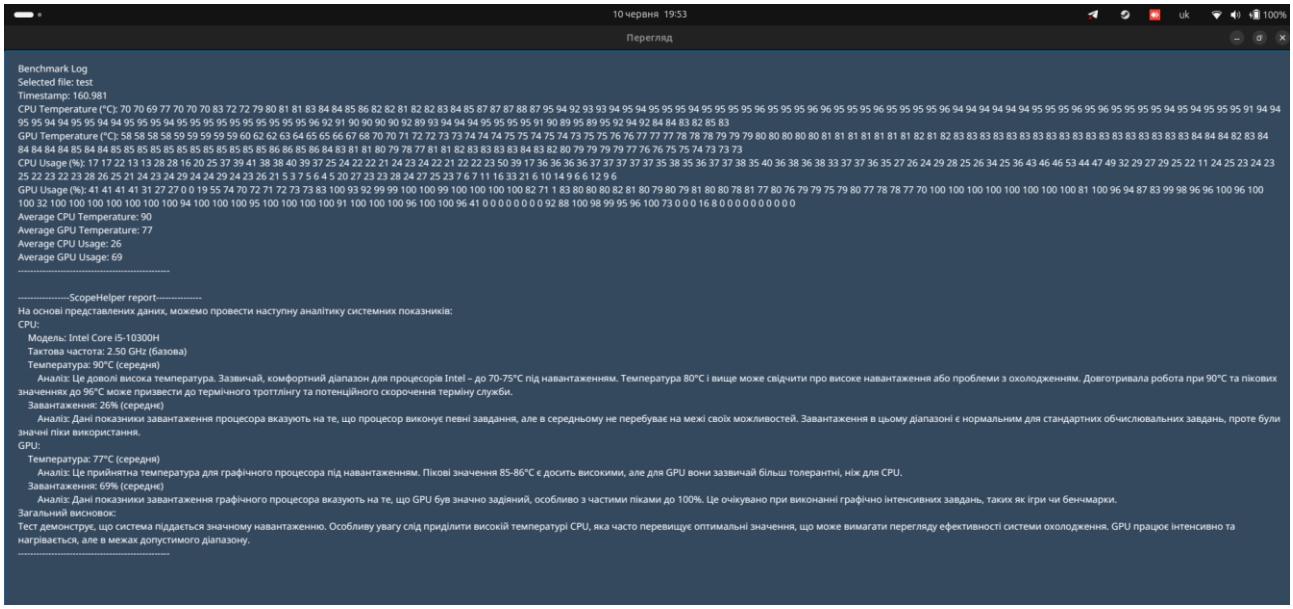

Рисунок 3.22 – Вікно відображення файлу логів бенчмарку

Програмний комплекс SystemScope має функціонал для проведення тестів продуктивності компонентів системи під навантаженням. Ця функція є корисною для оцінки продуктивності обладнання при виконанні важких завдань. На рисунку 3.23 показано вікно, що запускає бенчмарк та відображає його фінальні результати.

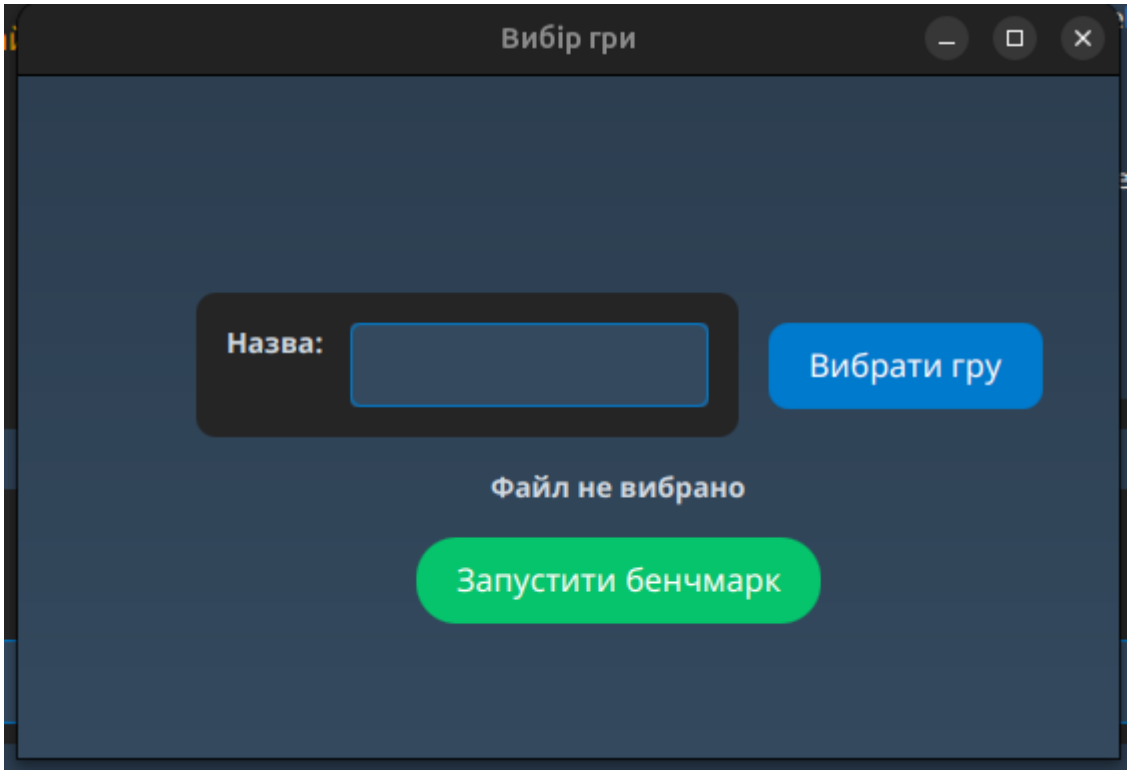

Рисунок 3.23 – Вікно вибору файлу для проведення тесту продуктивності

На рисунку 3.24 наведено приклад вікна з відображенням системних показників комп’ютерного обладнання під час запуску гри The Long Dark. Цей скріншот яскраво ілюструє одну з ключових переваг програмного комплексу SystemScope – можливість моніторингу стану системи під значним навантаженням.

Під час запуску та роботи ресурсоємних додатків, таких як сучасні ігри, апаратні компоненти (CPU, GPU, RAM) відчувають підвищене навантаження, а їхні температурні режими можуть суттєво змінюватися. "SystemScope" надає користувачеві актуальну інформацію про ці показники в реальному часі.

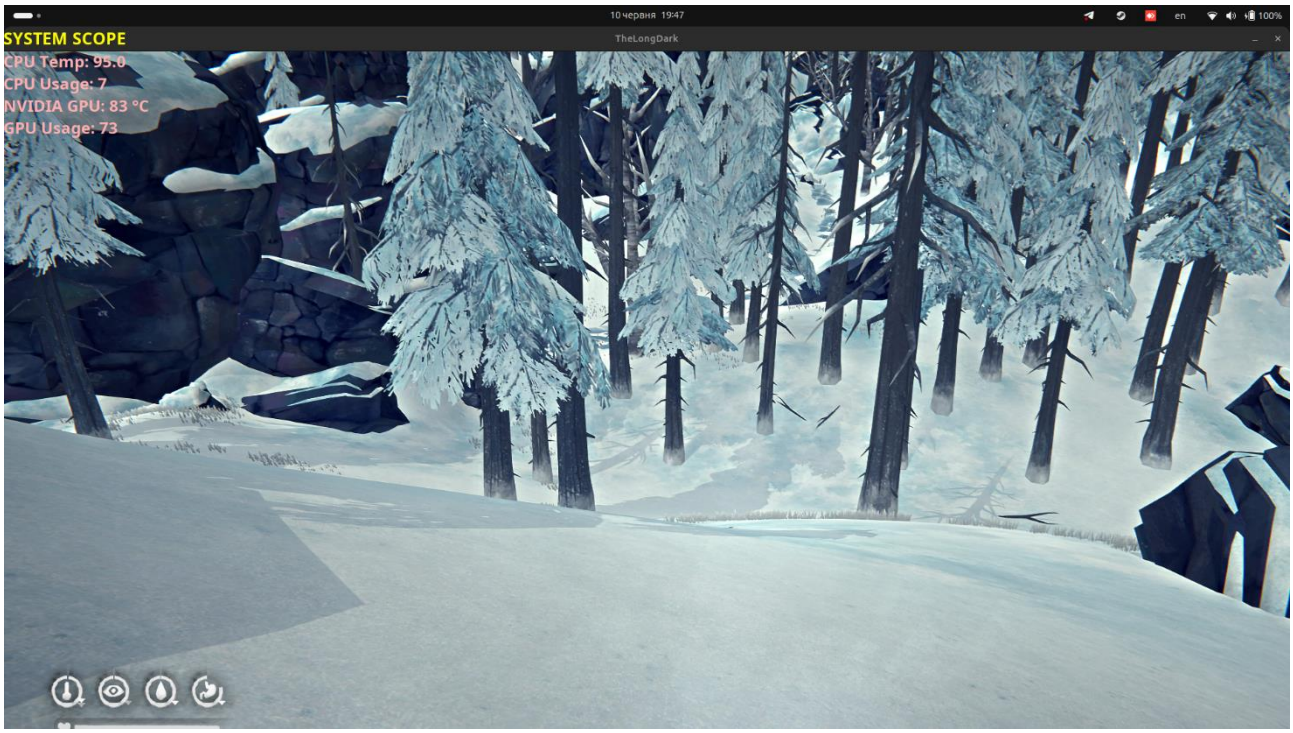

Рисунок 3.24 – Вікно бенчмарку під час запуску гри The Long Dark

Цей приклад демонструє практичну цінність "SystemScope" для геймерів та користувачів, які працюють з вимогливими до ресурсів програмами, забезпечуючи інструмент для оптимізації продуктивності та контролю за станом свого комп'ютера.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Надзвичайні ситуації, викликані пожежами, вибухами, техногенними та природними причинами

Розробка та експлуатація програмного забезпечення здійснюється в умовах, які можуть бути схильні до виникнення надзвичайних ситуацій (НС) різного характеру.

Надзвичайні ситуації (НС) – це порушення нормальних умов життя і діяльності людей, що призводить або може призвести до загибелі людей, значних матеріальних втрат та/або суттєвого погіршення стану довкілля.

НС класифікуються за характером походження на чотири основні класи, кожен з яких поділяється на групи та види. НС поділяються на загальнодержавний, регіональний, місцевий та об'єктовий рівні.

Надзвичайні ситуації техногенного характеру**:** Це НС, пов'язані з порушеннями у виробничій, транспортній, енергетичній сферах, а також з аваріями на інженерних мережах. Такими ситуаціями можуть бути:

- Аварії з викидом небезпечних хімічних, радіоактивних, біологічних речовин: Можливі при розміщенні робочого місця поблизу промислових об'єктів.
- Раптове руйнування споруд та будівель: Може статися внаслідок зносу конструкцій, порушення будівельних норм або зовнішніх впливів.
- Аварії на інженерних мережах: Відключення електроенергії, водопостачання, опалення, каналізації, аварії систем зв'язку та телекомунікацій.

Профілактичні дії:

- Ознайомлення з планами евакуації та цивільного захисту будівлі/організації.
- Забезпечення безперебійного живлення для критично важливого

обладнання.

- Дотримання інструкцій адміністрації або рятувальних служб у разі аварії.
- Надання першої допомоги постраждалим, якщо це можливо і безпечно.

Надзвичайні ситуації природного характеру: Це НС, спричинені небезпечними геологічними, метеорологічними, гідрологічними та іншими природними явищами. До таких ситуацій відносяться:

- Небезпечні геологічні явища: Зсуви, обвали, що можуть загрожувати будівлям та життям людей.
- Метеорологічні явища: Сильні буревії, шквали, снігопади, ожеледиця, зливи з ризиком підтоплення, блискавки. Можуть призвести до руйнувань, обривів електромереж.
- Природні пожежі: Лісові та торф'яні пожежі, що можуть спричинити задимлення, погіршення якості повітря.
- Інфекційна захворюваність людей: Епідемії, пандемії, що можуть призвести до значного погіршення умов життєдіяльності та обмежень.

Профілактичні заходи та дії:

- Регулярне відстеження прогнозів погоди та попереджень про НС.
- Відключення електроприладів від мережі під час грози.
- У разі землетрусу – сховатися у безпечному місці або евакуюватися з небезпечного місця.
- Дотримання санітарно-гігієнічних норм для профілактики інфекційних захворювань.

Надзвичайні ситуації соціально-політичного характеру: Це НС, пов'язані з протиправними діями терористичного та антиконституційного спрямування, що можуть загрожувати громадській безпеці та життю громадян. До них належать:

- Збройні напади, захоплення об'єктів: Терористичні акти, захоплення будівель, що створює пряму загрозу життю та здоров'ю людини.
- Встановлення вибухового пристрою в громадському місці: Загроза вибуху, що вимагає негайної евакуації та дотримання вказівок

правоохоронних органів.

- Виявлення застарілих боєприпасів: Знаходження вибухонебезпечних предметів часів минулих воєн, що вимагає негайного повідомлення ДСНС та правоохоронних органів.

Профілактичні заходи:

- Підвищена уважність до підозрілих предметів або осіб.
- Негайно повідомляти правоохоронні органи (102) або ДСНС (101) про будь-які виявлені підозрілі об'єкти чи ситуації.
- У разі загрози терористичного акту – діяти згідно з інструкціями правоохоронних органів, зберігати спокій, допомагати іншим, евакуюватися за вказівками.
- Навчання правилам поведінки при виявленні вибухонебезпечних предметів: не чіпати та повідомити фахівців про виявлення небезпечного предмету.

Надзвичайні ситуації воєнного характеру: В сучасних умовах НС воєнного характеру є реальною і значною загрозою для людини. Їхній вплив на безпеку життєдіяльності є першочерговим. Ці ситуації пов'язані з наслідками застосування зброї різного типу, що може призвести до масштабних руйнувань. Основні джерела небезпек у воєнний час:

1) Застосування зброї:
    - Зброя масового ураження: Ядерна, хімічна, біологічна зброя. Її використання призводить до масового ураження населення на великих територіях.
    - Звичайна зброя: Артилерійські обстріли, ракетні удари, авіабомбардування.
    - Засоби радіоелектронної боротьби: Хоча безпосередньо не руйнують споруди, можуть впливати на функціонування критичних систем зв'язку та навігації.
2) Антисанітарна обстановка та епідемії:

- Під час бойових дій порушується робота комунальних служб, що призводить до погіршення якості води, накопичення відходів.
- Масова загибель людей та тварин, неможливість своєчасного поховання, призводить до розповсюдження трупної отрути та інфекцій.
- Зростання популяції гризунів, комах та інших переносників небезпечних хвороб.
- Недостатнє медичне обслуговування, нестача медичних препаратів, що створює сприятливі умови для виникнення та поширення епідемій.

Війна є джерелом комплексних та взаємопов'язаних небезпек. Завданням безпеки життєдіяльності є максимально ефективний захист життя.

Управління ризиком НС базується на концепції прийнятного ризику, що вимагає виявлення небезпек, оцінки їх ймовірності та наслідків, а також розробки заходів для зниження ризику до припустимого рівня.

4.2 Оцінка технологічного процесу, обладнання, щодо умов електробезпеки, безпеки

Процес розробки програмного забезпечення, попри відсутність очевидних небезпек, пов’язаний із низкою факторів ризику, що впливають на фізичний, психічний та професійний стан працівника. Робоче місце програміста зазвичай є стаціонарним, а сам характер праці — сидячий, одноманітний, з тривалим використанням персональних електронно-обчислювальних машин, що зумовлює певні агрози.

Ризики у сфері ІТ значно більше пов’язані з біологічними, психологічними та організаційними чинниками**.**

Біологічні ризики пов’язані з тривалою статичною позою, недостатньою

фізичною активністю, що призводить до порушень кровообігу, розвитку захворювань хребта, суглобів, зору.

Психологічні ризики виникають через високий рівень концентрації, інформаційне перевантаження, дедлайни, монотонну роботу, що призводить до хронічного стресу, перевтоми, синдрому професійного вигорання.

Організаційні ризики пов’язані з поганою ергономікою робочого місця, відсутністю чітких регламентів на перерви, неврахуванням фізіологічних потреб працівників.

Особливу увагу слід звернути на вплив на зір**:** тривале перебування перед монітором викликає зорове перенапруження, "сухий синдром ока", а через 3–5 років регулярної роботи можливе зниження здатності сприймати кольори та контрастність, що особливо важливо для дизайнерів.

Також, слід враховувати екологічні фактори**:** кондиціонери без належного обслуговування можуть бути джерелом розповсюдження бактерій, а недостатня вентиляція — причиною накопичення вуглекислого газу, що погіршує концентрацію та самопочуття.

Електробезпека також є важливим аспектом організації робочого середовища. Комп’ютерна техніка під’єднується до електромережі з напругою 220 В, і при її несправності можливе виникнення коротких замикань, пробою ізоляції, або доторкання до струмопровідних частин. Основні джерела електротравматизму — несправні електроінструменти, відсутність заземлення, порушення правил експлуатації.

Основними причинами електротравматизму на виробництві є:

- випадкове доторкання до неізольованих струмопровідних частин електроустаткування;
- використання несправних ручних електроінструментів;
- робота без надійних захисних засобів та запобіжних пристосувань;
- доторкання до незаземлених корпусів електроустановок, що опинилися під напругою внаслідок пошкодження чи пробою ізоляції;

Проходячи через організм людини, електричний струм справляє на нього

термічну, електролітичну, механічну та біологічну дію.

Термічна дія струму спричинює опіки окремих ділянок тіла, нагрівання кровоносних судин, серця, мозку та інших органів.

Механічна дія струму загрожує ушкодженнями різноманітних тканин організму внаслідок електродинамічного ефекту.

Біологічна дія струму на живу тканину спричиняє небезпечне збудження клітин та тканин організму.

Характер впливу електричного струму на організм людини, а відтак і наслідки ураження, залежать від цілої низки чинників, які умовно можна поділити на чинники електричного та неелектричного характеру.

Сила струму, що проходить через тіло людини, є основним чинником, який обумовлює наслідки ураження. Різні за величиною струми справляють відповідний вплив на організм людини. Розрізняють три основних порогових значення сили струму:

- пороговий відчутний струм: найменше значення електричного струму, при проходженні якого через тіло людини виникають відчутні подразнення;
- пороговий невідпускаючий струм: найменше значення електричного струму, яке зумовлює судомні скорочення м'язів руки.
- пороговий фібриляційний струм: найменше значення електричного струму, що спричиняє при проходженні через тіло людини фібриляцію серця.

Пожежна безпека залишається актуальною. Можливими джерелами займання є перегрів компонентів комп’ютера, несправні блоки живлення або електромережа. У приміщенні мають бути встановлені первинні засоби пожежогасіння, продумані евакуаційні шляхи.

ВИСНОВКИ

У даній роботі мною було успішно розроблено та реалізовано програмний комплекс візуального моніторингу показників обладнання SystemScope на базі технології JavaFX. Виконане дослідження підтвердило актуальність обраної теми та досягнення поставлених цілей і завдань.

Було проведено аналіз існуючих рішень для моніторингу обладнання, що дозволило визначити вимоги до функціоналу та архітектури майбутнього програмного комплексу. На основі проведеного аналізу було розроблено архітектуру системи, що базується на принципах об'єктно-орієнтованого проєктування та із застосуванням патернів MVC для чіткого розділення логіки.

Було успішно реалізовано збір та відображення широкого спектру показників апаратного забезпечення, таких як завантаження та температура центрального процесора, обсяг оперативної пам'яті, стан дискових накопичувачів, а також детальні показники відеокарт різних. Для кросплатформного збору даних ефективно використовувалась бібліотека OSHI, а для специфічних показників GPU – системні утиліти.

Однією з ключових інновацій проєкту є інтеграція AI-модуля, що дозволяє користувачеві взаємодіяти з чат-ботом для отримання роз'яснень та рекомендацій на основі зібраних системних показників. Реалізація цього модуля включає керування історією чату, безпечну взаємодію з зовнішніми AI-сервісами через HTTP-запити та обробку JSON-відповідей.

Завдяки реалізації запланованих функціональних та нефункціональних вимог, розроблений програмний комплекс SystemScope є ефективним інструментом для візуального моніторингу стану обладнання. Він забезпечує користувачам актуальну інформацію, надає засоби для аналізу та діагностики, а також інтегрує сучасні технології штучного інтелекту для покращення надання допомоги.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Методичні вказівки до виконання дипломної роботи освітнього рівня “бакалавр” студентами усіх форм навчання для напряму підготовки 121 – Інженерія програмного забезпечення / М.Р. Петрик, Д.М. Михалик, Я.І. Кінах, С.В. Гладьо, Г.Б. Цуприк – Тернопіль: ТНТУ, 2016 – 28 с.

2. Сергій Г. Архітектура програмного забезпечення: все що треба знати. Wezom. URL: https://wezom.com.ua/ua/blog/arhitektura-programmnogo obespecheniya.

3. Петрик М. Р. Моделювання програмного забезпечення: науково-методичний посібник [Електронний ресурс] / М. Р. Петрик, О. Ю. Петрик. – 2015. – Режим доступу: http://elartu.tntu.edu.ua/handle/123456789/17796

4. UML [Електронний ресурс] – Режим доступу до ресурсу: https://www.uml.org/

5. Overview (JavaFX 24) [Електронний ресурс] – Режим доступу до ресурсу: https://openjfx.io/javadoc/24/.

6. oshi-core 6.8.2 API [Електронний ресурс] – Режим доступу до ресурсу: https://javadoc.io/doc/com.github.oshi/oshi-core/latest/index.html.

7. Java Language Specification [Електронний ресурс] – Режим доступу до ресурсу: https://docs.oracle.com/javase/specs/jls/se11/html/jls-1.html.

8. Maven Documentation [Електронний ресурс]. – 2025. – Режим доступу до ресурсу: https://maven.apache.org/guides/.

9. Apache Log4j [Електронний ресурс]. – 2025. – Режим доступу до ресурсу: https://logging.apache.org/log4j/2.x/.

10. Методичні вказівки для написання розділу “Безпека життєдіяльності, основи охорони праці” в кваліфікаційних роботах здобувачів освітнього ступеня ”бакалавр”. [Електронний ресурс] Режим доступу: https://elartu.tntu.edu.ua/bitstream/lib/35902/1/Metod._%20vkazivky_%20dlya_%20napysannnya_%20rozd._%20Bezp._%20zhyttyed._.pdf

11. Закон України «Про охорону праці». [Електронний ресурс] Режим доступу: https://zakon.rada.gov.ua/laws/show/2694-12

12. Сокуренко В.В. Безпека життєдіяльності та охорона праці : Підручник. Харків: Харків. нац. ун-т внутр. справ. 2021. 308 с

13. Бедрій Я.І. Основи охорони праці : навч. посіб. 4-е вид. перероб. і доп. Тернопіль : Навчальна книга – Богдан, 2018. 240 с

14. ДСТУ 4933:2008 Безпека у надзвичайних ситуаціях. Техногенні надзвичайні ситуації. Терміни та визначення основних понять.

15. ДСТУ 3994-2000 Безпека в надзвичайних ситуаціях. Надзвичайні ситуації природні. Чинники фізичного походження. Терміни та визначення.

16. ДСТУ EN 61140:2021 Захист від ураження електричним струмом. Загальні аспекти для установок та обладнання (IEC 61140:2016, IDT)

17. ДСТУ EN 50110-1:2017 Експлуатація електричних установок. Частина 1. Загальні вимоги (EN 50110-1:2013, IDT)

18. ДСТУ EN ISO 9241-5:2004 Ергономічні вимоги до роботи з відеотерміналами в офісі. Частина 5. Вимоги до компонування робочого місця та до робочої пози

19. Дистанційний курс «Кваліфікаційні роботи бакалаврів» сайту дистанційного навчання ТНТУ [Електронний ресурс]. – Режим доступу: URL: https://dl.tntu.edu.ua/bounce.php?course=5329

ДОДАТКИ

ДОДАТОК А. Повна діаграма класів згенерована помічником intellij idea ultimate

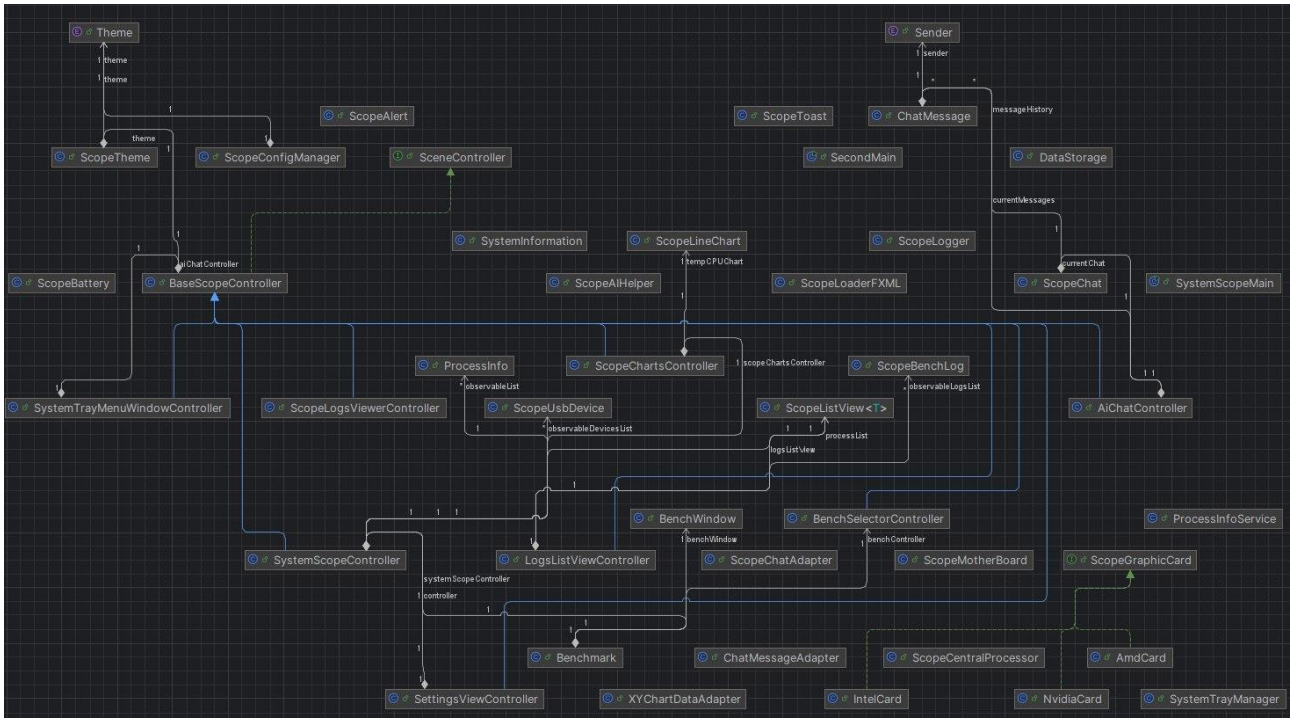

Рисунок А.1 – Діаграма класів згенерована в Intellij Idea Ultimate

ДОДАТОК Б. Тези для публікації на науково-технічну конференцію

УДК 004.4:004.93

Н. Муц – ст. гр. СП-42, док. філософії. І. Мудрик
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

РОЗРОБКА СИСТЕМИ ВІЗУАЛЬНОГО МОНІТОРИНГУ ПОКАЗНИКІВ КОМП'ЮТЕРНОГО ОБЛАДНАННЯ З ВИКОРИСТАННЯМ JAVAFX

UDC 004.4:004.93

N. Muts, Ph.D I. Mudryk

DEVELOPMENT OF A SYSTEM FOR VISUAL MONITORING OF COMPUTER HARDWARE PERFORMANCE USING JAVAFX

Ключові слова: JavaFX, моніторинг системи, штучний інтелект, OSHI, машинне навчання, інтерфейс користувача, аналіз температури, продуктивність ПК.

Keywords: JavaFX, system monitoring, artificial intelligence, OSHI, machine learning, user interface, temperature analysis, PC performance.

У даній роботі розглядається процес розробки системи моніторингу параметрів комп'ютерного обладнання із застосуванням технологій JavaFX та елементів штучного інтелекту. Основна мета проєкту полягає у створенні функціонального та зручного графічного інтерфейсу, що забезпечує візуалізацію стану комп'ютерної системи.

Проєкт включає в себе реалізацію модулів збору інформації, її обробки, збереження історичних даних, візуалізації, а також застосування методів машинного навчання для виявлення проблем у роботі обладнання. Збір системної інформації здійснюється з використанням бібліотеки OSHI (Operating System and Hardware Information), яка надає доступ до широкого спектру метрик.

Інтерфейс реалізований на основі JavaFX — сучасної платформи для створення графічних застосунків мовою Java. Вона дозволяє реалізовувати багатофункціональні інтерактивні елементи. Завдяки цьому користувач отримує не лише зручну візуалізацію, але й можливість взаємодії з інтерфейсом.

Окрему увагу приділено модулю штучного інтелекту, який використовує алгоритми машинного навчання. Наприклад, різке зростання температури у поєднанні зі зниженням продуктивності процесора може свідчити про початок перегріву або несправність охолоджувальної системи. Штучний інтелект здатний перевірити допустимі показники відносно переданих найменувань комп'ютерних комплектуючих та зробити висновки щодо ефективності роботи системи з наданими показниками.

У рамках розробки також реалізовано зберігання файлів логування тестів продуктивності в важких ситуаціях і історії вимірювань у форматі JSON, що дозволяє зручно експортувати дані, передавати їх в інші системи або проводити подальший аналіз. Програмний продукт є розширюваним і може бути адаптований під конкретні потреби користувача, а також масштабований для роботи з кількома пристроями одночасно.

У результаті реалізовано універсальну платформу для моніторингу, яка може бути застосована як у побутових, так і в корпоративних умовах, а також слугувати основою для подальших досліджень в області застосування штучного інтелекту в системному адмініструванні.

ДОДАТОК В. Лістинг коду модуля systeminformation

```
package nm.sc.systemscope.modules;

import nm.sc.systemscope.ScopeHardware.*;
import oshi.SystemInfo;
import oshi.hardware.*;

import java.util.ArrayList;
import java.util.stream.Collectors;
import java.util.List;

/**
 * The `SystemInfo` class provides methods for retrieving system
information, including details
 * about the graphics cards, RAM, disk storage, CPU, and other hardware
components.
 */
public class SystemInformation {
    private static final HardwareAbstractionLayer layer;

    static {
        SystemInfo systemInfo = new SystemInfo();
        layer = systemInfo.getHardware();
    }

    /**
     * Retrieves the names and vendors of the graphics cards in the system.
     *
     * @return a formatted string containing the names and vendors of the
graphics cards.
     */
    public static String getGraphicCards() {
        List<GraphicsCard> gpus = layer.getGraphicsCards();
        return formatGPU(gpus.stream()
                .map(gpu -> gpu.getName() + " (" + gpu.getVendor() + ")")
                .collect(Collectors.joining(", ")));
    }

    /**
     * Retrieves the total amount of RAM in the system.
     *
     * @return a formatted string with the total RAM size.
     */
    public static String getRAM() {
        long totalMemory = layer.getMemory().getTotal();

        return formatMemory(totalMemory);
    }

    /**
     * Retrieves detailed information about the physical memory (RAM) in
the system.
     * The method may not work if you do not have the appropriate access
```

```
rights to system folders.
     * @return a string with detailed information about each physical
memory module.
     */
    public static String getRamInfo(){
        try {
            List<PhysicalMemory> memoryList =
layer.getMemory().getPhysicalMemory();
            if (memoryList.isEmpty()) {
                ScopeLogger.logError("Physical memory information is
unavailable");
                return "Physical memory information is unavailable";
            }

            StringBuilder ramInfo = new StringBuilder();

            for (PhysicalMemory memory : memoryList) {
                ramInfo.append(String.format("Виробник: %s\n",
memory.getManufacturer()));
                ramInfo.append(String.format("Тип пам'яті: %s\n",
memory.getMemoryType()));
                ramInfo.append(String.format("Частота: %.2f GHz\n",
memory.getClockSpeed() / 1000.0));
                ramInfo.append(String.format("Об'єм: %s\n",
formatMemory(memory.getCapacity())));
            }

            return ramInfo.toString();
        }
        catch(Exception e){
            ScopeLogger.logError("Error while retrieving physical memory
information: {}", e.getMessage());
            return "Помилка при отриманні інформації про фізичну пам'ять: "
+ e.getMessage();
        }
    }

    /**
     * Retrieves information about the disk storage in the system.
     *
     * @return a string with the models and sizes of the disks.
     */
    public static String getDiskStorage() {
        List<HWDiskStore> disks = layer.getDiskStores();

        StringBuilder diskInfo = new StringBuilder();

        for (HWDiskStore disk : disks) {

            String diskModel = disk.getModel();
            long totalSpace = disk.getSize();

            diskInfo.append(String.format("Диск: %s, Об'єм: %s\n",
diskModel, formatMemory(totalSpace)));
        }
```

```
        return diskInfo.toString();
    }

    /**
     * Retrieves the discrete graphics card (GPU) from the system,
prioritizing NVIDIA or AMD.
     *
     * @return the discrete GPU if found, null otherwise.
     */
    private static GraphicsCard getDiscreteGPU() {
        List<GraphicsCard> gpus = layer.getGraphicsCards();

        for (GraphicsCard gpu : gpus) {
            String vendor = gpu.getVendor().toLowerCase();

            if (vendor.contains("nvidia") || vendor.contains("amd")) {
                return gpu;
            }
        }

        for (GraphicsCard gpu : gpus) {
            String vendor = gpu.getVendor().toLowerCase();
            if (vendor.contains("intel")) {
                return gpu;
            }
        }

        return null;
    }

    /**
     * Retrieves the temperature of the discrete GPU in the system.
     *
     * @return a string with the temperature of the discrete GPU, or "No
data" if not available.
     */
    public static String getTemperatureDiscreteGPU() {
        GraphicsCard gpu = getDiscreteGPU();

        StringBuilder temperatures = new StringBuilder();

        if(gpu != null) {

            String vendor = gpu.getVendor().toLowerCase();

            if (vendor.contains("nvidia")) {
                NvidiaCard nvidia = new NvidiaCard();
                temperatures.append(nvidia.getTemperature());
            } else if (vendor.contains("amd")) {
                AmdCard amd = new AmdCard();
                temperatures.append(amd.getTemperature());
            }
            else if(vendor.contains("intel")){
                IntelCard intel = new IntelCard();

                temperatures.append(intel.getTemperature());
            }
```

```
        }
        return !temperatures.isEmpty() ? temperatures.toString() : "Немає
даних";
    }

    /**
     * Retrieves the temperature of all GPUs in the system.
     *
     * @return a string with the temperatures of all GPUs, or "No data" if
not available.
     */
    public static String getTemperatureGPU() {
        List<GraphicsCard> gpus = layer.getGraphicsCards();

        StringBuilder temperatures = new StringBuilder();

        for (GraphicsCard gpu : gpus) {
            String vendor = gpu.getVendor().toLowerCase();

            if (vendor.contains("nvidia")) {
                NvidiaCard nvidia = new NvidiaCard();
                temperatures.append(nvidia.getTemperature());
            } else if (vendor.contains("amd")) {
                AmdCard amd = new AmdCard();
                temperatures.append(amd.getTemperature());
            } else if (vendor.contains("intel")) {
                IntelCard intel = new IntelCard();
                temperatures.append(intel.getTemperature());
            } else {
                temperatures.append("Невідомий виробник відеокарти:
").append(gpu.getName()).append("\n");
            }
        }

        return !temperatures.isEmpty() ? temperatures.toString() : "Немає
даних";
    }

    /**
     * Retrieves the model name of the computer.
     *
     * @return the computer's model name.
     */
    public static String getComputerName() {
        String model = layer.getComputerSystem().getModel();

        return model.split("_")[0].trim();
    }

    /**
     * Retrieves the fan speeds in RPM (Revolutions Per Minute).
     *
     * @return a string with the fan speeds or "Not found" if no fan data

 is available.
     */
    public static String getFansRPM(){
```

```
        int[] fanSpeeds = layer.getSensors().getFanSpeeds();
        if(fanSpeeds.length == 0){
            return "Не знайдено";
        }

        StringBuilder builder = new StringBuilder();
        for(int fan : fanSpeeds){
            builder.append(fan).append(" ");
        }

        return builder.append("RPM").toString();
    }

    /**
     * Retrieves the GPU usage percentage for the discrete GPU.
     *
     * @return a string with the GPU usage, or a message indicating no GPU
is found.
     */
    public static String getGPUUsage(){
        GraphicsCard gpu = getDiscreteGPU();

        if (gpu == null) {
            return "Дискретна відеокарта не знайдена";
        }

        String vendor = gpu.getVendor().toLowerCase();
        StringBuilder usage = new StringBuilder();

        if (vendor.contains("nvidia")) {
            NvidiaCard nvidia = new NvidiaCard();
            usage.append(nvidia.getGPULoad());
        } else if (vendor.contains("amd")) {
            AmdCard amd = new AmdCard();
            usage.append(amd.getGPULoad());
        } else if (vendor.contains("intel")) {
            IntelCard intel = new IntelCard();
            usage.append(intel.getGPULoad());
        } else {
            usage.append("Не підтримується виробником");
        }

        return !usage.isEmpty() ? usage.toString() : "Немає даних";
    }

    /**
     * Retrieves a list of USB devices connected to the system.
     *
     * @return a list of {@link UsbDevice} objects representing the USB
devices connected to the system
     */
    public static List<UsbDevice> getUsbDevices(){

        SystemInfo info = new SystemInfo();
        return info.getHardware().getUsbDevices(false);
    }
```

```
    /**
     * Converts a list of {@link UsbDevice} objects into a list of {@link
ScopeUsbDevice} objects.
     * Each {@link UsbDevice} in the input list is wrapped into a {@link
ScopeUsbDevice}.
     *
     * @param usbDevices the list of {@link UsbDevice} objects to be
converted
     * @return a list of {@link ScopeUsbDevice} objects representing the
converted USB devices
     */
    public static List<ScopeUsbDevice> getScopeUsbDevices(List<UsbDevice>
usbDevices){
        List<ScopeUsbDevice> devices = new ArrayList<>();

        for(UsbDevice device : usbDevices){
            devices.add(new ScopeUsbDevice(device));
        }

        return devices;
    }

    /**
     * Converts memory size from bytes to a human-readable format (GB).
     *
     * @param totalMemory the memory size in bytes.
     * @return a string representing the memory size in gigabytes.
     */
    private static String formatMemory(long totalMemory){
        double memoryInGB = totalMemory / (1024.0 * 1024.0 * 1024.0);
        return String.format("%.2f GB", memoryInGB);
    }

    /**
     * Formats the graphics card information.
     *
     * @param cards a string containing the graphics card details.
     * @return a formatted string with the graphics card names and models.
     */
    private static String formatGPU(String cards) {
        if (cards != null && !cards.isEmpty()) {
            String[] cardsList = cards.split(",");

            StringBuilder formattedCards = new StringBuilder();

            for (String card : cardsList) {
                card = card.trim();

                if (card.contains("Intel")) {
                    formattedCards.append("Intel UHD Graphics");

                } else if (card.contains("NVIDIA")) {
                    int startIndex = card.indexOf("[");
                    int endIndex = card.indexOf("]");

                    if (startIndex != -1 && endIndex != -1 && startIndex <
endIndex) {
```

```
                        startIndex += 1;
                        String model = card.substring(startIndex,
endIndex).trim();
                        model = model.replaceAll(" /.*", "");
                        formattedCards.append("NVIDIA ").append(model);
                    } else {
                        formattedCards.append(card);
                    }
                } else if (card.contains("AMD")) {
                    formattedCards.append("AMD
").append(card.replaceAll(".*\\[([A-Za-z0-9\\s]+)].*", "$1"));
                }

                if (!formattedCards.isEmpty() && cardsList.length > 1 &&
!card.equals(cardsList[cardsList.length - 1])) {
                    formattedCards.append("\n");
                }
            }

            String result = formattedCards.toString().trim();
            if (result.endsWith(" /")) {
                result = result.substring(0, result.length() - 2);
            }

            return result;
        }
        return "Відеокарти не знайдено";
    }
}
```

ДОДАТОК Г. Диск із кваліфікаційною роботою бакалавра