## Високорівневий опис реалізації майбутнього програмного рішення згідно розроблених вимог

1. **Архітектура програми:**

* Мови програмування: Програма реалізована з використанням Python та C++ для оптимальної продуктивності та швидкодії.
* Модульна структура: Кожен аспект моніторингу та аналізу ресурсів відокремлений у власні компоненти для полегшення розширення та керування.

1. **Моніторинг та аналіз ресурсів:**

* Використання системних викликів та бібліотек: Для збору даних про використання CPU, пам'яті, дискового простору та мережевого трафіку у реальному часі.
* Оптимізована структура даних: Зберігання даних відбувається з урахуванням оптимальної організації для забезпечення ефективного доступу та аналізу.

1. **Інтерфейс користувача:**

* Графічний інтерфейс: Розроблений з використанням Tkinter/ PyQt/ Kivy/ wxPython для зручності користувача.
* Налаштування інтерфейсу: Користувач може налаштовувати відображення показників та встановлювати порогові значення для сповіщень.

1. **Системні дані та журнали:**

* Використання системних API Windows: Для отримання доступу до системних подій та журналів, що дозволяє програмі відслідковувати системні зміни та помилки.

1. **Сповіщення та аналіз даних:**

* Механізми оповіщення ОС Windows: Повідомлення про перевищення порогових значень надсилаються користувачеві.
* Використання алгоритмів машинного навчання та статистичних методів: Для виявлення аномалій та рекомендацій щодо оптимізації використання ресурсів.

1. **Підтримка та поширення:**

* Підтримка операційної системи Windows: Програма підтримує версії Windows починаючи з Windows 7 для обох архітектур, 32-х та 64-бітних.
* Доступність для завантаження: Програма доступна для завантаження з веб-сайту розробника та розміщення у відповідних інтернет-магазинах.

## Структура програмного рішення; архітектуру класів, їх призначення і взаємодію, опис функцій

**1. Клас SystemMonitor:**

* **Призначення**: Головний клас, керує всіма аспектами моніторингу системи.
* **Функції**:
  + start\_monitoring(): Починає моніторинг системи.
  + stop\_monitoring(): Зупиняє моніторинг системи.
  + update\_system\_info(): Оновлює інформацію про систему.
  + set\_thresholds(): Встановлює порогові значення для сповіщень.

**2. Клас ResourceMonitor:**

* Призначення: Базовий клас для моніторингу ресурсів.
* Функції:
  + get\_usage(): Абстрактний метод для отримання використання ресурсу.
  + notify\_if\_threshold\_exceeded(): Повідомляє, якщо порогове значення використання ресурсу перевищено.

**3. Класи моніторів ресурсів (наприклад, CPUUsageMonitor, MemoryUsageMonitor, DiskSpaceMonitor, NetworkTrafficMonitor):**

* Призначення: Відповідають за моніторинг конкретного ресурсу.
* Функції:
  + get\_usage(): Отримує інформацію про використання конкретного ресурсу.
  + notify\_if\_threshold\_exceeded(): Повідомляє, якщо порогове значення використання ресурсу перевищено.

**4. Клас SystemEventMonitor:**

* Призначення: Відслідковує системні події та журнали.
* Функції:
  + get\_system\_events(): Отримує інформацію про системні події та журнали.
  + notify\_if\_critical\_event\_detected(): Повідомляє, якщо виявлено критичну подію або помилку.

**5. Клас UserInterface:**

* Призначення: Відповідає за користувацький інтерфейс програми.
* Функції:
  + display(): Відображає графічний інтерфейс.
  + customize(): Налаштовує інтерфейс користувача згідно його вимог.

**6. Клас NotificationManager:**

* Призначення: Керує сповіщеннями користувача.
* Функції:
  + send\_notification(): Надсилає сповіщення про перевищення порогових значень.

**7. Клас DataManager:**

* Призначення: Відповідає за зберігання та управління даними програми.
* Функції:
  + store\_data(): Зберігає дані для подальшого аналізу.
  + retrieve\_data(): Отримує збережені дані.

## Перша спроба реалізації програми

Реалізація програми знаходиться у файлах main.py та functions.py

Для справної роботи потрібно встановити залежності за допомогою команди

‘pip install -r re quirements.txt’

Запустити python скрипт за допомогою команди python main.py