

DOKUMENTACJA PROJEKTU

PC Asystent System Monitoring

Aleksandra Matysik 7424

Spis treści

Opis Projektu	4
OPIS PROJEKTU.....	4
GŁÓWNE FUNKCJONALNOŚCI	4
DIAGRAM UML USE CASE.....	5
Narzędzia i Technologie.....	6
JĘZYKI PROGRAMOWANIA	6
FRAMEWORKI I BIBLIOTEKI BACKENDOWE.....	6
FRAMEWORKI I BIBLIOTEKI FRONTENDOWE	6
BAZA DANYCH.....	7
NARZĘDZIA DO MONITOROWANIA SYSTEMU.....	7
Struktura Aplikacji	8
MODUŁ ODCZYTU DANYCH.....	8
MODUŁ PRZETWARZANIA DANYCH.....	8
MODUŁ API.....	8
MODUŁ WIDOKU WEB	8
MODUŁ ALERTÓW	9
MODUŁ BAZY DANYCH	9
Projektowanie Bazy Danych	11
WYMAGANIA DOTYCZĄCE BAZY DANYCH	11
MODEL KONCEPTUALNY BAZY DANYCH.....	11
NORMALIZACJA BAZ DANYCH	13
Pierwsza postać normalna (1NF).....	13
Druga postać normalna (2NF)	14
Trzecia postać normalna (3NF).....	14
Wynik normalizacji	14
PROJEKTOWANIE FIZYCZNE	14
Struktura tabel i relacje	14
Typy danych	14
Indeksy.....	15

DOKUMENTACJA PROJEKTU | PC Asystent System Monitoring

Partykjonowanie.....	15
Implementacja.....	16
BAZA DANYCH.....	16
BACKEND	16
FRONTEND.....	17
Opis Interfejsu Użytkownika.....	19
WYGLĄD INTERFEJSU	19
Kierunek Dalszych Działań.....	22

Opis Projektu

OPIS PROJEKTU

PC Asystent System Monitoring to aplikacja webowa, stworzona w Pythonie przy użyciu frameworka Flask, która umożliwia monitorowanie w czasie rzeczywistym kluczowych parametrów sprzętowych komputera. Zaprojektowana w celu monitorowania stanu i wydajności komputera.

Aplikacja umożliwia użytkownikom śledzenie kluczowych parametrów sprzętowych w czasie rzeczywistym, a także konfigurację progów ostrzeżeń i alarmów dla poszczególnych wskaźników.

Aplikacja przechowuje dane w relacyjnej bazie danych, która umożliwia efektywne zarządzanie danymi statycznymi oraz dynamicznymi. W projekcie uwzględniono projektowanie bazy danych, zbieranie danych systemowych oraz zapewnienie interfejsu API do ich pobierania.

GŁÓWNE FUNKCJONALNOŚCI

Monitorowanie parametrów systemowych

- Zbieranie danych o systemie z użyciem biblioteki psutil i GPUUtil.
 - Dane systemu operacyjnego,
 - Dane o procesorze,
 - Dane o pamięci (RAM i SWAP),
 - Dane o dyskach (wewnętrznych i zewnętrznych),
 - Dane o karcie graficznej.
 - Dane o połączeniach sieciowych,
 - Dane o użytkownikach,
 - Dane o zasilaniu.

System alertowy

- Powiadomienia desktopowe w przypadku przekroczenia progów bezpieczeństwa.
- Możliwość ustawienia progów przez użytkownika.
 - **Temperatura CPU i GPU:** Użytkownik może określić maksymalną dopuszczalną temperaturę procesora i karty graficznej, przy której zostanie wyświetcone ostrzeżenie lub alarm.
 - **Użycie CPU i RAM:** Możliwość kontrolowania poziomu zużycia procesora i pamięci operacyjnej, co pozwala zidentyfikować potencjalne problemy z wydajnością systemu.
 - **Poziom naładowania baterii:** Dla urządzeń przenośnych aplikacja informuje o niskim poziomie naładowania baterii, umożliwiając użytkownikowi podjęcie odpowiednich działań.
 - **Czas pracy systemu (uptime):** Funkcja monitorująca długość nieprzerwanego działania

systemu, co pozwala ocenić stabilność urządzenia.

- **Zajętość dysku:** Użytkownik może ustawić progi ostrzegawcze dla zajętości przestrzeni dyskowej, unikając problemów wynikających z braku miejsca na dane.

Wizualizacja danych

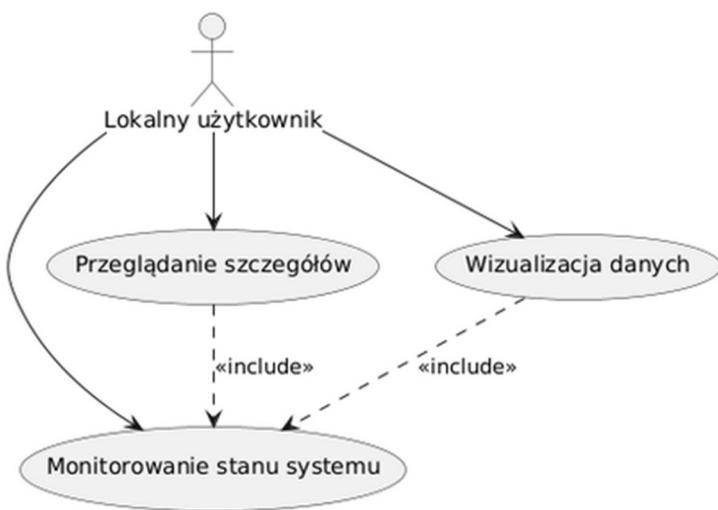
- Dynamiczne wizualizacje danych systemowych.

Zapis statycznych i dynamicznych danych do bazy danych.

- Statyczne dane (np. nazwa procesora, ilość rdzeni) zapisywane raz przy pierwszym uruchomieniu i po wykryciu zmiany.
- Dynamiczne dane (np. użycie CPU, temperatura GPU) zapisywane cyklicznie.

DIAGRAM UML USE CASE

W przypadku aplikacji monitorującej lokalny komputer, użytkownikiem może być tylko lokalny użytkownik systemu (np. administrator), który uruchamia aplikację na swoim komputerze, aby sprawdzić statystyki i wyniki monitorowania. Aplikacja nie ma potrzeby obsługi wielu użytkowników, a głównym celem jest wyświetlanie informacji o stanie systemu.



- Monitorowanie stanu systemu - użytkownik przegląda dane o CPU, GPU, pamięci, dys-kach, sieci, baterii.
- Przeglądanie szczegółów - użytkownik może zgłębiać szczegóły, takie jak temperatura CPU, użycie pamięci, użycie dysków itd.
- Wizualizacja danych - użytkownik może zobaczyć wykresy lub inne formy wizualizacji.

Narzędzia i Technologie

JĘZYKI PROGRAMOWANIA

Python

Główny język backendowy używany do obsługi logiki aplikacji oraz interakcji z bazą danych i systemem operacyjnym. Biblioteki takie jak *psutil* czy *py-cpuinfo* umożliwiają monitorowanie zasobów systemowych i sprzętowych.

Python zapewnia parsowanie danych wejściowych z formularza, przechowywanie i ładowanie konfiguracji z plików JSON, a także zapisywanie zmian w tych plikach.

HTML i CSS

Zastosowanie szablonów do renderowania dynamicznych stron w oparciu o dane w Pythonie.

JavaScript

Używany w połączeniu z bibliotekami frontendowymi do dynamicznej aktualizacji interfejsu użytkownika oraz wizualizacji danych

FRAMEWORKI I BIBLIOTEKI BACKENDOWE

Flask

Mikro-framework do budowania aplikacji webowych w Pythonie i serwerowania stron internetowych. Aplikacja zarządza żądaniami HTTP za pomocą routingu. Użycie metod *GET* i *POST* do obsługi formularzy i przekazywania danych między frontendem a backendem.

Zapewnia łatwą integrację z innymi bibliotekami, co pozwoliło na szybkie prototypowanie projektu.

Jinja2

Jinja2 to silnik szablonów, który Flask używa do renderowania dynamicznych stron HTML na podstawie danych przekazanych z backendu. Pozwala na wstrzykiwanie danych z backendu do szablonów HTML i ich dynamiczne renderowanie.

Psycopg2

To sterownik do komunikacji z bazą danych PostgreSQL. Używany do wykonywania zapytań SQL bezpośrednio z kodu Pythona.

FRAMEWORKI I BIBLIOTEKI FRONTENDOWE

Bootstrap

Bootstrap to framework CSS, który ułatwia tworzenie responsywnych i estetycznych stron

internetowych. Zawiera gotowe komponenty i klasy, które pomagają w tworzeniu profesjonalnych interfejsów użytkownika.

[jQuery](#)

Używana do obsługi dynamicznych elementów stron, takich jak interakcje użytkownika czy manipulacje DOM.

[Chart.js](#)

Biblioteka JavaScript do renderowania interaktywnych wykresów wizualizujących dane systemowe (np. użycie CPU czy RAM).

[BAZA DANYCH](#)

[PostgreSQL](#)

Relacyjna baza danych używana do przechowywania danych statycznych i dynamicznych systemów. Umożliwia przechowywanie i zarządzanie dużymi ilościami danych o wysokiej spójności i wydajności.

[NARZĘDZIA DO MONITOROWANIA SYSTEMU](#)

[psutil](#)

Biblioteka do monitorowania zasobów systemowych, takich jak procesor, pamięć RAM, dyski i sieci. Zapewnia dostęp do danych zarówno statycznych, jak i dynamicznych.

[py-cpuinfo](#)

Używana do pobierania szczegółowych informacji o procesorze (np. architektura, model, liczba rdzeni).

[GPUUtil](#)

Narzędzie do monitorowania parametrów GPU, takich jak zużycie pamięci czy temperatura.

[Matplotlib](#)

Używana do generowania wykresów w przypadku analizy danych.

Struktura Aplikacji

Aplikacja składa się z kilku głównych modułów, które współpracują ze sobą w celu zbierania, przetwarzania, wyświetlania i monitorowania danych systemowych.

MODUŁ ODCZYTU DANYCH

Moduł ten odpowiada za pobieranie danych systemowych. Bazuje na funkcjach w języku Python, korzystających z bibliotek takich jak *psutil*, *GUtil* czy *py-cpuinfo*. Obejmuje funkcje odpowiedzialne za odczyt parametrów systemu, CPU, RAM, GPU, dyski, sieci i użytkowników. Dane są zbierane w sposób dynamiczny, aby na bieżąco śledzić obciążenie systemu.

MODUŁ PRZETWARZANIA DANYCH

Odczytane dane są przekazywane do tego modułu, który odpowiada za przetwarzanie i organizowanie informacji w bardziej abstrakcyjne jednostki. Dane są segregowane w bloki takie jak:

- index (łączący platformy, użytkowników, energię),
- CPU,
- RAM,
- GPU,
- disks (dyski),
- networks (sieć).

Jest to abstrakcyjna warstwa aplikacji, nie znajdująca odzwierciedlenia w kodzie projektu. Jej zadaniem jest systematyzacja danych.

MODUŁ API

Moduł API pełni rolę mostu pomiędzy przetworzonymi danymi a widokiem aplikacji. Wystawia API, które udostępnia zebrane i przetworzone dane do dalszego wykorzystania przez warstwę widoku webowego. Dane organizowane są w api na podstawie modułu przetwarzania danych.

W tej warstwie znajduje się także *api_main_data*, które zbiera dane ze wszystkich komponentów i przekazuje je dalej. Ma kluczowe znaczenie dla jednostek projektu wymagających danych całościowych.

MODUŁ WIDOKU WEB

To główny komponent aplikacji odpowiedzialny za prezentację danych użytkownikowi. W tym module znajdują się szablony HTML i struktura interfejsu użytkownika, który wyświetla zebrane informacje w różnych podstronach aplikacji. Każda z podstron odpowiada za prezentację danych z różnych obszarów systemu:

- */cpu* – informacje o procesorze,
- */gpu* – informacje o karcie graficznej,
- */ram* – informacje o pamięci RAM,
- */disks* – informacje o dyskach,
- */network* – informacje o sieci,
- */config* – konfiguracja systemu alarmowego (zapewnia formularz konfiguracyjny).

MODUŁ ALERTÓW

Moduł odpowiedzialny za monitorowanie przekroczeń ustalonych progów alarmowych. Używa pliku konfiguracyjnego JSON, który zawiera domyślnie ustawione progi alarmowe. Użytkownik ma możliwość modyfikowania tych ustawień za pomocą formularza dostępnego w module widoku.

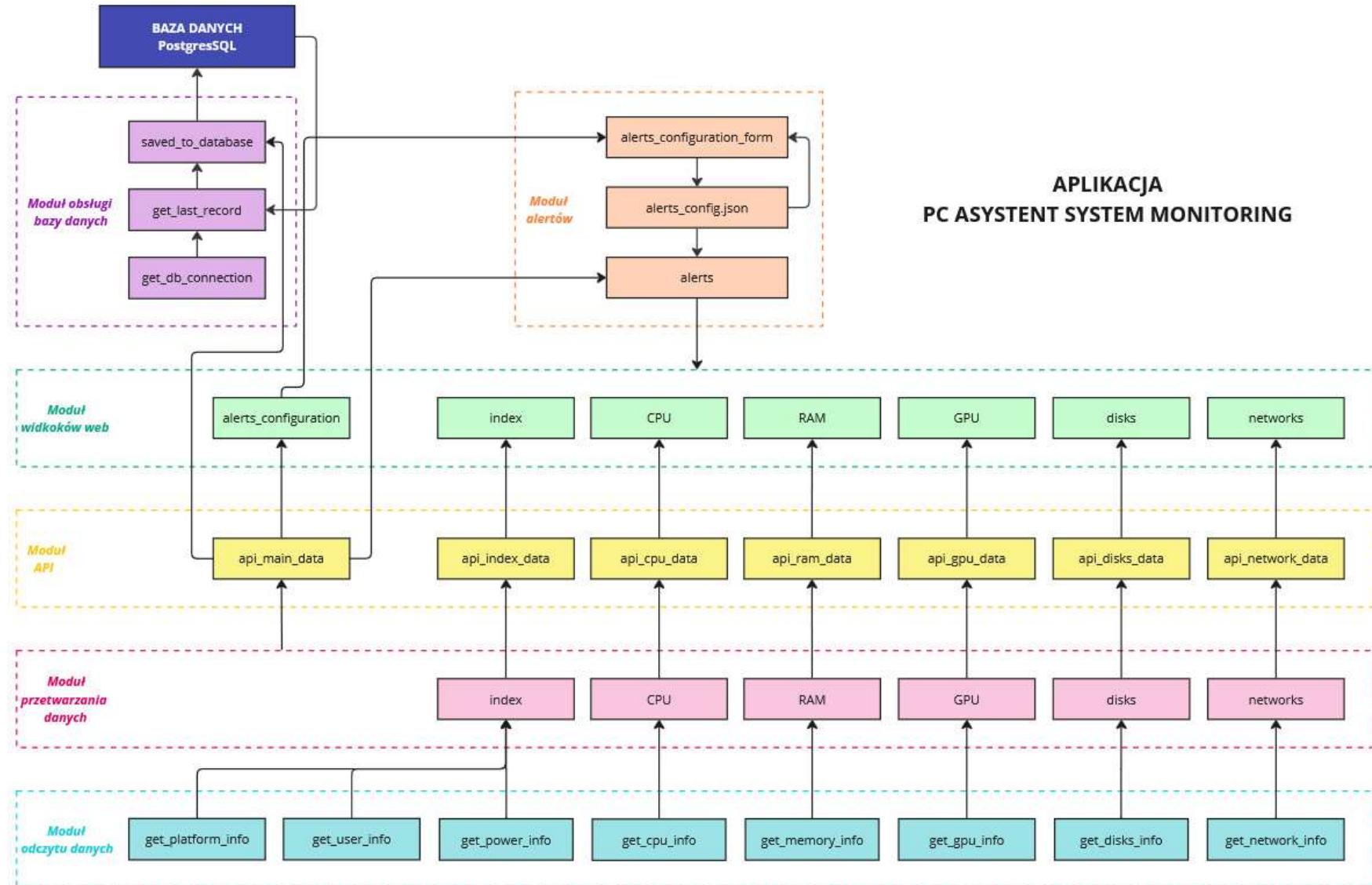
Moduł monitoruje, czy parametry systemowe nie przekraczają ustalonych granic, a jeśli takie przekroczenia wystąpią, wyświetla odpowiednie alerty. Analiza odbywa się za pomocą jQuery.

MODUŁ BAZY DANYCH

Aplikacja wykorzystuje bazę danych PostgreSQL do przechowywania danych o stanie systemu. Moduł ten zapewnia połączenie z bazą danych, zapisuje nowe rekordy oraz odczytuje ostatnie dane. Zapewnia również mechanizmy do przechowywania długoterminowych statystyk, które mogą być później analizowane.

Logika operacji na bazie danych jest bardziej zaawansowana, dlatego zostanie omówiona w oddzielnym rozdziale dokumentacji.

DOKUMENTACJA PROJEKTU | PC Asystent System Monitoring



Projektowanie Bazy Danych

WYMAGANIA DOTYCZĄCE BAZY DANYCH

- **Relacyjność**

Baza danych powinna pozwalać na przechowywanie i zarządzanie danymi w tabelach z relacjami między nimi.

- **Struktura tabel**

Tabele muszą oddzielać dane statyczne (np. informacje o procesorze) od danych dynamicznych (np. użycie CPU w danym czasie).

- **Zbieranie danych**

Zbieranie danych dynamicznych powinno odbywać się cyklicznie, z częstotliwością, która może być konfiguracja w aplikacji lub gdy zostanie wykryta zmiana.

- **Historia danych**

Baza musi przechowywać dane historyczne dla późniejszej analizy, np. monitorowania wykorzystania CPU w czasie.

- **Wydajność**

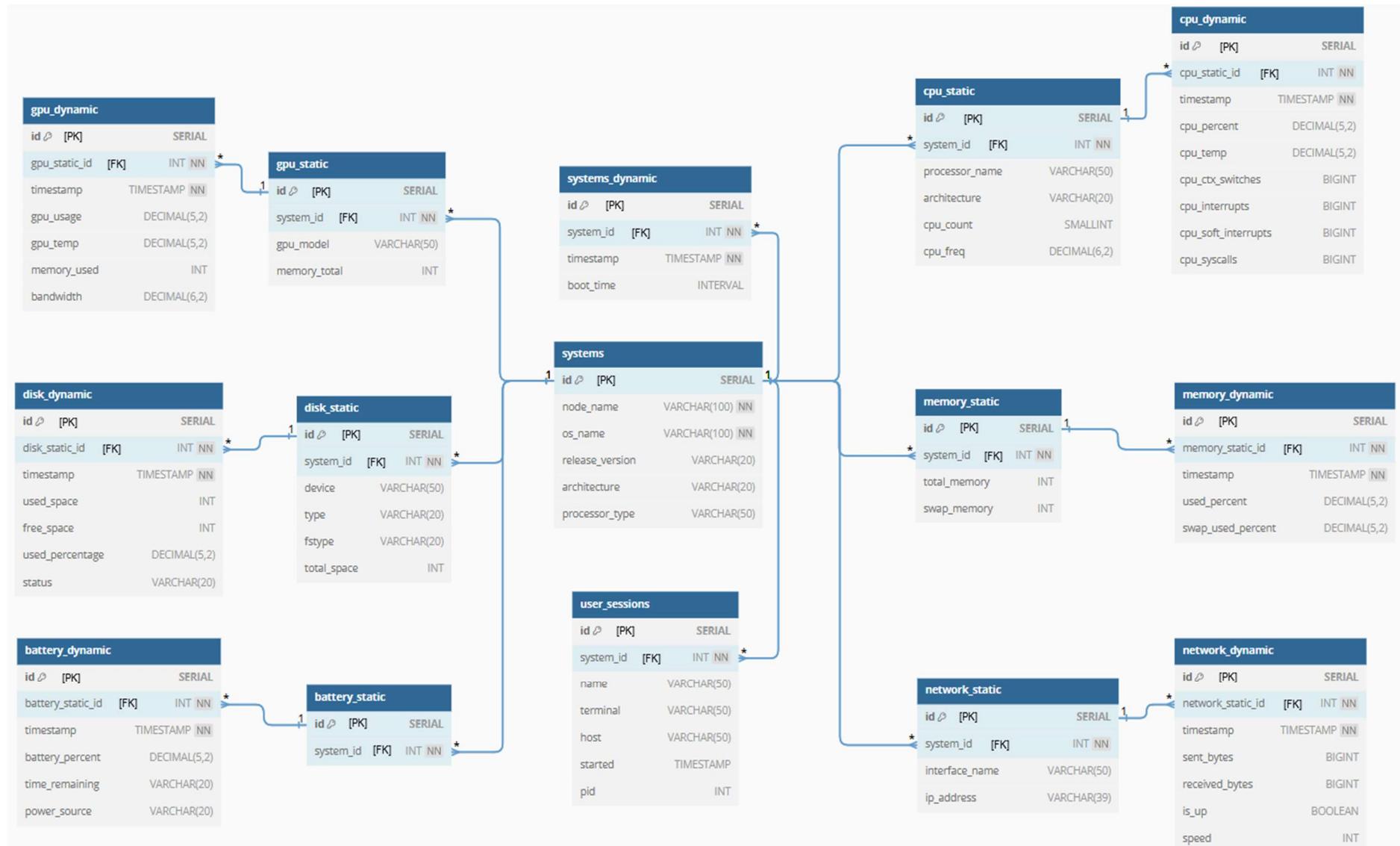
Baza danych powinna być zoptymalizowana pod kątem wydajności przy dużych zbiorach danych, z odpowiednimi indeksami na kolumnach czasu i identyfikatorach.

MODEL KONCEPTUALNY BAZY DANYCH

Model konceptualny bazy danych obejmuje ogólną strukturę tabel oraz relacje między nimi.

Został zaprezentowany na diagramie ERD

DOKUMENTACJA PROJEKTU | PC Asystent System Monitoring



Główne tabele:

- **Historia danych**

Baza musi przechowywać dane historyczne dla późniejszej analizy, np. monitorowania wykorzystania CPU w czasie.

- **Tabela systems**

Przechowuje informacje ogólne o systemach, takie jak nazwa urządzenia, system operacyjny i jego wersja, architektura procesora, czy typ procesora. Pełni funkcję tabeli nadzędnej, do której odnoszą się pozostałe tabele.

- **Tabele statyczne**

cpu_static, gpu_static, memory_static, disk_static, network_static, battery_static:

Przechowują dane o sprzęcie i zasobach systemu, które zmieniają się rzadko lub wcale.

- **Tabele dynamiczne**

cpu_dynamic, gpu_dynamic, memory_dynamic, disk_dynamic, network_dynamic, battery_dynamic:

Przechowują dane zmieniające się w czasie, np. obciążenie procesora, temperaturę GPU, procent wykorzystania pamięci, procent zajętej przestrzeni dyskowej, ilość przesłanych danych sieciowych, czy stan naładowania baterii.

Dane te są połączone z odpowiadającymi im tabelami statycznymi oraz tabelą *systems*, co pozwala na dokładne powiązanie metryk dynamicznych z konkretnymi systemami.

- **Tabela user_sessions**

Przechowuje dane o sesjach użytkowników, takie jak nazwa użytkownika, terminal, host, czas rozpoczęcia sesji oraz proces (PID).

powiązana z tabelą *systems* w celu identyfikacji systemu, którego sesja dotyczy.

- **Relacje**

Każda tabela dynamiczna posiada klucz obcy wskazujący na odpowiednią tabelę statyczną.

Tabele statyczne są powiązane z tabelą *systems*, co pozwala na identyfikację systemu, do którego należą dane.

NORMALIZACJA BAZ DANYCH

Celem normalizacji jest uniknięcie redundancji danych oraz poprawa integralności. Nasza baza danych będzie normalizowana do trzeciej postaci normalnej (3NF).

Pierwsza postać normalna (1NF)

Każda kolumna w tabeli jest atomowa - zawiera tylko pojedyncze wartości (brak grup danych).

Dane statyczne, takie jak nazwa systemu, procent obciążenia CPU, czy model GPU, są przechowywane w pojedynczych kolumnach, bez zagnieżdżonych struktur.

Powtarzające się dane, np. dynamiczne metryki sprzętowe, są zorganizowane w oddzielnych wierszach każdej tabeli dynamicznej.

Druga postać normalna (2NF)

Wszystkie kolumny niekluczowe są zależne od całego klucza głównego.

Dane, które są specyficzne dla sprzętu lub zasobów, zostały podzielone na tabele statyczne (*_static) i dynamiczne (*_dynamic), eliminując redundancję danych.

Dynamiczne dane, które zmieniają się w czasie, odnoszą się do unikalnych elementów sprzętowych poprzez klucze obce do tabel statycznych.

Trzecia postać normalna (3NF)

Brak zależności przejściowych, czyli każda kolumna niekluczowa jest bezpośrednio zależna od klucza głównego, a nie od innych kolumn.

W tabelach dynamicznych każda kolumna opisuje bezpośrednio dane powiązane z tabelą statyczną i kluczem głównym.

Informacje, które są wspólne dla wielu tabel (np. identyfikacja systemu w tabeli systems), są przechowywane w jednej tabeli nadzędnej, do której odnoszą się inne tabele poprzez klucze obce.

Wynik normalizacji

- Baza danych została pozbawiona redundancji, np. dynamiczne dane są przechowywane oddziennie od statycznych informacji o sprzęcie.
- Gwarantuje spójność danych – każda zmiana, np. w nazwie systemu, wymaga edycji tylko w jednym miejscu (systems).
- Pozwala na łatwe dodawanie i usuwanie danych, np. nowego GPU w systemie lub jego dynamicznych metryk.
- Jest bardziej elastyczna – łatwo można rozszerzyć system o nowe elementy, np. dodając tabele dla innych komponentów sprzętowych lub metryk.

PROJEKTOWANIE FIZYCZNE

Struktura tabel i relacje

Każda tabela dynamiczna ma klucz obcy do tabeli statycznej, co pozwala na powiązanie zmieniających się danych z ich kontekstem sprzętowym.

Tabele statyczne są powiązane z tabelą systems, która pełni funkcję centralną w strukturze. Centralizacja pozwala łatwo identyfikować, który system generuje dane, a także umożliwia skalowalność – dodanie nowego systemu wymaga jedynie dodania wpisu w tabeli systems.

Typy danych

- **Klucze główne i klucze obce (INT, SERIAL)**

Kluczami głównymi są pola id w każdej tabeli, które zostały zdefiniowane jako SERIAL. SERIAL automatycznie generuje unikalne wartości, co ułatwia zarządzanie identyfikatorami bez

konieczności ręcznego ustawiania.

Rozmiar INT (4 bajty) jest wystarczający dla przewidywanej liczby rekordów w każdej ta-beli.

- **Typy tekstowe (VARCHAR)**

Kolumny takie jak node_name, processor_name, czy interface_name używają VARCHAR z ograniczonym rozmiarem. Ograniczenia rozmiaru (np. VARCHAR(255)) pomagają kontrolować użycie pamięci i pozwalają silnikowi bazodanowemu lepiej optymalizować zapy-tania.

Pola przechowujące potencjalnie długie dane, np. logi (przyszły plan rozwoju aplikacji), mogłyby używać TEXT, choć w obecnym modelu tego nie przewidziano.

- **Liczby zmiennoprzecinkowe (DECIMAL, FLOAT)**

Kolumny takie jak cpu_percent, gpu_usage, czy used_percentage używają DECIMAL. Po-zwala to na precyzyjne przechowywanie wartości procentowych z ograniczeniem liczby miejsc po przecinku.

W miejscach, gdzie pełna precyzja nie jest krytyczna (np. temperatura GPU), można rozważyć użycie FLOAT dla lepszej wydajności.

- **Daty i czas (TIMESTAMP, INTERVAL)**

Kolumny takie jak timestamp w tabelach dynamicznych używają TIMESTAMP. Pozwala to na dokładne śledzenie momentu zebrania danych.

W tabeli systems_dynamic, kolumna boot_time używa INTERVAL. Format INTERVAL jest idealny do przechowywania przedziałów czasowych, takich jak czas działania systemu od ostatniego uruchomienia.

- **Dane logiczne (BOOLEAN)**

W tabeli network_dynamic, pole is_up wykorzystuje typ BOOLEAN. Wartości logiczne (prawda/fałsz) są bardziej wydajne i intuicyjne w przypadku zmiennych, które mają dwa możliwe stany.

Indeksy

Klucz główny w każdej tabeli (id) domyślnie tworzy indeks unikalny i jest automatycznie indeksowany. Umożliwia to obsługę relacji i wyszukiwanie rekordów.

Indeksy zostały dodane na kolumnach kluczy obcych (system_id, cpu_static_id, itp.). Każda tabela dynamiczna ma klucz obcy do tabeli statycznej. Tabele statyczne są powiązane z tabelą systems, która pełni funkcję centralną w strukturze.

Partycjonowanie

W przypadku dynamicznych danych, tabelę można podzielić na partycje według timestamp, aby poprawić wydajność zapytań historycznych.

Implementacja

BAZA DANYCH

Baza danych została zaprojektowana w sposób, który zapewnia efektywne przechowywanie i zarządzanie danymi systemowymi zbieranymi przez aplikację monitorującą. Struktura bazy danych obejmuje zestaw tabel przechowujących zarówno dane statyczne, jak i dynamiczne dla różnych komponentów systemu, takich jak CPU, GPU, pamięć, dyski, sieć oraz bateria.

Każdy komponent systemu (np. CPU, GPU, pamięć) ma dwie główne kategorie danych: statyczne i dynamiczne. Tabele statyczne przechowują informacje, które nie zmieniają się w czasie, jak typ procesora, model karty graficznej czy liczba rdzeni CPU. Tabele dynamiczne zawierają dane, które zmieniają się regularnie, jak obciążenie CPU, zużycie pamięci, czy prędkość sieci.

Wszystkie tabele są połączone ze sobą za pomocą kluczy obcych, co zapewnia spójność danych i umożliwia łatwe pobieranie informacji o systemie w kontekście różnych komponentów (np. CPU i GPU powiązane z systemem operacyjnym). Relacje między tabelami są zaprojektowane tak, aby wspierały zarówno wstawianie nowych danych, jak i modyfikowanie istniejących.

Wymagania implementacyjne

Aby wdrożyć bazę danych dla aplikacji monitorującej, należy spełnić następujące wymagania:

- **PostgreSQL**

Baza danych musi być uruchomiona w systemie przy użyciu silnika PostgreSQL. Aplikacja wymaga połączenia z działającą bazą danych PostgreSQL, dlatego przed rozpoczęciem implementacji należy zainstalować i uruchomić PostgreSQL.

- **Tworzenie bazy danych**

Wszystkie tabele i struktury bazy danych są zdefiniowane w pliku [setup.sql](#). Aby je stworzyć, użytkownik musi uruchomić ten plik w systemie PostgreSQL. Plik zawiera pełną definicję bazy danych, w tym tabel statycznych i dynamicznych, które przechowują dane monitorujące (np. CPU, GPU, pamięć, dyski, sieć).

- **Tworzenie bazy danych**

Wszystkie tabele i struktury bazy danych są zdefiniowane w pliku [setup.sql](#). Aby je stworzyć, użytkownik musi uruchomić ten plik w systemie PostgreSQL. Plik zawiera pełną definicję bazy danych, w tym tabel statycznych i dynamicznych, które przechowują dane monitorujące (np. CPU, GPU, pamięć, dyski, sieć).

BACKEND

Backend aplikacji monitorującej jest odpowiedzialny za zbieranie, przechowywanie oraz przetwarzanie danych o stanie systemu. Został zaprojektowany w języku **Python**, z użyciem frameworka **Flask** do tworzenia API, które komunikuje się z bazą danych PostgreSQL.

Struktura backendu

- **Połączenie z bazą danych**

Backend korzysta z biblioteki [psycopg2](#), aby łączyć się z bazą danych PostgreSQL. Konfiguracja połączenia jest przechowywana w pliku [db_config.json](#), który zawiera dane dostępowe (host, port, użytkownik, hasło).

- **API i Endpoints**

API jest kluczowe do komunikacji między frontendem a backendem. Przy użyciu Flask, stworzone zostały różne endpointy (np. [/api/main_data](#)), które zwracają dane o statusie systemu i parametrach komputera.

- **Przetwarzanie danych**

Backend regularnie zbiera dane systemowe i zapisuje je do odpowiednich tabel w bazie danych (np. [cpu_dynamic](#), [memory_dynamic](#)). Zbieranie danych odbywa się za pomocą skryptów w Pythonie, które używają narzędzi takich jak [psutil](#) (do monitorowania CPU, pamięci, dysków) i [pynvml](#) (do monitorowania GPU).

- **Konfiguracja alertów**

Backend przechowuje konfigurację alertów w pliku [alerts_config.json](#). Zawiera on progi dla różnych parametrów monitorowanych systemów (np. obciążenie CPU, temperatura GPU), które są później wykorzystywane przez frontend do generowania powiadomień. Backend nie przetwarza samych alertów, ale umożliwia frontendowi zapis i odczyt konfiguracji.

FRONTEND

Frontend aplikacji monitorującej odpowiada za interakcję użytkownika z systemem, umożliwiając wizualizację danych o stanie systemu oraz konfigurowanie alertów. Został zaprojektowany przy użyciu **HTML**, **CSS** oraz JavaScript z wykorzystaniem frameworka **jQuery** do dynamicznego ładowania danych i obsługi interakcji z użytkownikiem. Ponadto, frontend korzysta z frameworka **Bootstrap**, co zapewnia responsywny design i umożliwia działanie aplikacji na różnych urządzeniach.

Struktura frontendu

- **Interfejs użytkownika**

Frontend składa się z responsywnego interfejsu użytkownika, który wyświetla dane systemowe, takie jak obciążenie CPU, zużycie pamięci, stan GPU i inne. Widok jest zorganizowany w formie tabel i wykresów, co ułatwia użytkownikowi monitorowanie stanu systemu w czasie rzeczywistym.

- **Ładowanie danych**

Frontend komunikując się z backendem za pomocą **AJAX**, regularnie pobiera dane o systemie z endpointów API. Zastosowanie AJAX pozwala na dynamiczne aktualizowanie danych bez przeładowywania strony, co zapewnia płynność działania aplikacji.

- **Alerty**

Frontend wykorzystuje bibliotekę jQuery do obsługi alertów i powiadomień. Na podstawie ustawionych w pliku konfiguracyjnym *alerts_config.json* progów dla poszczególnych parametrów systemowych, frontend generuje powiadomienia w przypadku przekroczenia określonych wartości (np. zbyt wysokie obciążenie CPU czy temperatura GPU).

- **Zarządzanie konfiguracją alertów**

Frontend umożliwia użytkownikowi konfigurowanie progów alertów poprzez odpowiedni formularz, który następnie zapisuje zmiany w pliku *alerts_config.json*.

- **Responsywność**

Dzięki zastosowaniu CSS oraz frameworków takich jak Bootstrap, aplikacja jest responsywna i działa na różnych urządzeniach (komputery, tablety, telefony), zapewniając wygodny dostęp do monitorowania systemu w dowolnym miejscu.

Opis Interfejsu Użytkownika

Interfejs użytkownika aplikacji monitorującej system został zaprojektowany z myślą o prostocie i intuicyjności. Aplikacja jest responsywna, dzięki czemu działa na różnych urządzeniach. Główne elementy interfejsu to:

- **Nawigacja górnna i boczna**

Górny pasek nawigacyjny zawiera logo aplikacji oraz nazwę, a boczny panel pozwala na łatwe przejście między różnymi podstronami.

- **Dynamiczne ładowanie danych**

Dzięki wykorzystaniu AJAX dane o systemie są regularnie aktualizowane bez konieczności przeładowania strony.

- **Alerty i powiadomienia**

W przypadku wykrycia problemów (np. zbyt wysoka temperatura CPU), system wyświetla odpowiednie powiadomienia w formie alertów typu "danger" lub "warning".

- **Responsywność**

Interfejs został zaprojektowany tak, aby dobrze wyglądał na komputerach, tabletach i telefonach.

Dodatkowo, aplikacja zawiera funkcję konfiguracji alertów, pozwalającą użytkownikowi na ustawienie progów ostrzeżeń dla różnych parametrów systemowych.

WYGLĄD INTERFEJSU

Poniżej załączone są zrzuty ekranu widoku aplikacji, prezentujące wygląd interfejsu użytkownika.

The screenshot displays the main interface of the PC Asystent System Monitoring application. At the top, there's a dark header bar with the title 'PC Asystent' and a search bar. Below it is a navigation bar with a blue highlighted section labeled 'Informacje Podstawowe'. The main content area is divided into several sections:

- Główne informacje:** Shows basic system details: Nazwa hosta: HP, System operacyjny: windows, Wersja OS: Windows v10, Architektura: 64bit, Procesor: Intel® Core™ i7-13700K, Czas działania: 11 days, 14:26:10.188303.
- Zasilanie:** Displays power status with a green plug icon and '100%' charge level. It shows Procent: 100%, Czas: Calculating, and Źródło zasilania: Zasilanie z sieci.
- Użytkownicy:** A table showing user activity: Nazwa użytkownika: Aleksandra, Terminal: null, Host: null, Rozpoczęto: 26.01.2025, 11:36:06, PID: null.
- Alerts:** Two notifications at the bottom right: a yellow 'WARNING' box stating 'Czas pracy systemu przekroczył 11 dni!' and a pink 'ALERT' box stating 'Dysk E:\ Niedostępny'.

DOKUMENTACJA PROJEKTU | PC Asystent System Monitoring

Procesor

Podstawowe informacje

Rodzaj procesora:	Intel®6 Family 6 Model 186 Stepping 3, GenuineIntel
Architektura CPU:	64bit

Wydajność

Liczba rdzeni fizycznych:	10
Częstotliwość CPU:	1700 Mhz
Obciążenie CPU:	1.9 %
Temperatura CPU:	N / A

Statystyki procesora

Przełączenia kontekstu:	587532548
Przerwania sprzętowe:	2453524771
Przerwania programowe:	0
Wywołania systemowe:	506021849

Pamięć RAM

Informacje o pamięci RAM

Calkowita pamięć RAM:	31.6G MB
Używana:	51.7%
Calkowita pamięć swap:	4.8G MB
Używana swap:	120.2%

Karta Graficzna

Informacje o GPU

Model GPU:	N/A
Użycie GPU:	N/A
Temperatura GPU:	N/A
Zużycie pamięci VRAM:	N/A
Przepustowość:	N/A

DOKUMENTACJA PROJEKTU | PC Asystent System Monitoring

Dyski

Urządzenie	Zamontowany	Typ	System plików	Całkowita pojemność	Używana przestrzeń	Wolna przestrzeń	Wykres
C:\	C:\	Wbudowany	NTFS	1.8T	616.6G	1.2T	
E:\	null	null	null	null	null	null	
F:\	null	null	null	null	null	null	

ALERT: Dysk E:\ Niedostępny X

ALERT: Dysk F:\ Niedostępny X

Sieci

Typ	Interfejs	Status	Adres IP	Przekonk. (Mbps)	Wyslane	Odebrane
Ethernet	Ethernet	Online	172.30.7.27	1000	1.9G	12.6G
Ethernet	Ethernet 2	Online	192.168.56.1	1000	0.0B	0.0B
Ethernet	Ethernet 3	Offline	169.254.84.83	1073	140.9M	0.0B
Ethernet	Ethernet 4	Online	10.252.74.176	4294	355.2K	1.7M
Loopback Pseudo-Interface	Loopback Pseudo-Interface 1	Online	127.0.0.1	1073	0.0B	0.0B
Łączność lokalna*	Połączenie lokalne* 1	Offline	169.254.180.96	0	0.0B	0.0B
Łączność lokalna*	Połączenie lokalne* 2	Offline	169.254.184.153	0	0.0B	0.0B
Łączność sieciowa Bluetooth	Polaczenie sieciowe Bluetooth	Offline	169.254.95.45	3	0.0B	0.0B
VMware Network Adapter	VMware Network Adapter VMnet1	Online	192.168.110.1	100	6.1K	0.0B
VMware Network Adapter	VMware Network Adapter VMnet8	Online	192.168.237.1	100	6.1K	0.0B
Wi-Fi	Wi-Fi	Offline	172.30.252.232	866	59.1K	1.2K

Konfiguracja Alarmów

Czas pracy systemu

Czas pracy systemu - Ostrzeżenie (dni):

Czas pracy systemu - Alarm (dni):

RAM

Zbyt wysokie użycie RAM (%):

Wysokie użycie RAM (%) - Ostrzeżenie:

Bateria

Poziom baterii poniżej (%):

Poziom baterii - Ostrzeżenie (%):

GPU

Temperatura GPU - Ostrzeżenie (°C):

Temperatura GPU - Alarm (°C):

CPU

Zbyt wysoka temperatura CPU (°C) - Ostrzeżenie:

Wysoka temperatura CPU (°C) - Ostrzeżenie:

Zbyt wysokie użycie CPU (%):

Wysokie użycie CPU (%) - Ostrzeżenie:

Dysk

Poziom zajętości dysku - Ostrzeżenie (%):

Poziom zajętości dysku - Alarm (%):

Zapisz zmiany

Kierunek Dalszych Działań

Rozszerzenie monitorowania maszyn wirtualnych (VM)

Z planem na wsparcie monitorowania maszyn wirtualnych, aplikacja mogłaby oferować pełne zarządzanie zasobami maszyn wirtualnych w środowiskach takich jak VMware, VirtualBox czy Hyper-V. Użytkownicy mogliby monitorować obciążenie CPU, RAM, przestrzeń dyskową oraz aktywność sieciową maszyn wirtualnych. Dodatkowo, aplikacja mogłaby dostarczać szczegółowe statystyki o wirtualnych urządzeniach oraz alarmować w przypadku wystąpienia problemów związanych z zasobami. Taki rozwój umożliwiłby zarządzanie systemem wirtualnym w ramach jednego interfejsu.

Analiza logów systemowych

Wprowadzenie analizy logów systemowych mogłoby znacząco poprawić diagnostykę problemów i umożliwić użytkownikowi łatwe śledzenie błędów systemowych, ostrzeżeń oraz informacji operacyjnych. Implementacja filtrów i narzędzi wyszukiwania logów mogłaby ułatwić identyfikację potencjalnych problemów w czasie rzeczywistym. Użytkownicy mogliby przeglądać logi według daty, typu błędu, aplikacji lub poziomu ważności. Dodatkowo, system mógłby analizować logi pod kątem występujących wzorców i generować raporty lub zalecenia naprawcze.

Monitorowanie urządzeń peryferyjnych

Kolejnym krokiem w rozwoju systemu mogłoby być dodanie wsparcia dla monitorowania urządzeń peryferyjnych, takich jak drukarki, skanery, kamery, urządzenia USB czy inne sprzęty zewnętrzne. Dzięki integracji z odpowiednimi protokołami, aplikacja mogłaby informować użytkownika o stanie tych urządzeń, ich wykorzystaniu oraz problemach, takich jak brak połączenia, uszkodzenie lub błędy sprzętowe. Taki rozwój pozwoliłby na pełniejsze monitorowanie całego środowiska pracy użytkownika.

Inteligentne powiadomienia i rekomendacje

Aplikacja mogłaby oferować bardziej zaawansowane powiadomienia, które nie tylko informują użytkownika o przekroczeniu parametrów sprzętowych, ale także sugerują możliwe działania naprawcze. Na przykład, w przypadku wykrycia wysokiego obciążenia procesora, aplikacja mogłaby zasugerować zamknięcie nieużywanych procesów lub przeprowadzenie optymalizacji systemu. Możliwość automatycznego generowania rekomendacji na podstawie zebranych danych systemowych zwiększyłaby efektywność użytkownika w reagowaniu na problemy.