ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Пристрій дистанційного моніторингу та керування ресурсами комп'ютера на базі мікроконтролера ESP32

Виконав: ст. гр. КІУКІ-21-6

Савченко О.О.

Науковий керівник:

ас. Аргунов В.В.

Мета роботи

- Використання мікроконтролера ESP32 для розробки пристрою для збору даних про навантаження комп'ютера, відсоток завантаженості процесора, пам'яті, завантаження дисків та мережі, список процесів виконуваних на комп'ютері) та їх надсилання на десктопний додаток, а також можливість дистанційного завершення процесів, керування живленням, запуску скриптів.

Аналіз предметної області

Огляд наявних інструментів моніторингу в різних операційних системах

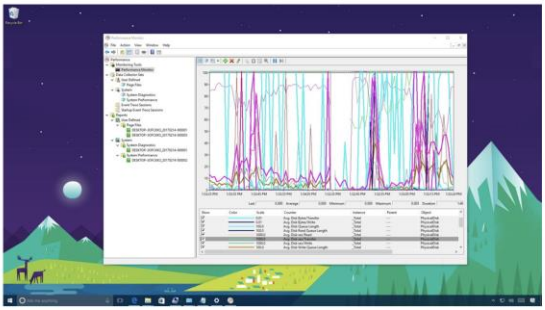


Рисунок 1 – Performance Monitor(Windows)

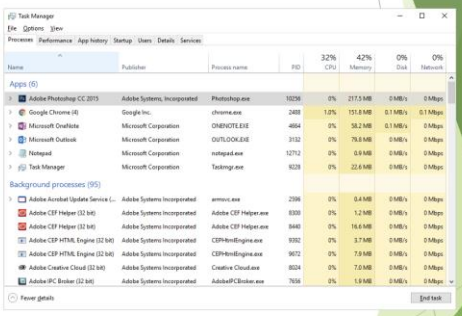


Рисунок 2 – Task Manager(Windows)

3

Аналіз предметної області

Огляд наявних інструментів моніторингу в різних операційних системах

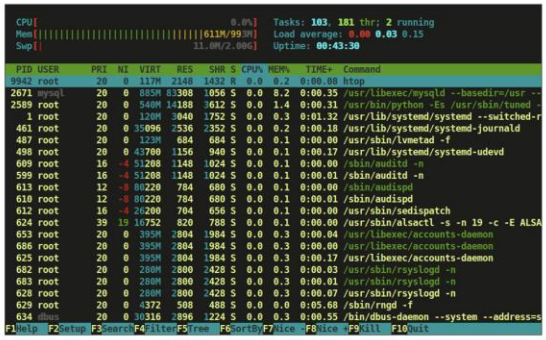


Рисунок 3 – htop(Linux)

4

Аналіз предметної області

Огляд наявних інструментів моніторингу в різних операційних системах

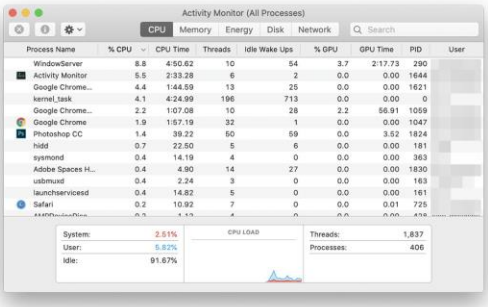


Рисунок 4 – Activity Monitor(MacOS)

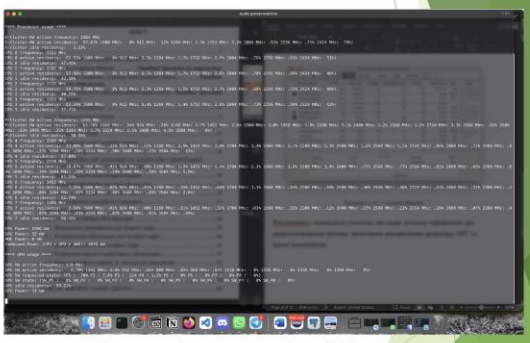


Рисунок 2 – powermetrics(MacOS)

Аналіз предметної області

Огляд інструментів дистанційного моніторингу(Remote Monitoring and Management, RMM)

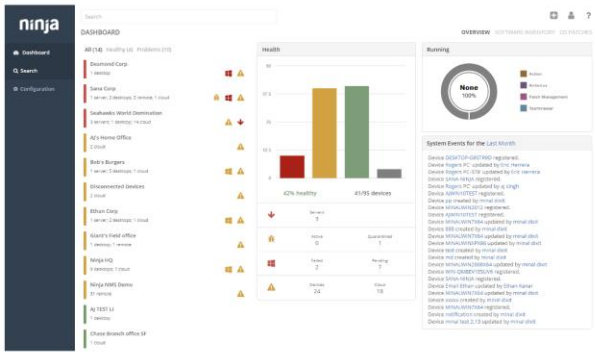


Рисунок 6 – NinjaOne

Аналіз предметної області

Огляд інструментів дистанційного моніторингу(Remote Monitoring and Management, RMM)

Рисунок 7 – ConnectWise Automate

Аналіз предметної області

Огляд інструментів дистанційного моніторингу(Remote Monitoring and Management, RMM)

Рисунок 8 – Atera RMM

Складання вимог та постановка завдання

Функціонал рішення

- Збір моніторингових даних з цільового комп'ютера
- Отримання загальних характеристик системи
- Передача моніторингових даних через зашифрований канал зв'язку до клієнта
- Відображення моніторингових даних у вигляді графіків та діаграм
- Збереження/Завантаження даних у файл
- Можливість надсилати команди на цільовий комп'ютер з метою знищення процесів та/або запуску скриптів
- Сповіщення користувача у разі виявлення аномального використання ресурсів
- Незалежність від основної системи

9

Опис інструментів розробки

Серверна частина

Клієнтська частина

10

Огляд використаних технологій

TCP Socket

- TCP/IP сокети є базовим механізмом обміну даними між пристроями в мережі, який широко використовується в сучасних комп'ютерних системах або вбудованих пристроях

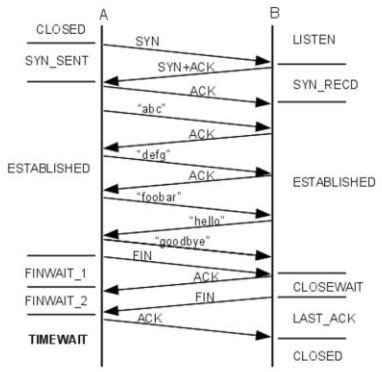


Рисунок 9 – Flow-діаграма TCP комунікації

Огляд використаних технологій

USB-to-UART та Serial

- UART — це апаратний модуль, який забезпечує асинхронну передачу даних без окремого тактового сигналу.

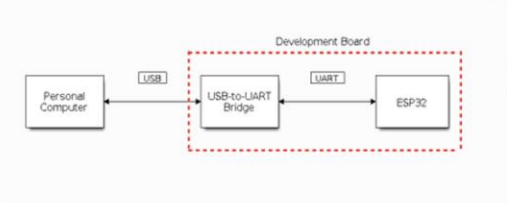


Рисунок 10 – Підключення до комп'ютера через вбудований USB-to-UART

Огляд використаних технологій

Формат даних JSON

- JSON (JavaScript Object Notation) - простий формат обміну даними, що є зручним як для читання та написання людиною, так і для парсінгу та генерації комп'ютером. Він базується на підмножині мови програмування JavaScript стандарту ECMA-262 3rd Edition - December 1999.

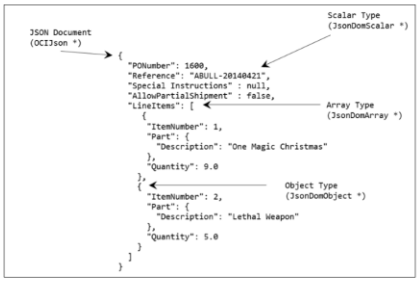


Рисунок 11 – Приклад формату JSON

Огляд використаних технологій

Transport Layer Security(TLS)

- TLS створює захищений канал зв’язку між клієнтом і сервером, запобігаючи перехопленню, підробці або модифікації даних під час передачі. У більшості випадків клієнтом є веббраузер, однак TLS може використовуватись і в інших типах клієнтів

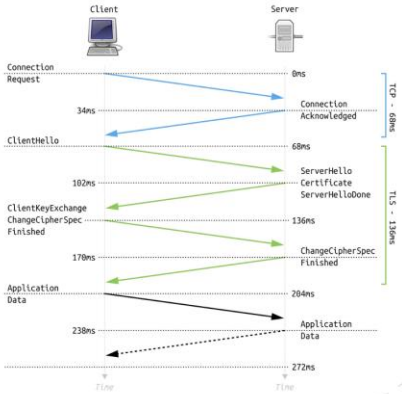


Рисунок 12 – Встановлення TLS сесії

Огляд використаних технологій

Технологія багатопоточності

- Сучасна комп'ютерна система зазвичай виконує велику кількість активних процесів та потоків. Це справедливо навіть для систем із єдиним ядром виконання, де в кожен момент часу реально працює лише один потік. Час процесора в таких системх розподіляється між процесами та потоками операційної системи, що зветься тайм-слайсінгом
- В сучасних реаліях є дуже поширенним використання комп'ютерних систем з кількома процесорами або процесорами, що мають декілька обчислювальних ядер. Це значно підвищує здатність системи до паралельного виконання процесів і потоків. Однак, паралелізм можливий навіть у простих системах без багатоядерних процесорів

15

Структура та реалізація серверної частини

Структура Proxy-скрипта на Python

Функції скрипта:

- Ідентифікація мікроконтролера
- Організація Wi-Fi підключення
- Виконання команд від мікроконтролера та повернення результату

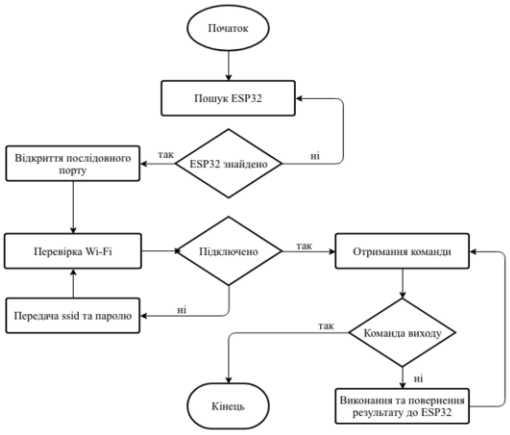


Рисунок 13 – Алгоритм роботи скрипта

16

Структура та реалізація серверної частини

Структура прошивки ESP32-сервера

Функціонал сервера:

- Відкриття послідовного з'єднання
- Підключення до мережі Wi-Fi
- Створення TLS-сокета
- Обробка отриманих від клієнта команд
- Отримання даних про стан системи
- Парсінг інформації та конвертація у JSON
- Надсилання інформації до клієнта
- Постійний моніторинг системи на збої

Список команд:

- /get-sys-info
- /start-monitor
- /stop-monitor
- /get-proc-info <pid>
- /kill-proc <pid>
- /reset-conn
- /execute <path>

17

Структура та реалізація серверної частини

Структура прошивки ESP32-сервера

Архітектура проєкту:

- /certs – SSL-сертифікат та ключ
- /headers – Структури для парсінгу
- /modules – модулі проєкту
- /handlers – обробники повідомлень
- /tls – модуль TLS-з'єднання
- /uart – модуль UART-з'єднання
- /wifi – модуль Wi-Fi-з'єднання
- main.cpp – вхідний файл проєкту

```
> tree
.
├── CMakeLists.txt
├── certs
│   └── dev
│       ├── server_cert.h
│       └── server_key.h
├── headers
│   ├── monitoringdata.h
│   └── systemdata.h
├── main.cpp
└── modules
    ├── handlers
    │   ├── client_message_handler.cpp
    │   ├── client_message_handler.hpp
    │   ├── system_data_request_handler.cpp
    │   ├── system_data_request_handler.hpp
    │   ├── system_info_request_handler.cpp
    │   └── system_info_request_handler.hpp
    ├── tls
    │   ├── tls_server.cpp
    │   └── tls_server.hpp
    ├── uart
    │   ├── uart_handler.cpp
    │   └── uart_handler.hpp
    └── wifi
        ├── wifi_manager.cpp
        └── wifi_manager.hpp

9 directories, 18 files
```

Рисунок 14 – Ієрархія файлів та директорій

18

Структура та реалізація серверної частини

Додаткові модифікації ESP32

Додавання живлення у вигляді акумулятора:

- Акумулятор підключений через плату захисту
- Дільник напруги на резисторах 10 kΩ та 18 kΩ
- Контакт GPIO4 використовується для детектування джерела живлення

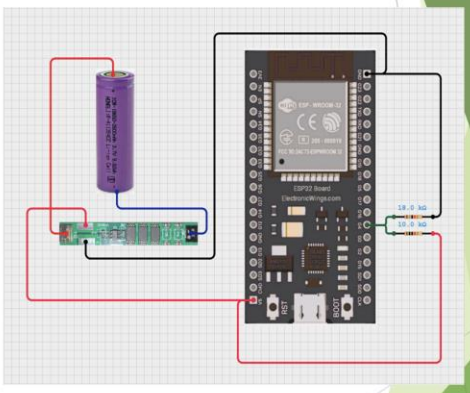


Рисунок 15 – Підключення компонентів

19

Структура та реалізація серверної частини

Додаткові модифікації ESP32

Таблиця розділів для запису прошивки

- Стандартний розмір пам’яті, виділеної під прошивку = 1Мб
- Розмір прошивки – 1079696 Байт
- Розмір пам’яті після використання таблиці розділів = 2Мб

Name	Type	SubType	Offset	Size
nvs	data	nvs	0x9000	0x5000
otadata	data	ota	0xe000	0x2000
app0	app	ota_0	0x10000	0x1E0000
spiffs	data	spiffs	0x1F0000	0x10000

Рисунок 16 – Таблиця розділів

20

Структура та реалізація клієнтської частини

Структура класів Java-додатку

Клієнтський додаток складається з пакетів:

- SystemInfo – інформація про систему
- SystemLoad – навантаження системи
- Tabs – вкладки для відображення інформації
- common – пакет з основними класами програми
- Net - пакет з класами що відповідають за підключення з сервером
- Ui – Пакет з основними елементами інтерфейсу користувача

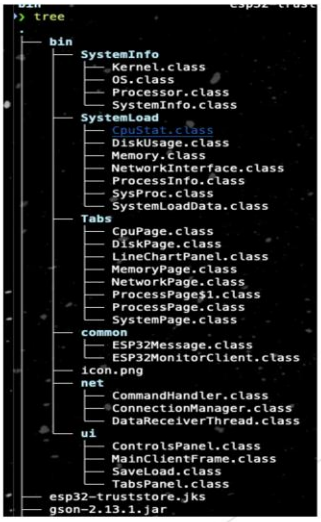


Рисунок 17 – Ієрархія файлів

Структура та реалізація клієнтської частини

Компоненти графічного інтерфейсу користувача

Графічний інтерфейс складається з:

- Основного клієнтського вікна
- Споміжних діалогових вікон

При старті програмі створюється діалогове вікно з полем введення IP-адреси пристрою та кнопками Connect та Exit

Рисунок 18 – Вікно підключення до пристрою

Структура та реалізація клієнтської частини

Компоненти графічного інтерфейсу користувача

Основне вікно програми містить елементи керування та інформаційну панель

Інформаційна панель розділена на вкладки, такі як System, CPU, Network, Disk та Processes

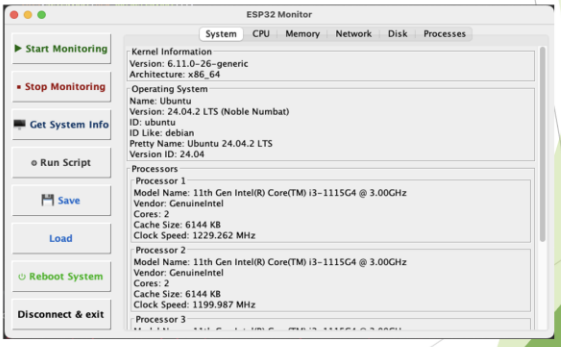


Рисунок 19 – Основне вікно програми

23

Структура та реалізація клієнтської частини

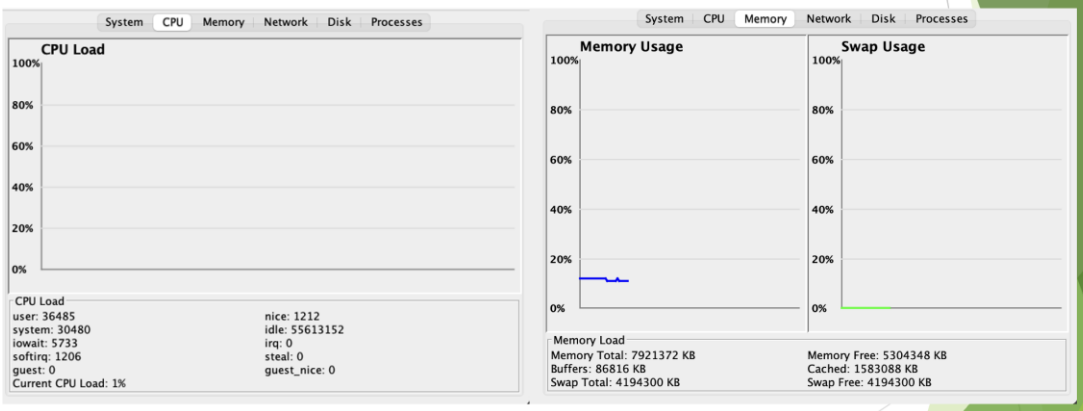


Рисунок 20 – Вкладки CPU та Memory

24

Структура та реалізація клієнтської частини

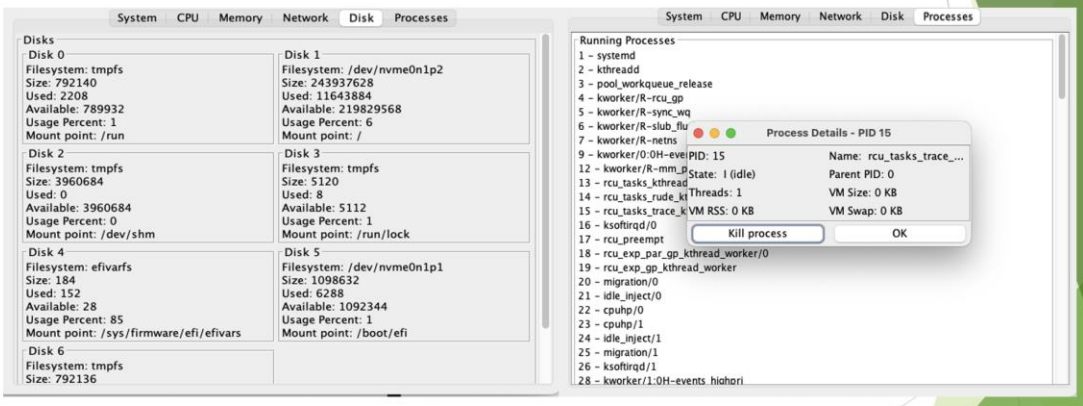


Рисунок 21 – Вкладки Disk та Processes

25

Структура та реалізація клієнтської частини

Розрахунок завантаженості системи

Оскільки сервер повертає завантаженість системи лише в поточний момент часу, для розрахунку проценту завантаженості процесора необхідно розрахувати час використання процесора(t_{total}) та час простою(t_{idle}) і, використовуючи дві точки вимірювання обчислити процент завантаженості

$$usage = \frac{\Delta t_{total} - \Delta t_{idle}}{\Delta t_{total}} * 100$$

$$MemUsage = \frac{M_{total} - M_{free}}{M_{total}} * 100$$

26

Інструкція користувача

Налаштування на цільовому комп'ютері

```
oleksandr@oleksandr-Swift-SF314-59:~/esp32/ESP32Monitor$ sudo python3 script.py
ESP32 detected on port /dev/ttyUSB0!

Initial response: WiFiDisconnected

Please write your WLAN Name:
```

Рисунок 22 – Запуск скрипта

```
I (274132) WiFi: Got IP: 192.168.178.74
I (274132) WiFi: Connected to Wi-Fi
Connected successfully
I (274132) main_task: Returned from app_main()
I (274142) TLS_SERVER: Listening on port 4433
```

Рисунок 23 – Результат підключення Wi-Fi

27

Інструкція користувача

Використання клієнтського додатку

Рисунок 24 – Підключення до пристрою

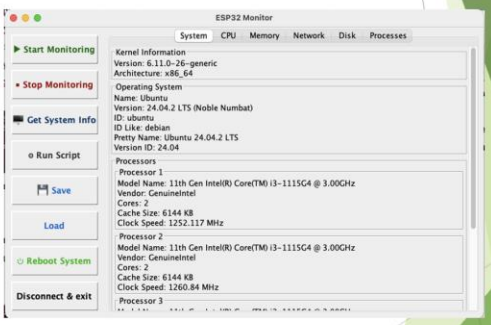


Рисунок 25 – Головне вікно

28

Інструкція користувача

Використання клієнтського додатку

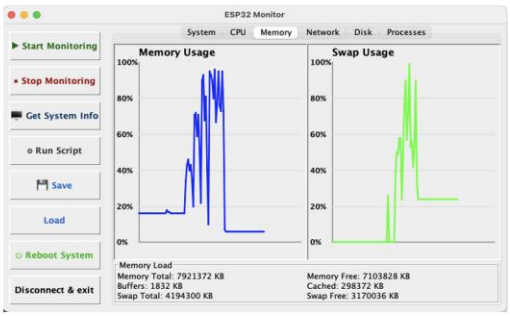


Рисунок 26 – Приклад моніторингу пам’яті

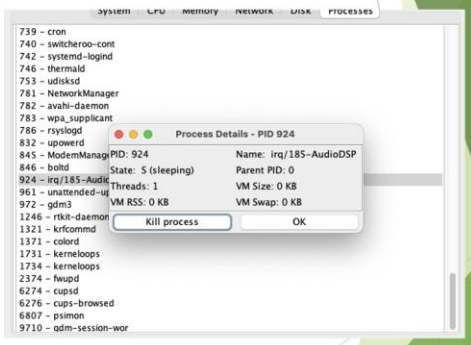


Рисунок 26 – Детальна інформація про процес

Інструкція користувача

Використання клієнтського додатку

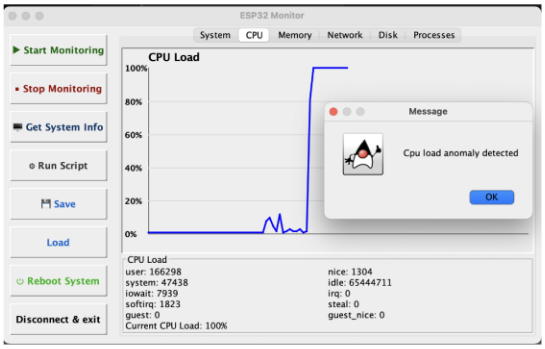


Рисунок 27 – Аномальне використання CPU

Рисунок 28 – Відсутність реакції системи

Рисунок 29 – Відсутність живлення

Висновки

У межах проекту було реалізовано:

- Створення Proxy-скрипта на Python
- Розробку серверної частини на мікроконтролері ESP32
- Створення клієнтського застосунку на Java
- Впровадження системи безпечного обміну даними через TLS та оптимізовану багатопоточну архітектуру

Порівняння з існуючими аналогами

- Моніторинг здійснюється незалежно від системи
- Пристрій має власне джерело живлення
- Застосунок має інтуітивно зрозумілий інтерфейс

31

Висновки

Практичне застосування системи

- Повсякденне адміністрування систем
- Використання у навчальному процесі

Подальша робота в цьому напрямку

- Розширення функціональності клієнтського застосунку
- Впровадження системи SMS або Email повідомлень
- Додавання більш гнучких налаштувань системи моніторингу
- Адаптація системи для використання на серверному обладнанні

32

ДОДАТОК Б

Простір імен systemdata

```
#pragma once
#include<string>
#include<cJSON.h>
#include<vector>
#include<map>

namespace systemdata
{
    struct kernel_info{
        std::string version;
        std::string architecture;
    };

    struct cpu_info {
        std::string model_name;
        int cores;
        int cache_size;
        double cpu_mhz;
        std::string vendor;
    };

    struct os_info {
        std::string name;
        std::string version;
        std::string id;
        std::string id_like;
        std::string pretty_name;
        std::string version_id;
    };

    struct system_info {
        kernel_info kernel;
        std::vector<cpu_info> cpu;
        os_info os;

        void serializeSystemInfo(cJSON* root) {
            cJSON_AddStringToObject(root, "datatype",
"system_info");

            cJSON* Jkernel = cJSON_CreateObject();
            cJSON_AddItemToObject(root, "kernel", Jkernel);
            cJSON_AddStringToObject(Jkernel, "version",
kernel.version.c_str());
            cJSON_AddStringToObject(Jkernel, "architecture",
kernel.architecture.c_str());
```

```
            cJSON* Jcpus = cJSON_CreateArray();
            cJSON_AddItemToObject(root, "processors", Jcpus);
            for (const auto& cpu : this->cpu) {
                cJSON* Jcpu = cJSON_CreateObject();
                cJSON_AddStringToObject(Jcpu, "model_name",
cpu.model_name.c_str());
                cJSON_AddNumberToObject(Jcpu, "cores",
cpu.cores);
                cJSON_AddNumberToObject(Jcpu, "cache_size",
cpu.cache_size);
                cJSON_AddNumberToObject(Jcpu, "cpu_mhz",
cpu.cpu_mhz);
                cJSON_AddStringToObject(Jcpu, "vendor",
cpu.vendor.c_str());
                cJSON_AddItemToArray(Jcpus, Jcpu);
            }

            cJSON* Jos = cJSON_CreateObject();
            cJSON_AddItemToObject(root, "os", Jos);
            cJSON_AddStringToObject(Jos, "name",
os.name.c_str());
            cJSON_AddStringToObject(Jos, "version",
os.version.c_str());
            cJSON_AddStringToObject(Jos, "id", os.id.c_str());
            cJSON_AddStringToObject(Jos, "id_like",
os.id_like.c_str());
            cJSON_AddStringToObject(Jos, "pretty_name",
os.pretty_name.c_str());
            cJSON_AddStringToObject(Jos, "version_id",
os.version_id.c_str());
        }
    };
}
```

ДОДАТОК В

Простір імен monitoringdata

```
#include<string>
#include<cJSON.h>
#include<vector>
#include<map>

namespace monitoringdata
{

    struct MemoryInfo {
        unsigned long MemTotal;
        unsigned long MemFree;
        unsigned long Buffers;
        unsigned long Cached;
        unsigned long SwapTotal;
        unsigned long SwapFree;
    };
    struct CPUStat {
        unsigned long user;
        unsigned long nice;
        unsigned long system;
        unsigned long idle;
        unsigned long iowait;
        unsigned long irq;
        unsigned long softirq;
        unsigned long steal;
        unsigned long guest;
        unsigned long guest_nice;
    };
    struct NetworkInterface {
        std::string name;
        unsigned long receive_bytes;
        unsigned long transmit_bytes;
    };
    struct ProcessInfo {
        int pid;
        std::string name;
        std::string state;
        int ppid;
        int threads;
        unsigned long vmsize;
        unsigned long vmrss;
        unsigned long vmswap;
        void serializeProcessInfo(cJSON* root)
        {
            cJSON_AddStringToObject(root, "datatype",
"process_info");

            cJSON* JProc = cJSON_CreateObject();
```

```
            cJSON_AddItemToObject(root, "process", JProc);
            cJSON_AddNumberToObject(JProc, "pid", pid);
            cJSON_AddStringToObject(JProc, "name",
name.c_str());
            cJSON_AddStringToObject(JProc, "state",
state.c_str());
            cJSON_AddNumberToObject(JProc, "ppid", ppid);
            cJSON_AddNumberToObject(JProc, "threads", threads);
            cJSON_AddNumberToObject(JProc, "vmsize", vmsize);
            cJSON_AddNumberToObject(JProc, "vmrss", vmrss);
            cJSON_AddNumberToObject(JProc, "vmswap", vmswap);
        }
    };
    struct DiskUsage {
        std::string filesystem;
        unsigned long size;
        unsigned long used;
        unsigned long available;
        unsigned short use_percent;
        std::string mount_point;
    };
    struct SystemLoadData {
        MemoryInfo memory;
        CPUStat cpu_stat;
        std::vector<NetworkInterface> network_interfaces;
        std::map<int, std::string> processes;
        std::vector<DiskUsage> disk_usage;
        void serializeMonitoringData(cJSON* root) {
            cJSON_AddStringToObject(root, "datatype",
"system_load_data");
            cJSON* JMemory = cJSON_CreateObject();
            cJSON_AddItemToObject(root, "memory", JMemory);

            cJSON_AddNumberToObject(JMemory, "mem_total",
memory.MemTotal);
            cJSON_AddNumberToObject(JMemory, "mem_free",
memory.MemFree);
            cJSON_AddNumberToObject(JMemory, "buffers",
memory.Buffers);
            cJSON_AddNumberToObject(JMemory, "cached",
memory.Cached);
            cJSON_AddNumberToObject(JMemory, "swap_total",
memory.SwapTotal);
            cJSON_AddNumberToObject(JMemory, "swap_free",
memory.SwapFree);

            cJSON* JCpu = cJSON_CreateObject();
            cJSON_AddItemToObject(root, "cpu_stat", JCpu);

            cJSON_AddNumberToObject(JCpu, "user",
cpu_stat.user);
            cJSON_AddNumberToObject(JCpu, "nice",
```

```
cpu_stat.nice);
            cJSON_AddNumberToObject(JCpu, "system",
cpu_stat.system);
            cJSON_AddNumberToObject(JCpu, "idle",
cpu_stat.idle);
            cJSON_AddNumberToObject(JCpu, "iowait",
cpu_stat.iowait);
            cJSON_AddNumberToObject(JCpu, "irq", cpu_stat.irq);
            cJSON_AddNumberToObject(JCpu, "softirq",
cpu_stat.softirq);
            cJSON_AddNumberToObject(JCpu, "steal",
cpu_stat.steal);
            cJSON_AddNumberToObject(JCpu, "guest",
cpu_stat.guest);
            cJSON_AddNumberToObject(JCpu, "guest_nice",
cpu_stat.guest_nice);
            cJSON* JNetifs = cJSON_CreateArray();
            cJSON_AddItemToObject(root, "network_interfaces",
JNetifs);
            for (const auto& iface : network_interfaces) {
                cJSON* JIface = cJSON_CreateObject();
                cJSON_AddItemToArray(JNetifs, JIface);
                cJSON_AddStringToObject(JIface, "name",
iface.name.c_str());
                cJSON_AddNumberToObject(JIface, "receive_bytes",
iface.receive_bytes);
                cJSON_AddNumberToObject(JIface,
"transmit_bytes", iface.transmit_bytes);
            }
            cJSON* Jprocs = cJSON_CreateObject();
            cJSON_AddItemToObject(root, "processes", Jprocs);
            for (const auto& [pid, name] : processes) {
                cJSON_AddStringToObject(Jprocs,
std::to_string(pid).c_str(), name.c_str());
            }
            cJSON* Jdisks = cJSON_CreateArray();
            cJSON_AddItemToObject(root, "disk_usage", Jdisks);
            for (const auto& disk : disk_usage) {
                cJSON* Jdisk = cJSON_CreateObject();
                cJSON_AddStringToObject(Jdisk, "filesystem",
disk.filesystem.c_str());
                cJSON_AddNumberToObject(Jdisk, "size",
disk.size);
                cJSON_AddNumberToObject(Jdisk, "used",
disk.used);
                cJSON_AddNumberToObject(Jdisk, "available",
disk.available);
                cJSON_AddNumberToObject(Jdisk, "use_percent",
disk.use_percent);
                cJSON_AddStringToObject(Jdisk, "mount_point",
disk.mount_point.c_str());
                cJSON_AddItemToArray(Jdisks, Jdisk);
            }
```

ДОДАТОК Г

Зміст файлу main.cpp

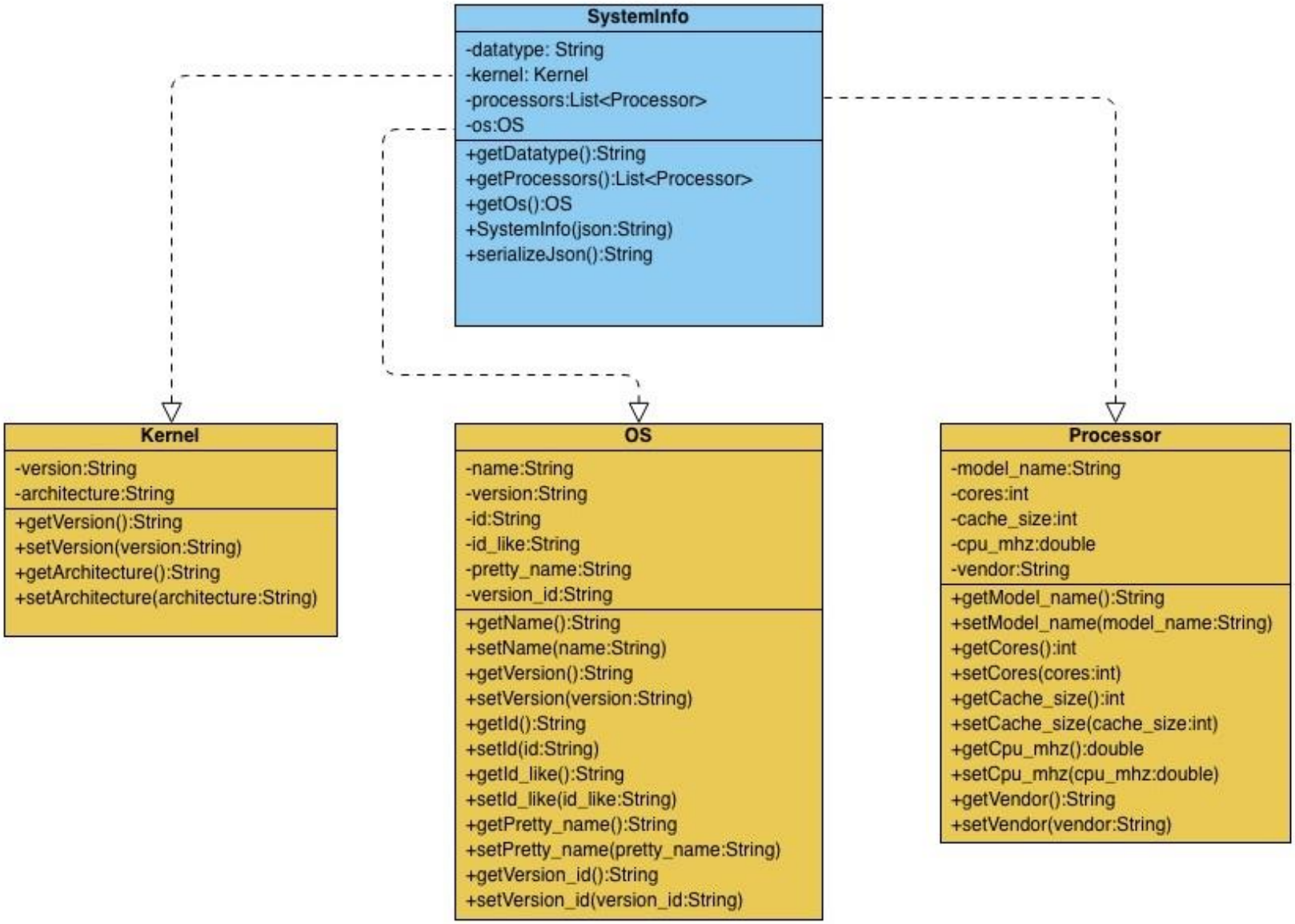
```
#include "esp_system.h"
#include "esp_wifi.h"
#include "esp_flash.h"
#include "nvs_flash.h"
#include "sdkconfig.h"
#include "esp_event.h"
#include <cJSON.h>

#include "modules/wifi/wifi_manager.hpp"
#include "modules/uart/uart_handler.hpp"
#include "modules/tls/tls_server.hpp"

#define LED_GPIO GPIO_NUM_2
#define RX_BUF_SIZE 512

TaskHandle_t monitor_task_handle = NULL;
mbedtls_ssl_context* monitor_ssl = NULL;
SemaphoreHandle_t ssl_mutex = NULL;
SemaphoreHandle_t serial_mutex = NULL;
EventGroupHandle_t s_wifi_event_group;
bool conn_timeout = false;
bool usb_disconnected = false;
bool system_timeout = false;

void device_state_task(void* arg);

// Main Application
extern "C" void app_main(void)
{
    ESP_ERROR_CHECK(nvs_flash_init());
    uart_init();
    gpio_set_direction(LED_GPIO, GPIO_MODE_OUTPUT);
    ssl_mutex = xSemaphoreCreateMutex();
    serial_mutex = xSemaphoreCreateMutex();
    if (ssl_mutex == NULL) {
        ESP_LOGE(TAG_SERVER, "Failed to create mutex");
        vTaskDelete(NULL);
    }
    xTaskCreate(&device_state_task, "device_state_task", 8192,
NULL, 5, NULL);
    while (true) {
        if(check_connection()) break;
        vTaskDelay(pdMS_TO_TICKS(10));
    }

    gpio_set_level(LED_GPIO, 1);
```

```
    xTaskCreate(&tls_server_task, "tls_server_task", 8192, NULL,
1, NULL);
}

void device_state_task(void* arg)
{
    char line[256];
    while (true) {
        int len = 0;
        if(xSemaphoreTake(serial_mutex, portMAX_DELAY)){
            len = uart_readline(line, sizeof(line));
            xSemaphoreGive(serial_mutex);
        }
        if (len > 0) {
            if (strcmp(line, "check_conn") == 0) {
                const char *response = check_connection()
                                       ? "WiFiConnected\n"
                                       : "WiFiDisconnected\n";
                uart_write_bytes(UART_PORT, response,
strlen(response));
            }
            if((std::string(line).find(",") !=
std::string::npos) && !check_connection())
            {
                ESP_LOGI(TAG, "Received line: %s", line);
                char *ssid = strtok(line, ",");
                char *password = strtok(NULL, ",");

                if (wifi_init_sta(ssid, password) == ESP_OK) {
                    ESP_LOGI(TAG, "Connected to Wi-Fi");
                } else {
                    ESP_LOGI(TAG, "Wi-Fi connection failed");
                }
            }
        }
        //if(!uart_check_connection()) ESP_LOGW("POWER", "NO
POWER ON GPIO4");
        if (!uart_check_connection() &&(monitor_ssl != NULL &&
ssl_mutex != NULL) && !usb_disconnected) {
                cJSON *root = cJSON_CreateObject();
                cJSON_AddStringToObject(root, "datatype",
"device_message");
                cJSON_AddStringToObject(root, "message", "Device
not powered from USB. Is system running?");
                char *json_str = cJSON_PrintUnformatted(root);
                cJSON_Delete(root);
                if (xSemaphoreTake(ssl_mutex,
pdMS_TO_TICKS(1000)) == pdTRUE) {
                    ssl_write_all(monitor_ssl, (const unsigned
char*)json_str, strlen(json_str));
                    xSemaphoreGive(ssl_mutex);
                }
                free(json_str);
```

```
                usb_disconnected = true;
            }
            vTaskDelay(pdMS_TO_TICKS(500));
        }
    }
```

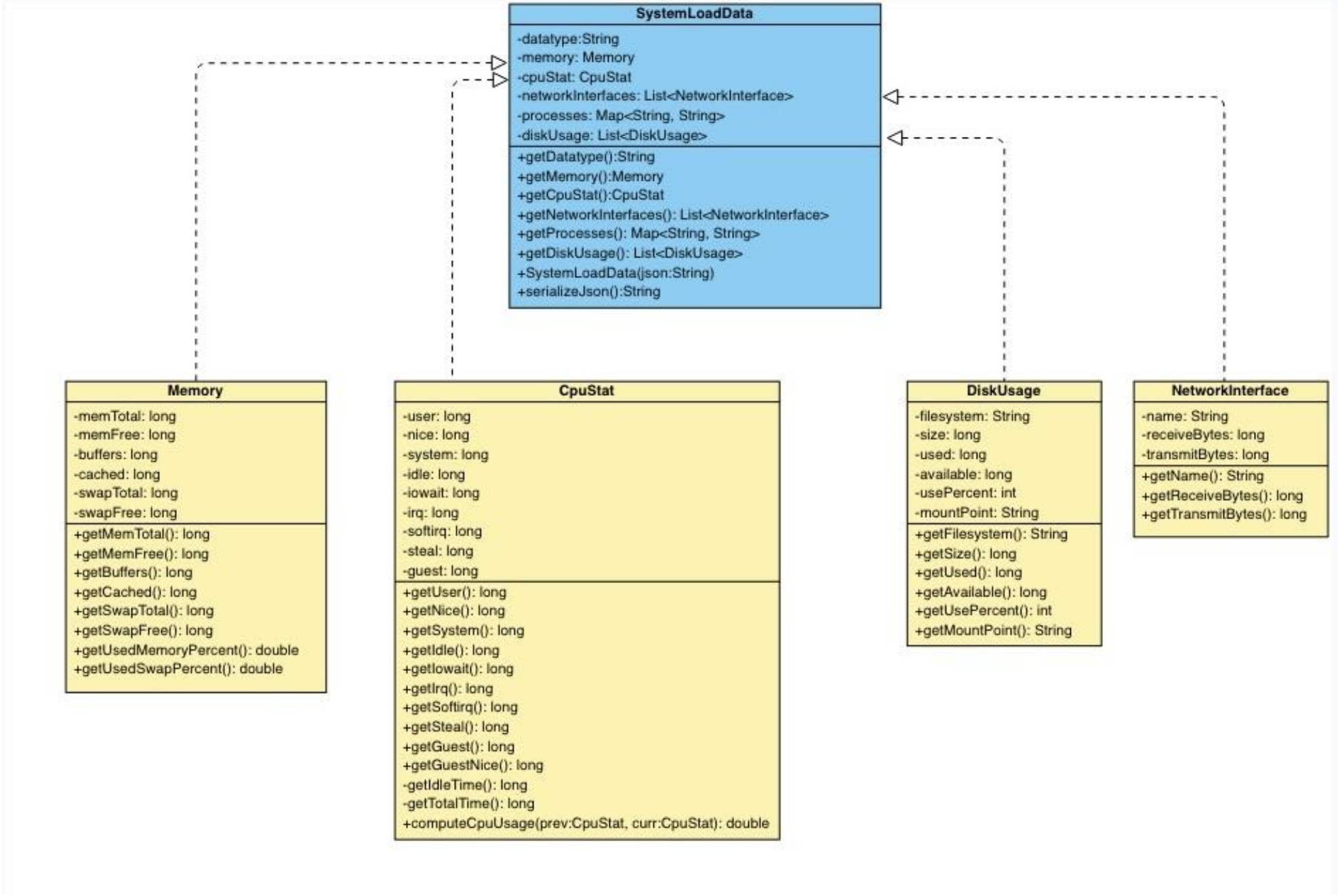
ДОДАТОК Д

UML-діаграма класу SystemInfo


SystemInfo
-datatype: String
-kernel: Kernel
-processors:List<Processor>
-os:OS
+getDatatype():String
+getProcessors():List<Processor>
+getOs():OS
+SystemInfo(json:String)
+serializeJson():String
Kernel
-version:String
-architecture:String
+getVersion():String
+setVersion(version:String)
+getArchitecture():String
+setArchitecture(architecture:String)
OS
-name:String
-version:String
-id:String
-id_like:String
-pretty_name:String
-version_id:String
+getName():String
+setName(name:String)
+getVersion():String
+setVersion(version:String)
+getId():String
+setId(id:String)
+getId_like():String
+setId_like(id_like:String)
+getPretty_name():String
+setPretty_name(pretty_name:String)
+getVersion_id():String
+setVersion_id(version_id:String)
Processor
-model_name:String
-cores:int
-cache_size:int
-cpu_mhz:double
-vendor:String
+getModel_name():String
+setModel_name(model_name:String)
+getCores():int
+setCores(cores:int)
+getCache_size():int
+setCache_size(cache_size:int)
+getCpu_mhz():double
+setCpu_mhz(cpu_mhz:double)
+getVendor():String
+setVendor(vendor:String)

ДОДАТОК Е

UML-діаграма класу SystemLoadData


SystemLoadData
-datatype:String
-memory: Memory
-cpuStat: CpuStat
-networkInterfaces: List<NetworkInterface>
-processes: Map<String, String>
-diskUsage: List<DiskUsage>
+getDatatype():String
+getMemory():Memory
+getCpuStat():CpuStat
+getNetworkInterfaces(): List<NetworkInterface>
+getProcesses(): Map<String, String>
+getDiskUsage(): List<DiskUsage>
+SystemLoadData(json:String)
+serializeJson():String
Memory
-memTotal: long
-memFree: long
-buffers: long
-cached: long
-swapTotal: long
-swapFree: long
+getMemTotal(): long
+getMemFree(): long
+getBuffers(): long
+getCached(): long
+getSwapTotal(): long
+getSwapFree(): long
+getUsedMemoryPercent(): double
+getUsedSwapPercent(): double
CpuStat
-user: long
-nice: long
-system: long
-idle: long
-iowait: long
-irq: long
-softirq: long
-steal: long
-guest: long
+getUser(): long
+getNice(): long
+getSystem(): long
+getIdle(): long
+getIowait(): long
+getIrq(): long
+getSoftirq(): long
+getSteal(): long
+getGuest(): long
+getGuestNice(): long
-getIdleTime(): long
-getTotalTime(): long
+computeCpuUsage(prev:CpuStat, curr:CpuStat): double
DiskUsage
-filesystem: String
-size: long
-used: long
-available: long
-usePercent: int
-mountPoint: String
+getFilesystem(): String
+getSize(): long
+getUsed(): long
+getAvailable(): long
+getUsePercent(): int
+getMountPoint(): String
NetworkInterface
-name: String
-receiveBytes: long
-transmitBytes: long
+getName(): String
+getReceiveBytes(): long
+getTransmitBytes(): long

ДОДАТОК Є

Посилання на повний код проєкту

https://drive.google.com/drive/folders/1NHKk7YcPR9p-YSH7erWzj5eMqyFn9ewE?usp=sharing