Inhalt

1. Einleitung

1.2 Projektziel

Ziel dieses Projekts ist es, ein nützliches und benutzerfreundliches Computerprogramm für Windows-Betriebssysteme zu entwickeln. Dieses Tool, das als "System-Monitoring-Tool" bezeichnet wird und den Projektnamen „py_sysmon“ trägt, soll folgende Anforderungen erfüllen:

Überwachung von Systemdaten

Es wird wichtige Leistungsindikatoren des Computers überwachen. Dazu gehören beispielsweise die Auslastung der CPU (die zentrale Recheneinheit des Computers) oder die Menge des verwendeten Arbeitsspeichers.

Darstellung der Daten

Das Programm wird in der Lage sein, die gesammelten Daten in einer übersichtlichen Weise grafisch darzustellen. Dies ermöglicht es den Benutzern, Muster leichter zu erkennen, wie zum Beispiel den Anstieg der CPU-Auslastung über einen bestimmten Zeitraum.

Benutzerbenachrichtigungen

Eine wichtige Funktion des Tools wird es sein, Benutzer zu benachrichtigen, wenn bestimmte vordefinierte Grenzwerte erreicht werden. Beispielsweise könnte das Programm eine Warnung aussenden, wenn der Speicherplatz des Computers knapp wird oder die CPU über einen bestimmten Zeitraum ausgelastet ist

1.3 Kontext des Projekts

Dieses Projekt ist ein Teil des schulischen Ausbildungsprogramms zum Fachinformatiker in der Anwendungsentwicklung. Es wird hauptsächlich an Berufsschultagen umgesetzt und findet überwiegend an der Berufsbildenden Schule (BBS) in Soltau statt.

1.4 Beweggrund

Ziel dieses Projekts ist es, auf die Erstellung der Projektdokumentation vorzubereiten, die ein wesentlicher Bestandteil der Ausbildung zum Fachinformatiker der Anwendungsentwicklung für die Industrie- und Handelskammer (IHK) ist.

2. Projektphasen

Das Projekt gliedert sich in drei Hauptphasen: Vorbereitung, Durchführung und Abschluss. Die zur Verfügung stehende Zeit umfasst die Berufsschultage vom 15.11.2023 bis zum 20.12.2023, was einer Gesamtbearbeitungszeit von ca. 34 Schulstunden entspricht.

Vorbereitungsphase (2 Stunden)

- Ist- und Soll-Analyse: Festlegung der genauen Anforderungen und Ziele der App.
- Technologieentscheidung: Wahl von Python und der Benutzeroberfläche.

Durchführungsphase (20 Stunden)

- Anforderungsanalyse: Klärung spezifischer Monitoring-Anforderungen und Nutzererwartungen.
- Design: Entwurf der Anwendungsarchitektur und der Benutzeroberfläche.
- Implementierung: Entwicklung der Überwachungsfunktionen (CPU, RAM, Festplattenplatz, etc.) und der Benutzeroberfläche.
- Testphase: Durchführung von Tests und Korrekturen.

Abschlussphase (12 Stunden)

- Soll-Ist-Vergleich (2 Stunden): Überprüfung, ob die App den gestellten Anforderungen entspricht.
- Dokumentation (10 Stunden): Erstellung einer ausführlichen Projektdokumentation für Benutzer und Entwickler (dieses Dokument bzw. das Github-Repository).

Bei der Entwicklung soll auf eine einfache und erweiterbare Struktur geachtet werden, um spätere Wartung und Erweiterungen zu erleichtern.
Es ist möglich, dass im Projektverlauf zusätzliche Funktionen oder Anforderungen hinzukommen.
Dieser Plan bietet einen strukturierten Überblick über das Projekt und ist darauf ausgerichtet, die Entwicklung innerhalb von ca. 34 Schulstunden erfolgreich abzuschließen.

3. Entwurf

In diesem Kapitel wird der Soll-Zustand, sowie die Technischen Voraussetzungen des Projekts festgehalten und dargestellt.

3.1 Werkzeuge/Technologien:

Beim Projekt kommen vor allem drei wichtige Entwicklungs-Tools zum Einsatz, das Versionierungssystem „Git“, die dazugehörige Plattform „Github“ und die Entwicklungsumgebung „Pycharm“.

Pycharm

PyCharm ist eine Entwicklungsumgebung speziell für Python. Es ist wie ein fortschrittlicher Texteditor, der speziell für das Programmieren ausgelegt ist. PyCharm bietet viele Hilfsmittel, um den Code zu schreiben, zu testen und zu debuggen. Es erleichtert die Programmierung, indem es Funktionen wie Code-Vervollständigung, Fehlererkennung und automatische Formatierung bietet. Es ist ein Werkzeug, das den Programmierprozess effizienter und fehlerfreier macht.

Git

Git ist ein Versionierungssystem, dass es Entwicklern ermöglicht, Änderungen am Code zu verfolgen und mit anderen zusammenzuarbeiten. Stellen Sie es sich wie eine Art Zeitmaschine für Programmierprojekte vor: Sie können Änderungen speichern, zu früheren Versionen zurückkehren und sehen, wer wann was geändert hat. Es ist besonders nützlich für Teams, um Konflikte zu vermeiden, wenn mehrere Personen am selben Projekt arbeiten.

GitHub

GitHub ist eine Plattform, die Hosting für Software-Entwicklungsprojekte bietet. Es verwendet Git für die Versionierung und ermöglicht es, Repositories online zu speichern („Remote Repositories“). GitHub erleichtert die Zusammenarbeit in Teams. Teammitglieder können auf gemeinsame Repositories zugreifen, Änderungen vornehmen und diese Änderungen mit anderen teilen. Ein zentrales Feature von GitHub sind Pull Requests. Sie ermöglichen es Entwicklern, Änderungen vorzuschlagen und zu diskutieren, bevor sie in den Hauptcode integriert werden. GitHub bietet auch Werkzeuge für das Verfolgen von Problemen („Issues“), was die Organisation und

Priorisierung von Aufgaben in einem Projekt erleichtert. Es existieren noch ähnliche Plattformen mit ähnlicher Funktionalität, ein populäres Beispiel wäre zb. GitLab.

3.2 Interaktion zwischen Git und GitHub

Push und Pull: Entwickler können Änderungen aus ihrem lokalen Repository auf GitHub „pushen“ (hochladen). Ebenso können sie Änderungen von GitHub „pullen“ (herunterladen), um lokal am neuesten Stand des Projekts zu arbeiten.

Code Reviews

Durch die Integration von Git in GitHub können Code-Änderungen durch andere Teammitglieder überprüft werden, bevor sie in den Hauptcode integriert werden.

Automatisierung

Viele Entwicklungsprozesse (wie das Ausführen von Tests oder das Deployen von Code) können durch die Integration von Git und GitHub automatisiert werden. Insgesamt bietet die Kombination aus Git und GitHub ein mächtiges Werkzeugset für moderne Softwareentwicklung, dass die Kollaboration, Codeverwaltung und Qualitätssicherung wesentlich verbessert. Das öffentliche Github-Repository dieses Projekts findet sich unter: **https://github.com/tf4482/py_sysmon**

4 Projektdurchführung

4.1 Setup

Zunächst erstellen wir mit PyCharm ein neues Projekt. Wir nennen es „py_sysmon“ und wählen als Interpreter die Python 3.12 die aktuelle, verfügbare Python-Version aus. Es ist „best-practice“, eine virtuelle Umgebung für Python-Projekte zu verwenden. Dies hilft, Abhängigkeiten und Paketversionen spezifisch für ein Projekt zu verwalten. PyCharm sollte dies im Übrigen automatisch tun, wenn ein neues Projekt erstellt wird.

Grundlegende Projektstruktur

Als erstes wird ein neues Verzeichnis „src“ im Projektverzeichnis erstellt. In diesem Verzeichnis werden alle Python-Dateien gespeichert, die für das Projekt von Nöten sind. Wir erstellen auch ein Verzeichnis „Tests“, in dem Tests abgelegt werden, außerdem ein Verzeichnis „docs“ für die Dokumentation.

src

Alle Python-Dateien, die für das System-Monitoring-Tool relevant sind, sollten hier abgelegt werden. Dazu gehören die Hauptanwendungsdatei (main.py), sowie Module, die verschiedene Funktionen des Tools umsetzen (z.B. Systemüberwachung, GUI-Elemente, Datenmanagement).

tests

Dieses Verzeichnis wird dazu verwendet, um Unit-Tests und Integrationstests für das Projekt zu schreiben.

docs

Hier wird die Dokumentation des Projekts gespeichert. Das sind zb. die Anforderungsspezifikation, die Architektur, die Benutzerdokumentation und die Entwicklerdokumentation.

Erstellen der ersten Python-Datei

Wir erstellen eine neue Python-Datei „main.py“ im Verzeichnis „src“. Diese Datei wird die Hauptanwendung des System-Monitoring-Tools enthalten.

Dependencies bzw. Bibliotheken

Um das Projekt zu starten, müssen wir einige Bibliotheken installieren. Wir planen folgende Bibliotheken zu verwenden:

- psutil: Eine Bibliothek, die es ermöglicht, Systeminformationen zu sammeln.
- matplotlib: Eine Bibliothek für die Diagrammerstellung.
- PyQt5: PyQt5 ist eine Bibliothek, welche für die Erstellung von GUIs verwendet wird.
- pyqtgraph: Wird benötigt, um die Diagramme in der GUI darzustellen.
- pyinstaller: Bibliothek zur Bündelung des Projekts.

Um diese Bibliotheken zu installieren, müssen wir die Datei „requirements.txt“ erstellen. Diese Datei enthält alle Bibliotheken, die für das Projekt benötigt werden. Wir können diese Datei mit dem Befehl „pip install -r requirements.txt“ installieren oder die PyCharm-Funktion „Install requirements“ nutzen. Diese requirements.txt wird üblicherweise im Hauptverzeichnis des Projekts gespeichert.

Versionskontrolle

Hoppla! Es wurde ganz vergessen ein Versionskontrollsystem zu initialisieren. Das ist aber kein Problem, denn wir können das ganz einfach nachholen. Dazu öffnen wir die Konsole und navigieren in das Projektverzeichnis. Dort führen wir den Befehl „git init“ aus.

Nun haben wir ein lokales Git-Repository erstellt. Wir können nun alle Dateien, die wir in unserem Projektverzeichnis haben, dem Repository hinzufügen. Dazu führen wir den Befehl „git add .“ aus. Der Punkt bedeutet, dass alle Dateien hinzugefügt werden sollen. Wenn wir nur bestimmte Dateien hinzufügen wollen, können wir diese auch explizit angeben. Zum Beispiel „git add main.py“. Auch erstellen wir ein Online-Repository bei GitHub. Dort können wir unser lokales Repository hochladen. Dazu müssen wir zunächst ein neues Repository erstellen. Wir nennen es „py_sysmon“. Wir können nun die URL des Repositories kopieren und in der Konsole den Befehl „git remote add origin <URL>“ ausführen. Nun können wir die Dateien in das Online-Repository hochladen. Dazu führen wir den Befehl „git push -u origin main“ aus. Nun sind alle Dateien in unserem Online-Repository gespeichert.

Zum Thema Lizensierung: Dieses Projekt wird unter der MIT-Lizenz veröffentlicht. Diese Lizenz erlaubt es, das Projekt zu verwenden, zu kopieren, zu modifizieren, zu veröffentlichen, zu verkaufen und zu vertreiben. Es ist lediglich nötig, den Urheberrechtsvermerk und die Lizenzbedingungen in Kopien der Software beizubehalten. Die Lizenzbedingungen sind in der Datei „LICENSE“ zu finden.

Die Datei .gitignore enthält alle Dateien, die nicht in das Repository hochgeladen werden sollen. Dazu gehören zum Beispiel die Dateien, die von PyCharm erstellt werden, sowie die Dateien, die von der Kompilierung des Projekts erstellt werden. Diese Dateien sind nicht relevant für das Projekt und sollten daher nicht in das Repository hochgeladen werden. Da wir eine JetBrains IDE verwenden, entscheiden wir uns dazu den gesamten .idea Ordner zu ignorieren. Auch .code-workspace-Dateien sehen wir zum Ignorieren vor, welche von Visual Studio Code für Projekte erstellt werden, falls wir uns während des Projekts dazu entscheiden, die IDE zu wechseln.
Die Datei README.md enthält eine kurze Beschreibung des Projekts und eine Anleitung zur Installation und Verwendung des Projekts. Diese Datei wird in der Regel im Hauptverzeichnis des Projekts gespeichert. Auch Angaben zur Lizensierung des Projekts sollten in dieser Datei enthalten sein.
An diesem Punkt erstellen wir auch die ersten Branches für das Projekt. Der Branch „main“ wurde bereits erstellt und enthält bereits die ersten zwei commits. Dazu erstellen wir den Branch „develop“. Der Branch „main“ enthält den aktuellen Stand des Projekts. Der Branch „develop“ enthält die aktuelle Entwicklungsversion des Projekts. Wenn wir neue Features entwickeln, werden wir diese in einem eigenen Branch entwickeln und dann in den Branch „develop“ mergen. Wenn wir eine neue Version des Projekts veröffentlichen wollen, mergen wir den Branch „dev“ in den Branch „main“.
Branches werden erstellt mit dem Befehl „git branch <branchname>“ und gewechselt mit dem Befehl „git checkout <branchname>“.

Datensicherheit

Die Versionskontrolle mit git liefert und schon eine sehr gute Sicherheit unserer Projektdaten, zusätzlich empfiehlt es sich aber natürlich noch eine weitere Datensicherungsmethode anzuwenden, beispielsweise ein Backup-Skript. Ein Beispiel wie ein solches Skript aussehen kann findet sich im Dokumentationsverzeichniss unter misc/backup_project.bat bzw. im Anhang dieser Dokumentation. Außerdem bietet PyCharm mit der Funktion „Lokal History“ auch noch ein eigenes Versionskontrollsystem. Grundsätzlich empfiehlt es sich in der Softwareentwicklung, eigentlich allgemein in der Datenverarbeitung, immer mehrere Backuplösungen parallel zu nutzen. Besser ein Backup haben und es nicht brauchen, als es zu brauchen und nicht zu haben. Natürlich muss man auch hier abwägen zwischen Benutzbarkeit

und Sicherheit, aber solange entsprechende Maßnahmen das Projekt nicht über Gebühr einschränken, spricht nichts dagegen.

MVC-Modell

Wir haben uns entschieden, das MVC-Modell für die Architektur unseres Projekts zu verwenden. MVC steht für Modell-View-Controller. Das Modell ist für die Datenverwaltung verantwortlich. Es enthält die Daten, die von der Anwendung verarbeitet werden. Die View ist für die Darstellung der Daten zuständig. Sie enthält die Benutzeroberfläche, die dem Benutzer angezeigt wird. Der Controller ist für die Datenverarbeitung verantwortlich. Er enthält die Logik, die die Daten verarbeitet.

Das MVC-Modell ist ein weit verbreitetes Modell für die Softwarearchitektur. Es eignet sich sehr gut für die Entwicklung von GUI-Anwendungen. Es ermöglicht eine klare Trennung zwischen Datenverwaltung, Datenverarbeitung und Präsentation. Dadurch verbessert sich die Wartbarkeit und Erweiterbarkeit der Software. Außerdem ermöglicht es eine parallele Entwicklung der verschiedenen Komponenten.

Insgesamt ist das MVC-Modell eine ausgezeichnete Wahl für die Entwicklung von GUI-Anwendungen, da es die genannten Vorteile bietet.

4.2 Implementierung

Fangen wir zunächst das Fenster der Anwendung zu implementieren. Wir entscheiden uns zunächst für eine initiale Größe von 600 mal 400 Pixeln und legen einen festen Punkt auf dem Bildschirm fest, wo das Fenster der Anwendung bei Programmstart erscheint, das sind die x und y Koordinaten 100. Auch definieren wir an dieser Stelle den Titel des Fensters ("py_sysmon"). Das erste Label wird auch hier festgelegt ("CPU Usage").

Davor haben wir auch die ersten Module und Klassen, die wir zunächst brauchen importiert. Wir commiten die Änderungen und pushen sie in den Branch "dev".

Mithilfe der psutil-Bibliothek haben wir die Funktion get_cpu_usage programmiert. Diese Funktion holt uns die aktuelle CPU-Auslastung in Prozent.

Die MainController-Klasse, die wir neu hinzugefügt haben, kümmert sich darum, dass die CPU-Auslastung auch im Fenster angezeigt wird. Die Methode get_cpu_usage aktualisiert regelmäßig das Label mit den neuesten Daten zur CPU-Auslastung.

Starten der Anwendung

Der Abschnitt if __name__ == "__main__" ist unser Startsignal. Hier wird MainController aktiviert und die Anwendung startet mit der Darstellung der Systeminformationen.

Aktualisierung der Anwendungslogik

Nachdem wir das Grundgerüst unserer Anwendung etabliert hatten, konzentrierten wir uns darauf, die Logik zur Überwachung der Systemleistung zu verfeinern und zu erweitern.

Um eine regelmäßige Aktualisierung der Systeminformationen zu gewährleisten, haben wir die QTimer-Klasse aus PyQt5 eingeführt. Dieser Timer wird verwendet, um die CPU-Auslastung jede Sekunde zu aktualisieren, was eine kontinuierliche Überwachung ermöglicht.

In der MainController-Klasse haben wir den QTimer konfiguriert und mit unserer update_cpu_usage-Methode verbunden. Der Timer sendet alle 1000 Millisekunden (1 Sekunde) ein Signal, das diese Methode aufruft, um die CPU-Auslastung zu ermitteln und das Label in unserem Hauptfenster entsprechend zu aktualisieren.

Optimierung der Update-Methode

Die update_cpu_usage-Methode wurde dahingehend optimiert, dass sie direkt auf die psutil.cpu_percent()-Funktion zugreift, ohne ein Intervall zu benötigen. Dies verbessert die Effizienz der Methode und gewährleistet, dass die CPU-Auslastungsinformationen in Echtzeit aktualisiert werden. Die alte get_cpu_usage-Methode wurde entfernt, da sie nicht mehr benötigt wird.

Verbesserung der Anwendungsleistung

Durch die Verwendung des Timers stellen wir sicher, dass unsere Anwendung effizient und reaktionsfähig bleibt. Da die update_cpu_usage-Methode jetzt durch den Timer gesteuert wird, verhindern wir blockierende Aufrufe und ermöglichen eine flüssigere Benutzererfahrung. Diese Verbesserungen tragen zur Stabilität und Effizienz der Anwendung bei und ermöglichen eine genauere und zuverlässigere Überwachung der Systemleistung. Durch die kontinuierliche Aktualisierung der CPU-Auslastung kann der Benutzer die Leistung seines Systems in Echtzeit überwachen.

Erweiterung der Anwendungslogik

Wir haben die Anwendungslogik erweitert, um die die Speicherauslastung des Systems zu überwachen. Dazu haben wir die Funktion get_cpu_usage umbenannt in update_system_info. Die Funktion update_system_info aktualisiert nun die CPU- und Speicherauslastung des Systems. In ähnlicher Weise fügen wir nun nach und nach weitere Funktionalitäten hinzu.

Zusätzliche UI-Elemente

Wir fügen der Anwendung auch einen Button hinzu (close) zum einfachen Beenden. Wir platzieren ihn in der rechten unteren Ecke des Fensters. Wir möchten allerdings irgendwann auch noch die Dimensionen des Fensters ändern, also ändern wir die Implementierung des Buttons dahingehend, dass seine Position relativ zu den vorher bestimmten Maßen des Hauptfensters festgelegt wird. Aber leider haben wir auch diese Änderung nicht zu Ende gedacht. Wird jetzt die Anwendung aufgerufen, wird der Button zwar immer richtig platziert, unabhängig davon welche Maße wir im Programm definiert haben, ändert aber der Nutzer im laufenden Betrieb die Größe des Fensters, wird der Button nicht mit verschoben. Wir müssen also noch eine Möglichkeit finden, die Position des Buttons dynamisch zu halten.

Wir überschreiben Sie die resizeEvent-Methode in Ihrer SystemMonitorWindow-Klasse. In der resizeEvent-Methode, aktualisieren wir die Position des Buttons basierend auf der neuen Fenstergröße. Das bedeutet für uns konkret, die Methode update_button_position wird sowohl in initUI als auch in resizeEvent aufgerufen. Dadurch wird sichergestellt, dass der Button bei der Initialisierung und bei jeder Größenänderung des Fensters richtig positioniert wird. Die resizeEvent-Methode überschreibt das Standardverhalten von QMainWindow bei einer Größenänderung, indem sie die Button-Position aktualisiert und dann das ursprüngliche resizeEvent aufruft.

Speichern der Daten

Die erhobenen Daten wollen wir nun bei jeder Erhebung in einer SQLite Datenbank abspeichern. Da SQLite eine eingebettete SQL-Datenbank ist, müssen wir keinen separaten Datenbankserver installieren oder konfigurieren. Wir können direkt in Python mit SQLite arbeiten, um die Daten zu speichern. Außerdem ist SQLite in der Standardbibliothek von Python enthalten, daher müssen wir normalerweise nichts zusätzlich installieren, um es zu verwenden. Als erstes importieren wir „sqlite3“ in unserer Python-Skript, dann stellen wir eine Verbindung zu einer SQLite-Datenbankdatei her. Wenn die Datei nicht existiert, wird sie automatisch erstellt. In unserem Projekt nennen wird die Datei py_sysmon_data.db. Wir updaten auch unsere -gitignore Datei, damit die Datenbankdatei nicht in das Repository hochgeladen wird. Dann fügen wir bei jedem Programmzylkus die Daten in die Datenbank ein wobei jeder Wert seine eigene Tabelle hat, vielleicht möchten wir dem einzelnen Wert später noch weitere Informationen hinzufügen. Auch sorgen wir dafür, das jeder Spalte maximal 1000 Einträge zugeordnet werden können, danach werden die ältesten Einträge gelöscht. Das ist

nötig, da wir die Daten später in einem Diagramm darstellen wollen und die Datenmenge nicht zu groß werden soll.

Erstellen der Diagramme

Wir nutzen jetzt das Modul matplotlib, um in einem neuen Anwendungsfenster ein Diagramm darzustellen, welches die einzelnen erhobenen Werte aus der Datenbank ausließt, farbig abgegrenzt und fortlaufend darstellt. Auch einen neuen Button im Hauptfenster zum Öffnen des Diagrammfensters fügen wir hinzu. Jetzt fehlen nur noch animierte Kurven, wir implementiere Diese, in dem wir eine Funktion erstellen zum Auslesen der Daten aus der Datenbank, fetch_data. Dann erweitern wir die Visualisierungsfunktion update_plot um die Werte für cpu_usage, memory_usage und diskursive darzustellen. Somit haben wir die geplanten Funktionalitäten umgesetzt.

Erstellen der direkt ausführbaren Anwendung

Wir nutzen jetzt Pyinstaller um aus unserem Projekt eine ausführbare, von Python unabhängige Anwendung zu erstellen. Das machen wir mit dem Befehl Konsolenbefehl „pip install pyinstaller“ auf der Windows Kommandozeile, wechseln in das „src“ Verzeichnis unseres Projekts und kopieren erst die Datei main.py zu main.pyw. Wir tun das, damit wir später kein überflüssiges Konsolenfenster bei der Ausführung der Anwendung haben. Dann führen wir den Befehl „pyinstaller --onefile main.pyw“ aus. Das erstellt eine ausführbare Datei „main.exe“ im Verzeichnis „dist“ unseres Projekts. Diese Datei können wir nun auf jedem Windows-System ausführen, ohne das Python installiert sein muss, pyinstaller hat alle nötigen Abhängigkeiten in dieser einen Datei gebündelt. Es muss besonders darauf hingewiesen werden, dass hier keine klassische Kompilierung stattfindet, sondern wirklich nur ein Bündeln des Projekts. Um den Vorgang zu vereinfachen, erstellen wir eine Windows-Batchdatei „build.bat“ die den Vorgang automatisiert. Diese Datei befindet sich im root Verzeichnis unseres Projekts.

5. Fazit

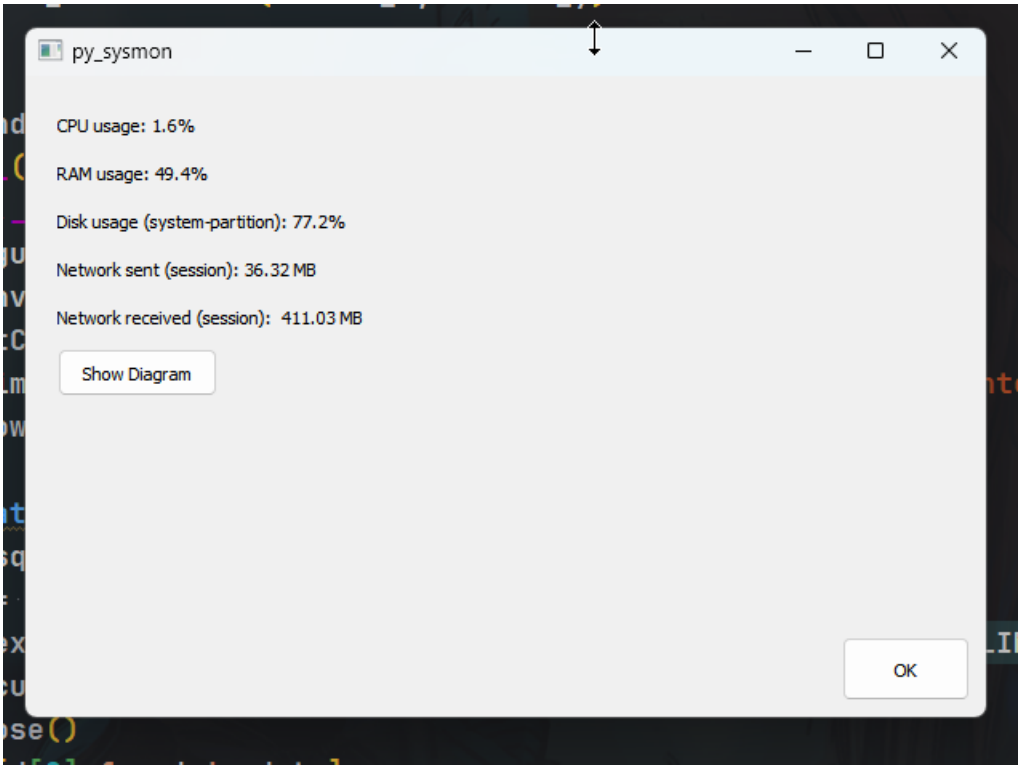
In diesem Schulprojekt wurde ein System-Monitor-Tool mit Python erstellt, das verschiedene Parameter wie CPU-Auslastung, Speicherverbrauch und Netzwerkaktivität überwacht und grafisch darstellt. Das Ziel war es, ein nützliches und benutzerfreundliches Programm zu entwickeln, das hilft, die Leistung und den Zustand eines Computers zu überprüfen.

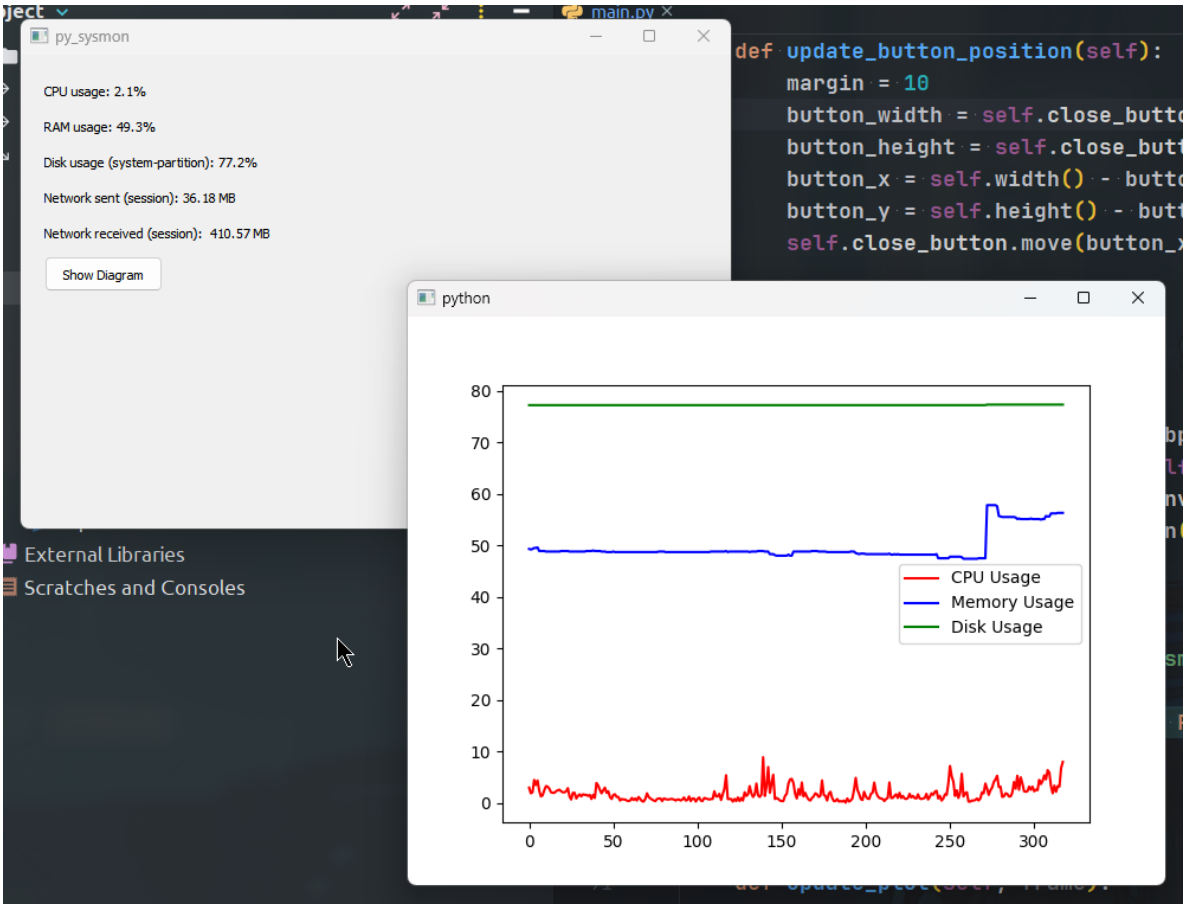
Das Projekt war sehr lehrreich, da viele neue Konzepte und Fähigkeiten im Bereich der Programmierung und der Systemverwaltung erlernt wurden. Ich habe mich mit verschiedenen Python-Bibliotheken wie psutil oder matplotlib vertraut gemacht, die mir die Arbeit erleichtert haben. Ich habe auch gelernt, wie man eine grafische Benutzeroberfläche (GUI) erstellt, die die Daten in Form von Diagrammen anzeigt. Außerdem habe ich mich mit den Grundlagen der Versionskontrolle und der Dokumentation beschäftigt.

Leider konnte ich nicht alle geplanten Features umsetzen, da ich mich noch viel in Python selbst einarbeiten musste das es (bisher) nicht zu meinem alltäglichen Tech-Stack gehört. Es war zb. dem Nutzer Mitteilungen zu senden, wenn bestimmte Systemzustände erreicht werden, wie z.B. eine bestimmte Speicher- oder CPU-Auslastung. Ich wollte auch Tests durchführen, um die Funktionalität und die Qualität meines Programms zu überprüfen. Diese Features musste ich jedoch aufgrund von Zeitmangel und technischen Schwierigkeiten auslassen.

6. Anhang

6.1 Anwendungs-Screenshots





6.2 main.py

```
import sqlite3

import matplotlib.pyplot as plt
import psutil
from PyQt5.QtCore import QTimer
from PyQt5.QtWidgets import QApplication, QMainWindow, QLabel, QPushButton
from matplotlib.animation import FuncAnimation
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas


def get_network_activity():
    net_io = psutil.net_io_counters()
    sent_bytes = net_io.bytes_sent / 1024 / 1024
    received_bytes = net_io.bytes_recv / 1024 / 1024
    return sent_bytes, received_bytes


class SystemMonitorWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.close_button = None
        self.initUI()

    def initUI(self):
        self.setWindowTitle("py_sysmon")
        self.setGeometry(100, 100, 600, 400)

        self.cpu_label = QLabel("CPU Usage: ", self)
        self.cpu_label.move(20, 20)
        self.cpu_label.resize(200, 20)

        self.memory_label = QLabel("RAM Usage: ", self)
        self.memory_label.move(20, 50)
        self.memory_label.resize(200, 20)

        self.disk_label = QLabel("Disk Usage: ", self)
        self.disk_label.move(20, 80)
        self.disk_label.resize(200, 20)

        self.network_label_sent = QLabel("Network sent: ", self)
        self.network_label_sent.move(20, 110)
        self.network_label_sent.resize(300, 20)
```

```
        self.network_label_received = QLabel("Network received: ", self)
        self.network_label_received.move(20, 140)
        self.network_label_received.resize(300, 20)

        self.close_button = QPushButton("OK", self)
        self.close_button.resize(80, 40)
        self.close_button.clicked.connect(self.close)

        self.show_diagram_button = QPushButton("Show Diagram", self)
        self.show_diagram_button.move(20, 170)
        self.show_diagram_button.clicked.connect(self.show_diagram)

        self.update_button_position()

    def show_diagram(self):
        self.diagram_window = DiagramWindow(self)

    def resizeEvent(self, event):
        self.update_button_position()
        super().resizeEvent(event)

    def update_button_position(self):
        margin = 10
        button_width = self.close_button.width()
        button_height = self.close_button.height()
        button_x = self.width() - button_width - margin
        button_y = self.height() - button_height - margin
        self.close_button.move(button_x, button_y)


class DiagramWindow(QMainWindow):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.figure, self.ax = plt.subplots()
        self.canvas = FigureCanvas(self.figure)
        self.setCentralWidget(self.canvas)
        self.animation    =    FuncAnimation(self.figure,    self.update_plot,    interval=1000,
cache_frame_data=False)
        self.show()

    def fetch_data(self, table):
        conn = sqlite3.connect('py_sysmon_data.db')
        cursor = conn.cursor()
        cursor.execute(f'SELECT usage FROM {table} ORDER BY rowid DESC LIMIT 1000')
```

```
        data = cursor.fetchall()
        conn.close()
        return [d[0] for d in data]

    def update_plot(self, frame):
        self.ax.clear()
        cpu_usage = self.fetch_data('cpu_usage')
        memory_usage = self.fetch_data('memory_usage')
        disk_usage = self.fetch_data('disk_usage')

        if cpu_usage:
            self.ax.plot(cpu_usage, label='CPU Usage', color='red')
        if memory_usage:
            self.ax.plot(memory_usage, label='Memory Usage', color='blue')
        if disk_usage:
            self.ax.plot(disk_usage, label='Disk Usage', color='green')

        self.ax.legend()
        self.canvas.draw()


class MainController:
    def __init__(self):
        self.app = QApplication([])
        self.main_window = SystemMonitorWindow()
        self.init_db()

        self.timer = QTimer()
        self.timer.timeout.connect(self.update_system_info)
        self.timer.start(1000)

    def init_db(self):
        self.conn = sqlite3.connect('py_sysmon_data.db')
        self.cursor = self.conn.cursor()
        self.cursor.execute('''CREATE TABLE IF NOT EXISTS cpu_usage (usage REAL)''')
        self.cursor.execute('''CREATE TABLE IF NOT EXISTS memory_usage (usage REAL)''')
        self.cursor.execute('''CREATE TABLE IF NOT EXISTS disk_usage (usage REAL)''')
        self.cursor.execute('''CREATE TABLE IF NOT EXISTS sent_mb (usage REAL)''')
        self.cursor.execute('''CREATE TABLE IF NOT EXISTS received_mb (usage REAL)''')

    def run(self):
        self.main_window.show()
        self.app.exec_()
```

```
    def update_system_info(self):
        cpu_usage = psutil.cpu_percent()
        memory_usage = psutil.virtual_memory()
        memory_usage = memory_usage.percent
        disk_usage = psutil.disk_usage('/')
        disk_usage = disk_usage.percent
        sent_mb, received_mb = get_network_activity()

        self.main_window.cpu_label.setText(f"CPU usage: {cpu_usage}%")
        self.main_window.memory_label.setText(f"RAM usage: {memory_usage}%")
        self.main_window.disk_label.setText(f"Disk usage (system-partition): {disk_usage}%")
        self.main_window.network_label_sent.setText(f"Network  sent  (session):  {sent_mb:.2f}
MB")
        self.main_window.network_label_received.setText(f"Network     received     (session):
{received_mb: .2f} MB")

        self.insert_into_db('cpu_usage', cpu_usage)
        self.insert_into_db('memory_usage', memory_usage)
        self.insert_into_db('disk_usage', disk_usage)
        self.insert_into_db('sent_mb', sent_mb)
        self.insert_into_db('received_mb', received_mb)

    def insert_into_db(self, table, value):
        self.cursor.execute(f'INSERT INTO {table} (usage) VALUES (?)', (value,))
        self.cursor.execute(f'DELETE FROM {table} WHERE rowid NOT IN (SELECT rowid FROM {table}
ORDER BY rowid DESC LIMIT 1000)')

        self.conn.commit()

    def closeEvent(self, event):
        self.conn.close()
        super().closeEvent(event)

    def show_diagram(self):
        self.diagram_window = DiagramWindow(self)
        self.diagram_window.show()


if __name__ == "__main__":
    controller = MainController()
    controller.run()
```

6.2 backup.bat

```
@echo off

if not DEFINED IS_MINIMIZED set IS_MINIMIZED=1 && start "" /min "%~dpnx0" %* && exit

set "project=py_sysmon"

set "projects_dir=<projects_dir>"

set "target_dir=%projects_dir%\backups"

for /f "tokens=2 delims==" %%a in ('wmic OS Get localdatetime /value') do set "dt=%%a"
set "YY=%dt:~2,2%" & set "YYYY=%dt:~0,4%" & set "MM=%dt:~4,2%" & set "DD=%dt:~6,2%"
set "HH=%dt:~8,2%" & set "Min=%dt:~10,2%" & set "Sec=%dt:~12,2%"
set "fullstamp=%YYYY%-%MM%-%DD%-%HH%%Min%%Sec%"

mkdir "%target_dir%\%project%_backup_%fullstamp%"

set "exclude_dirs="
set "exclude_files="

call :RobocopyTask "%projects_dir%\%project%" "%target_dir%\%project%_backup_%fullstamp%"

exit

:RobocopyTask

setlocal

set "source_dir=%~1"
set "dest_dir=%~2"
set   "options=/UNILOG+:%target_dir%\%project%_backup_%fullstamp%\backup_%fullstamp%.log   /MIR
/MT:16 /W:2 /R:10 /XJ /XC /ETA /TEE /XD %exclude_dirs% /XF %exclude_files%"

robocopy "%source_dir%" "%dest_dir%" %options%

endlocal

:eof
```

6.3 build.bat

```
@echo off

pip install pyinstaller
cd src
copy main.py py_symon.pyw
pyinstaller --onefile main.pyw
del main.pyw
```