Projektbeschrieb

Thursday, December 11, 2025 8:50 AM

Automatisierte System Monitoring Verison2
Beschreibung: Ein Skript was CPU, RAM, Netzwerk und Festplatten Auslastungen überwacht und bei hohen Werte automatisch eine Benachrichtigung schickt. Dazu sollte es automatisiert werden. Speziel daran ist ein Echtzeit-Dashboard, der bei der Ausführung des Skripts sichtbar ist, über ein GUI (.exe Datei). Dabei sollte man den Skript starten/beenden können, über den GUI. Das optionale dabei ist, dass man beim GUI auf die Logs zugreifen kann.

GitHub Repository Link: Sweazyyy881/M122

1. Projektstruktur

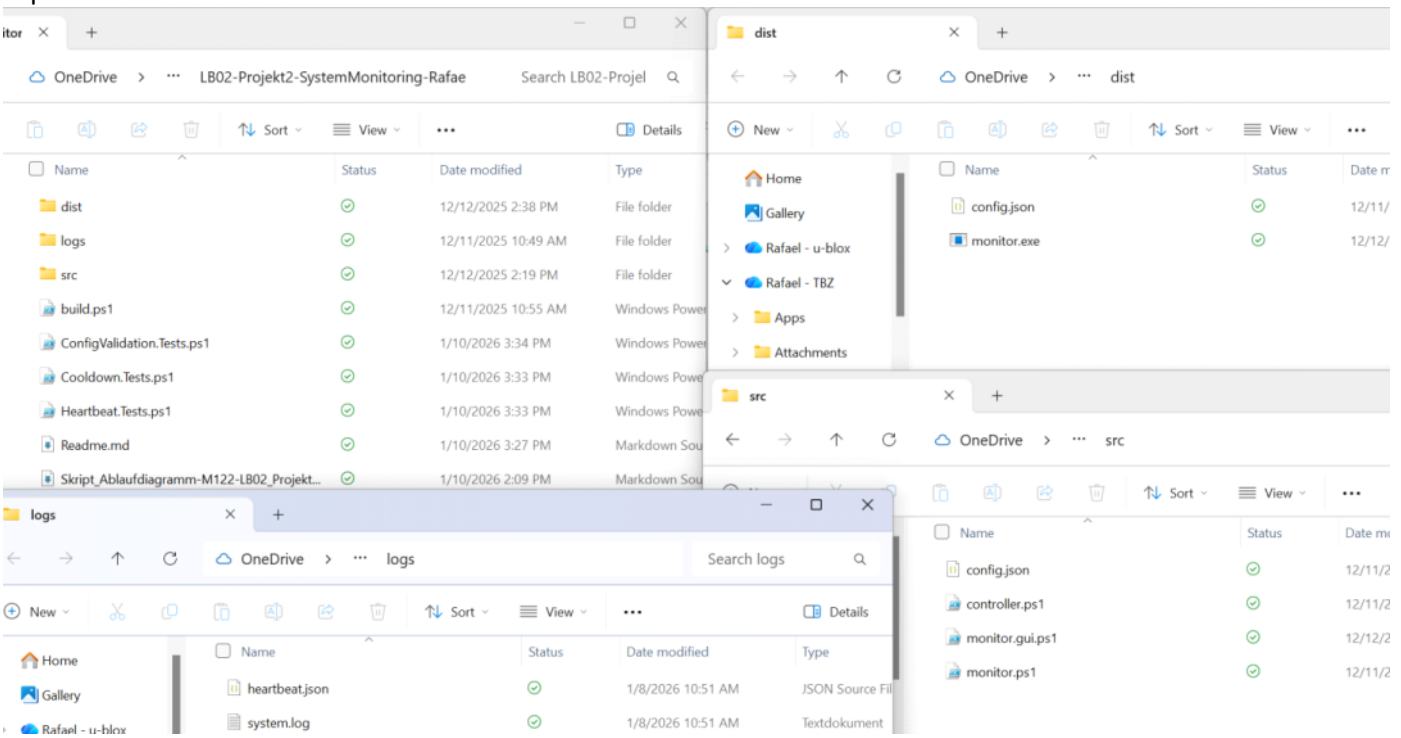
Thursday, December 11, 2025 8:51 AM

Ich musste einen Ordner erstellen für die neue Version meines Projekts. Die "monitor.exe" Datei wird während dem Prozess anhand einer anderer Datei erstellt.

Struktur:

```
LB02-Projekt2-SystemMonitoring-Rafael
├── dist
│   ├── config.json
│   └── monitor.exe
├── logs
│   ├── heartbeat.json
│   └── system.log
├── src
│   ├── config.json
│   ├── controller.ps1
│   ├── monitor.gui.ps1
│   └── monitor.ps1
├── build.ps1
├── ConfigValidation.Tests.ps1
├── Cooldown.Tests.ps1
├── Heartbeat.Tests.ps1
├── Readme.md
└── Skript_Ablaufdiagramm-M122-LB02_Projekt...
```

Explorer:



2. config.json Datei (/dist)

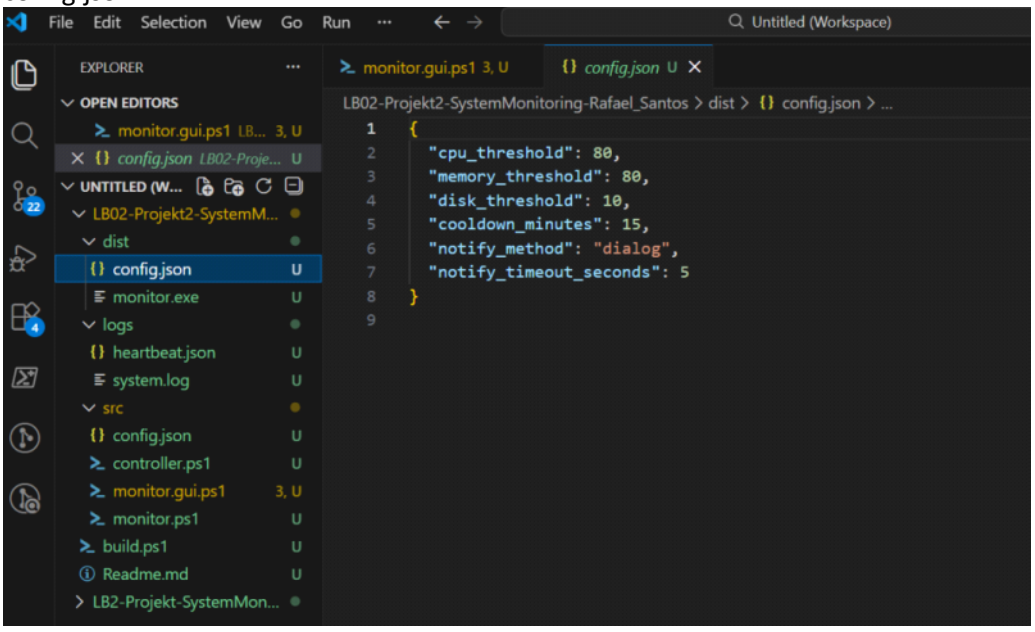
Thursday, December 11, 2025 8:51 AM

In dieser Datei kommt Werte rein die man haben möchte mit einer Angabe von dem Prozent Zahl. Der Haupt Skript liest diese Datei beim Start ab und nutzt die Werte für die Angegebenen Angaben. Dieser "config.json" Datei ist für den EndBenutzer (monitor.exe Datei).

Vorteil:

- Ich kann die Werte ändern, ohne den Code anzufassen.

config.json:



3. monitor.exe Datei (/dist)

Thursday, December 18, 2025 8:53 AM

Diese Datei ist kein Skript, sondern eine .exe Datei die ausführbar ist. Diese Datei wird während dem Prozess anhand einer anderer Datei erstellt.

Mit diesem Command, habe ich die Datei .exe Datei erstellt.

Code

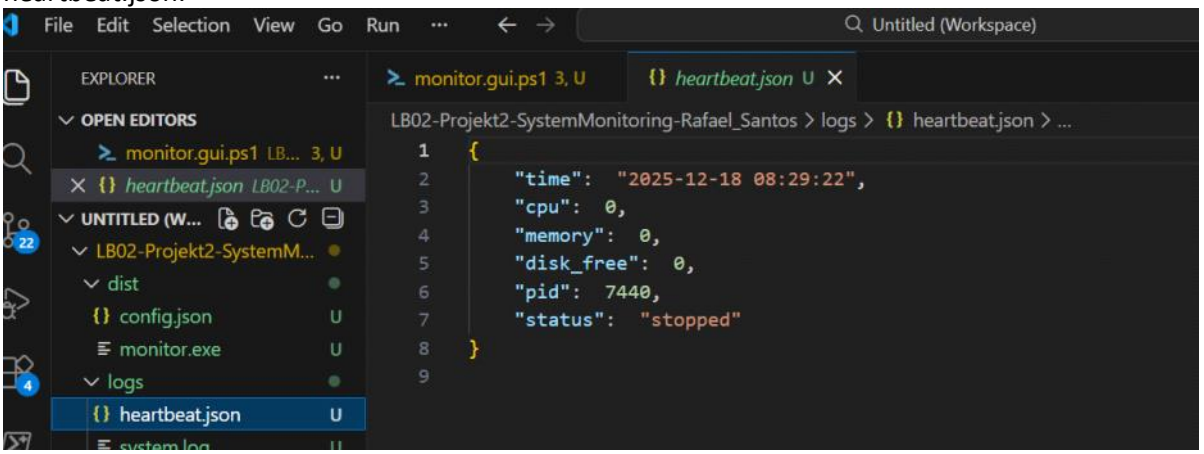
```
1. Import-Module ps2exe
2. Invoke-ps2exe -inputFile ".\src\monitor.gui.ps1" -outputFile ".\dist\monitor.exe" -noConsole -title "System Monitor" -version "1.1.0"
```

4. heartbeat.json Datei (/logs)

Thursday, December 11, 2025 8:52 AM

Diese Datei ist ein Statussignal des Monitoring Prozesses. Diese Datei wird regelmässig kontrolliert vom Skript, da die Datei bestimmte Werte enthält.

heartbeat.json:



5. system.log Datei (/logs)

Thursday, December 18, 2025 8:54 AM

In dieser Datei werden alle Ausgabe Informationen auf geschrieben und damit kann ich sehen, was auch beim GUI angezeigt wird. Ich kann daraus lesen wann es gestartet und beendet wurde. Man sieht beim Start auch den Limit den man eingestellt hat. Beim GUI hat man auch eine option den "Interval" einzustellen, den sieht man auch hier.

system.log:

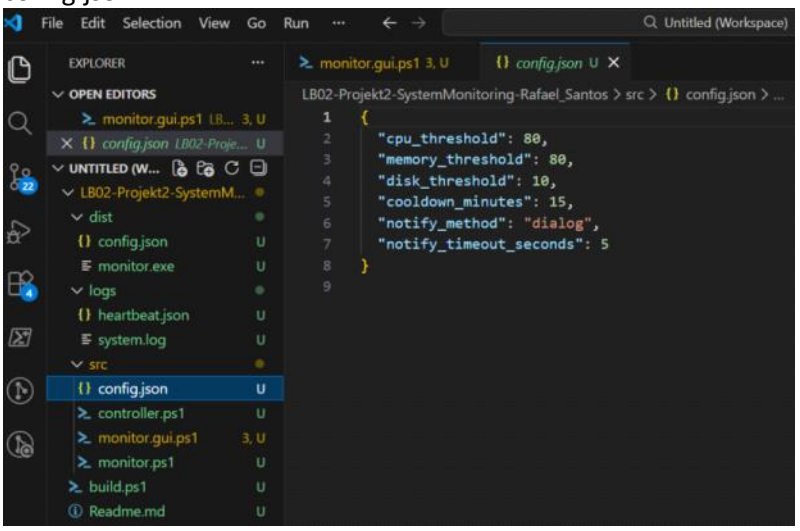
```
61  2025-12-18 08:28:11 START monitor (GUI, Interval=5 s, CPU>80, RAM>80, DiskFree<10)
62  2025-12-18 08:28:22 STATS CPU=8.48% RAM=52.87% DISKFREE=14.19%
63  2025-12-18 08:28:27 STATS CPU=1.8% RAM=52.87% DISKFREE=14.19%
64  2025-12-18 08:28:32 STATS CPU=2.66% RAM=52.77% DISKFREE=14.19%
65  2025-12-18 08:28:37 STATS CPU=3.41% RAM=52.76% DISKFREE=14.19%
66  2025-12-18 08:28:42 STATS CPU=1.85% RAM=53.34% DISKFREE=14.19%
67  2025-12-18 08:28:47 STATS CPU=4.44% RAM=53.39% DISKFREE=14.19%
68  2025-12-18 08:28:52 STATS CPU=4.19% RAM=53.39% DISKFREE=14.19%
69  2025-12-18 08:28:57 STATS CPU=3.58% RAM=53.38% DISKFREE=14.19%
70  2025-12-18 08:29:02 STATS CPU=3.41% RAM=53.32% DISKFREE=14.19%
71  2025-12-18 08:29:07 STATS CPU=4.66% RAM=53.28% DISKFREE=14.19%
72  2025-12-18 08:29:12 STATS CPU=2.51% RAM=52.87% DISKFREE=14.19%
73  2025-12-18 08:29:17 STATS CPU=1.47% RAM=52.75% DISKFREE=14.19%
74  2025-12-18 08:29:22 STATS CPU=8.72% RAM=52.93% DISKFREE=14.19%
75  2025-12-18 08:29:22 END monitor (GUI stop)
76
```

6. config.json Datei (/src)

Thursday, December 18, 2025 8:54 AM

In dieser Datei kommt Werte rein die man haben möchte mit einer Angabe von dem Prozent Zahl. Der Haupt Skript liest diese Datei beim Start ab und nutzt die Werte für die Angegebenen Angaben. Der Unterschied zwischen diesem "config.json" Datei und der andere ist, dass dieser während der Entwicklung und Tests verwendet wird. Wenn man hier etwas ändert, wirkt es auf die Entwicklungsumgebung aus.

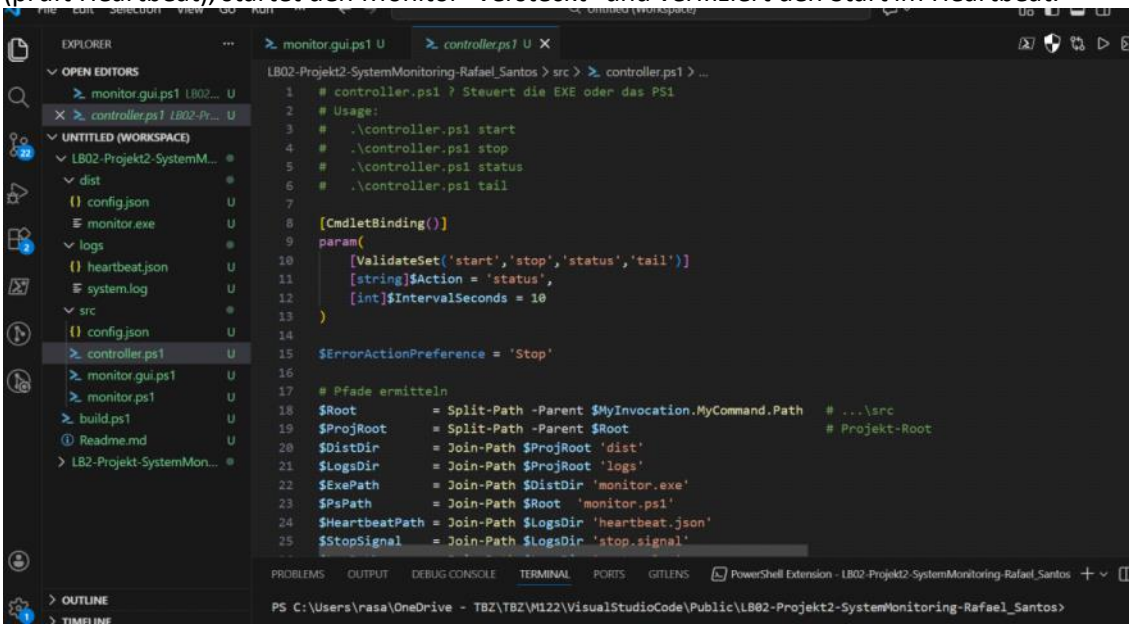
config.json:



7. controller.ps1 Datei (/src)

Thursday, December 18, 2025 8:54 AM

Der controller.ps1 ist das Stuer Skript für den Monitoring. Er startet und stoppt den Monitor prozess, liest den Heartbeat und kann die Logs live verfolgen. Der Controller vermeidet auch die Doppelstarts (prüft Heartbeat), startet den Monitor "versteckt" und verifiziert den Start im Heartbeat.



Aktionen:

- Start: Startet den Monitor
- Stop: Sendet ein Stop Signal aus und wartet auf das beenden
- Status: Liest den Heartbeat und zeigt Status/Werte
- Tail: Verfolgt live das Logfile

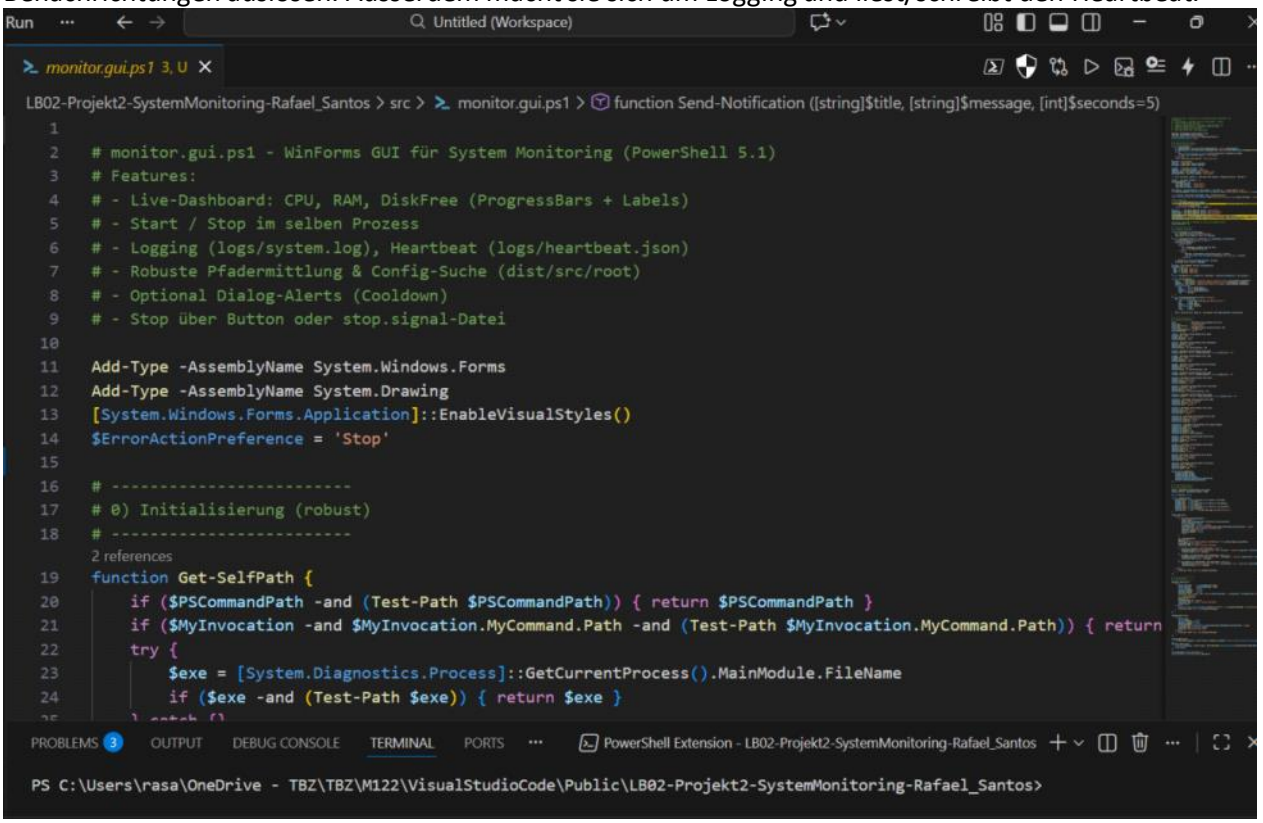
IntervalSeconds: Übergibt an den Monitor, wie oft es gemessen werden soll. Hier auf den Bild sind es 10 Sekunden. Wird auch beim Start an die EXE/PS1 weiter gegeben.

Pfade ermitteln: Dies leitet dem Controller die nötigen Pfade zum Skript ab. Er legt auch den logs Ordner automatisch an, falls der fehlen sollte.

8. monitor.gui.ps1 Datei (/src)

Thursday, December 18, 2025 8:54 AM

Diese Datei ist die Windows Forms GUI für den System-Monitoring. Sie zeigt Live Werte an, wie CPU und RAM mit dem ProgressBars + Labels, die bieten Start/Stop im selben Prozess und kann Benachrichtungen auslösen. Ausserdem macht sie sich um Logging und liest/schreibt den Heartbeat.



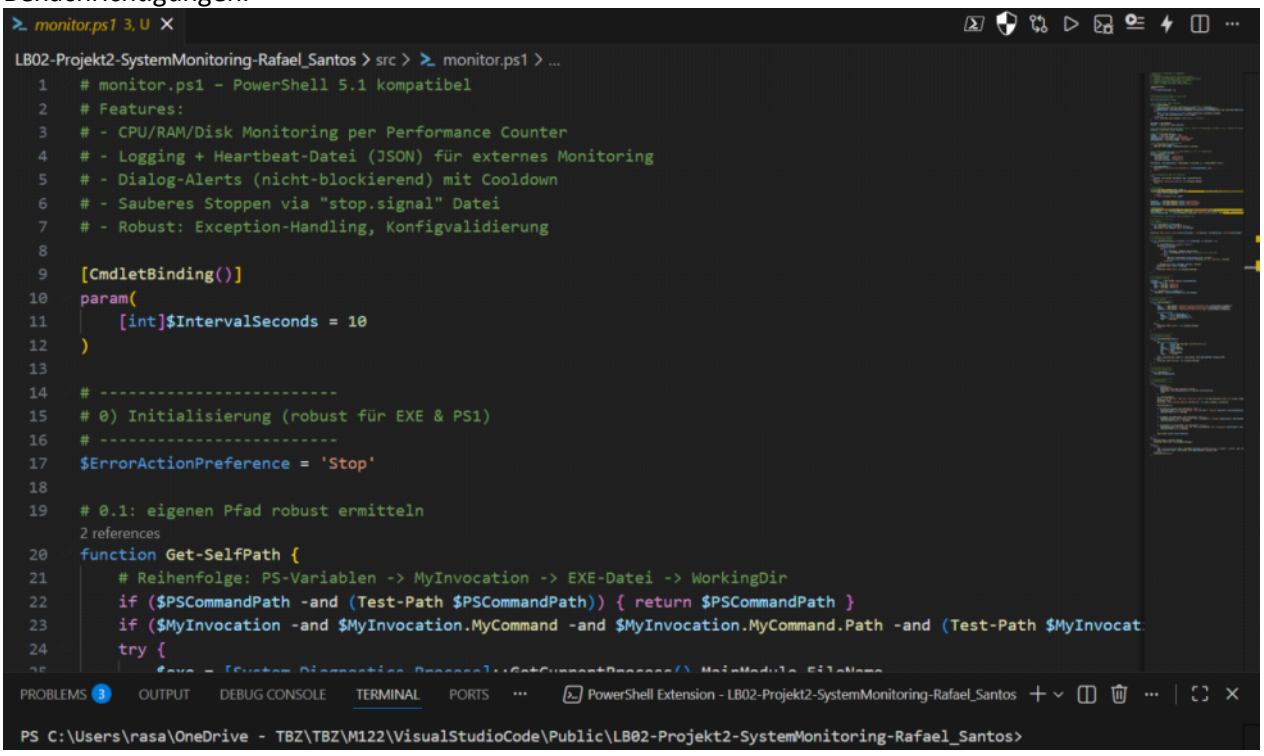
Hauptaufgaben:

- Windows Forms Oberfläche laden:
  - GUI für den System-Monitoring anzeigen. Eine Sicherheit haben, falls es einen Fehlverhalten gibt, damit es sich direkt stoppt und nicht weitere Fehler entstehen
- Live-Anzeige (CPU, RAM, Disk):
  - ProgressBars (0-100%) + Text Labels
  - Regelmässige Aktualisierung anhand des Messloops bekommen
- Start/Stop im sleben Prozess:
  - Buttons: Start und Stop
  - Beim Stop wird die Datei logs/stop.signal gesetztm damit der Monitor sauber beenden kann
- Logging & Hearbeat:
  - Schreibt Laufzeit-Infos in logs/system.log
  - Aktualisiert logs/hearbeat.json (Zeit, Status, letzte Werte)
- Konfiguration & Pfade:
  - Nutzt Hilfsfunktionen zur Pfad-Ermittlung
- Benachrichtigungen:
  - Cooldown wird eingesetzt, damit es keine Spam-Popups gibt
  - Funktion wie "Send-Notification", öffnet einen kleinen Dialog mit Nachricht, schliesst automatisch nach "notify_timeout_seconds"
- Fehlerrobustheit
  - Schutz gegen fehlende Dateien (Heartbeat/Logs/Config)

9. monitor.ps1 (/src)

Thursday, December 18, 2025 8:56 AM

Der monitor.ps1 ist der Kern Monitor. Er führt den Messloop aus (CPU, RAM, Disk), vergleicht die Werte mit der Konfiguration, schreibt Heartbeat & Logs und triggert je nach Situation die GUI Benachrichtigungen.



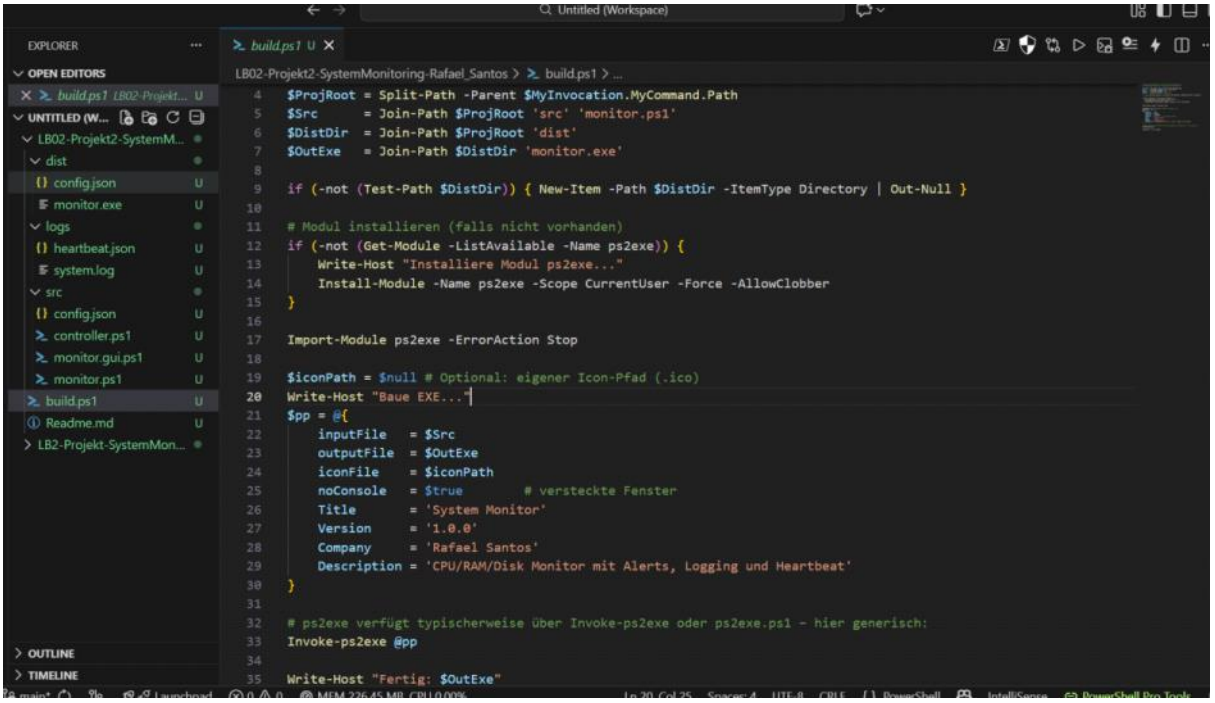
Hauptaufgaben:

- Kompatibel & robust
  - Setzt bei Fehler ein "Stop" statt Abbrüche
  - Verwendet eine Hilfsfunktion wie "Get-SelfPath", um den richtigen Arbeitsordner zu bestimmen, damit ist es sehr zuverlässig für beide Fälle wie PS1 und EXE. -> Dadurch funktionieren die Pfade zu dist/, logs/, src/ sauber ohne Fehler.
- Konfiguration laden:
  - Cooldowns setzen für die Anzeige
- Messungen erfassen:
  - CPU: Durchschnitt über mehrere Samples
  - RAM: Prozent genutzt
  - Disk: Prozent frei je Laufwerk
- Entscheiden & Benachrichtigen:
  - Vergleicht Messwerte mit den Schwellen aus der Config.
  - Verhindert Spam über einen Cooldown pro Kategorie (CPU/RAM/Disk)
  - Löst bei Überschreitung eine Benachrichtigung aus.
- Heartbeat & Logs schreiben
  - Schreibt regelmässig den Status der "CPU/RAM/Disk" ind die logs rein.
- Stop Signal überwachen:
  - Beobachtet die Datei logs/stop.signal
- Intervallgesteuerter Loop:
  - Nimmt "-IntervalSeconds" in meinem Fall 10 Sekunden per Parameter entgegen.
  - Führt in jeder Aktion: messen -> prüfen -> Benachrichtigen -> heartbeat/log -> sleep
- Fehlerbehandlung
  - Try/Catch um den Loop -> schreibt ERROR ins Log, setzt status als "error" im Hearbeat, verhindert somit eine endlose Crash Schleife.

10. build.ps1 (/LB02-Projekt2-SystemMonitoring-Rafael_Santos)

Thursday, December 18, 2025 8:56 AM

Der Sinn von build.ps1 ist, dass der Build Skript kompiliert den Quellskript src/monitor.ps1 zu einer ausführbaren Datei dist/monitor.exe mit Hilfe des PowerShell Moduls PS2EXE. Damit hat man eine portable, doppelklickbare EXE Datei für den Produktivbetrieb, während der Quellcode im src/ - Ordner bleibt. Die Voraussetzungen für dies sind PowerShell 5.1 (Windows) und PS2EXE modul installiert zu haben.



Ablauf im Skript:

1. Projektpfade ermitteln (Zeile 4 bis 7)
    a. Errechnet relative Pfade, wie Quellskript, Zielordner und Ausgabedatei.
2. Zielordner anlegen ((falls es nötig ist) Zeile 9)
3. PS2EXE installieren ((falls nicht vorhanden) Zeile 11 bis 17)
4. Build-Parameter definieren (Zeile 19 bis 30)
5. Kompilieren PS2EXE (Zeile 32 bis 35)
    a. Erstellt die monitor.exe im dist/.

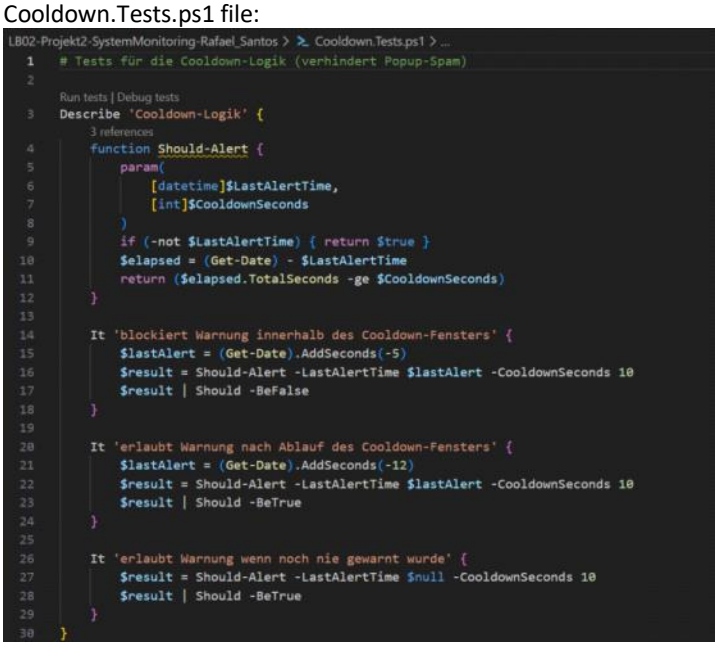
11. Tests Implementiert

Saturday, January 10, 2026 3:31 PM

Tests ausführen:

Code
Invoke-Pester -Path .\tests -Output Detailed

Cooldown.Tests.ps1 file:



```
LB02-Projekt2-SystemMonitoring-Rafael_Santos > Cooldown.Tests.ps1 > ...
# Tests für die Cooldown-Logik (verhindert Popup-Spam)

Run tests | Debug tests
Describe 'Cooldown-Logik' {
    3 references
    function Should-Alert {
        param(
            [datetime]$LastAlertTime,
            [int]$CooldownSeconds
        )
        if (-not $LastAlertTime) { return $true }
        $elapsed = (Get-Date) - $LastAlertTime
        return ($elapsed.TotalSeconds -ge $CooldownSeconds)
    }

    It 'blockiert Warnung innerhalb des Cooldown-Fensters' {
        $lastAlert = (Get-Date).AddSeconds(-5)
        $result = Should-Alert -LastAlertTime $lastAlert -CooldownSeconds 10
        $result | Should -BeFalse
    }

    It 'erlaubt Warnung nach Ablauf des Cooldown-Fensters' {
        $lastAlert = (Get-Date).AddSeconds(-12)
        $result = Should-Alert -LastAlertTime $lastAlert -CooldownSeconds 10
        $result | Should -BeTrue
    }

    It 'erlaubt Warnung wenn noch nie gewarnt wurde' {
        $result = Should-Alert -LastAlertTime $null -CooldownSeconds 10
        $result | Should -BeTrue
    }
}
```

Heartbeat.Tests.ps1 file:

LB02-Projekt2-SystemMonitoring-Rafael_Santos > Heartbeat.Tests.ps1 > Describe 'Heartbeat-Datei' {}

```powershell
# Tests für Heartbeat-Datei (Status & Werte)

Run tests | Debug tests
Describe 'Heartbeat-Datei' {
    $logsDir = Join-Path $PSScriptRoot '..\logs'
    $hbPath  = Join-Path $logsDir 'heartbeat.json'

    BeforeAll {
        if (-not (Test-Path $logsDir)) { New-Item -ItemType Directory
    }
    AfterAll {
        if (Test-Path $hbPath) { Remove-Item $hbPath -Force }
    }

    7 references
    function Write-Heartbeat {
        param(
            [string]$Status,
            [double]$CPU,
            [double]$RAM,
            [double]$Disk,
            [string]$Path
        )
        $obj = [pscustomobject]@{
            timestamp = (Get-Date).ToString('o')
            status    = $Status
            cpu       = [math]::Round($CPU, 2)
            ram       = [math]::Round($RAM, 2)
            disk      = [math]::Round($Disk, 2)
            pid       = $PID
        }
        $json = $obj | ConvertTo-Json -Depth 3
        $json | Set-Content -Path $Path -Encoding UTF8
    }

    It 'legt heartbeat.json an und enthält gültige Felder' {
        Write-Heartbeat -Status 'running' -CPU 23.1 -RAM 41.4 -Disk 7
        (Test-Path $hbPath) | Should -BeTrue

        $hb = Get-Content $hbPath -Raw | ConvertFrom-Json
        $hb.status | Should -Be 'running'
        $hb.pid    | Should -Be $PID
        [double]$hb.cpu  | Should -BeGreaterThanOrEqual 0
        [double]$hb.ram  | Should -BeGreaterThanOrEqual 0
        [double]$hb.disk | Should -BeGreaterThanOrEqual 0
        $hb.timestamp | Should -Not -BeNullOrEmpty
    }
}
```

ConfigValidation.Tests.ps1 file:


mMonitoring-Rafael_Santos > ConfigValidation.Tests.ps1 > Describe 'Config-Validierung' {} > It 'lädt eine g
Tests für das Laden und Validieren der config.json
Run tests | Debug tests
Describe 'Config-Validierung' {
$distDir = Join-Path $PSScriptRoot '..\dist'
$configPath = Join-Path $distDir 'config.json'
BeforeAll {
if (-not (Test-Path $distDir)) { New-Item -ItemType Directory
@"
{
"cpu_threshold": 85,
"memory_threshold": 80,
"disk_threshold": 15,
"notify_timeout_seconds": 5
}
"@ | Set-Content -Path $configPath -Encoding UTF8
}
AfterAll {
if (Test-Path $configPath) { Remove-Item $configPath -Force }
}
2 references
function Load-Config {
param([string]$Path)
if (-not (Test-Path $Path)) {
throw "Konfiguration nicht gefunden: $Path"
}
$cfg = Get-Content $Path -Raw | ConvertFrom-Json
foreach ($k in 'cpu_threshold','memory_threshold','disk_thresh
if (-not ($cfg.PSObject.Properties.Name -contains $k)) {
throw "Fehlender Schlüssel: $k"
}
$v = $cfg.$k
if (-not ($v -is [int])) { throw "Ungültiger Typ für $k: $
if ($v -lt 1 -or $v -gt 99) { throw "$k außerhalb 1..99: $
}
if (-not ($cfg.PSObject.Properties.Name -contains 'notify_time
$cfg | Add-Member -NotePropertyName 'notify_timeout_seconc
} elseif (-not ($cfg.notify_timeout_seconds -is [int])) {
$cfg.notify_timeout_seconds = 5
}
return $cfg
}
It 'lädt eine gültige Konfiguration' {
$cfg = Load-Config -Path $configPath
$cfg.cpu_threshold | Should -Be 85
$cfg.memory_threshold | Should -Be 80
$cfg.disk_threshold | Should -Be 15
$cfg.notify_timeout_seconds | Should -Be 5
}
It 'wirft Fehler bei fehlenden Keys' {
$bad = Join-Path $distDir 'bad.json'
@"
{ "cpu_threshold": 90, "memory_threshold": 80 }
"@ | Set-Content -Path $bad -Encoding UTF8

11. README.md (/LB02-Projekt2-SystemMonitoring-Rafael_Santos)

Thursday, December 18, 2025 8:57 AM

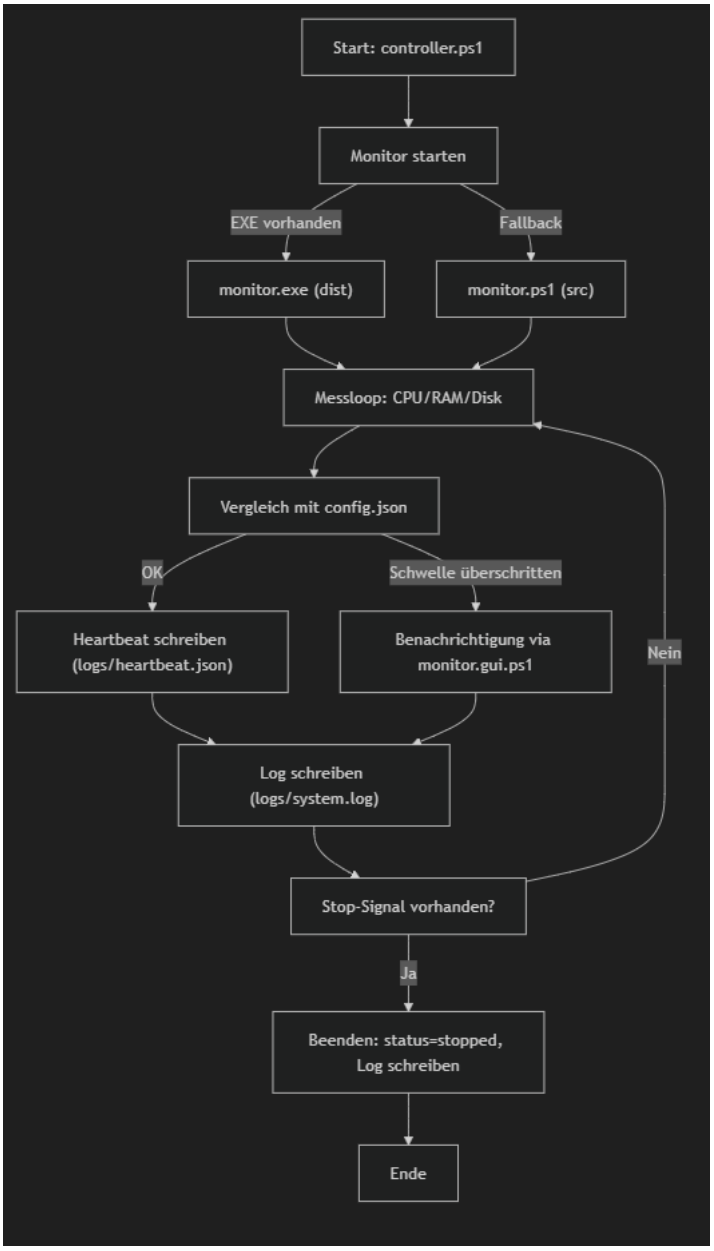
````
# System Monitor (CPU/RAM/Disk) – Rafael Santos
Ein leichtgewichtiges Monitoring für Windows: überwacht CPU, RAM und freien
Plattenplatz, schreibt Logs, erzeugt Heartbeat und zeigt Benachrichtigungen.
Start/Stop/Status via Controller-Skript, optional als EXE gepackt.
## Features
- Echtzeit-Überwachung (Intervall konfigurierbar)
- Benachrichtigungen (Dialog-Popups, Cooldown)
- Logging (`logs/system.log`)
- Heartbeat (`logs/heartbeat.json` – Status, PID, letzte Werte)
- Sauberes Stoppen via `logs/stop.signal`
- Build als EXE via `ps2exe`
- Controller: `start`, `status`, `tail`, `stop`

## Voraussetzung
- Windows PowerShell **5.1**
- Modul **ps2exe** (für EXE)
- Optional: **Pester** (für Tests)
## Installation
```powershell
Set-ExecutionPolicy RemoteSigned -Scope CurrentUser -Force
Install-Module ps2exe -Scope CurrentUser -Force -AllowClobber

## Sicherheit
Dieses Projekt führt Monitoring und Benachrichtigungen lokal auf Windows aus.
Für einen sicheren Betrieb beachte bitte:
### Ausführungsrichtlinien & Signierung
- Set-ExecutionPolicy **RemoteSigned** (Scope **CurrentUser**) für
Entwicklung/Tests:
  ```powershell
  Set-ExecutionPolicy RemoteSigned -Scope CurrentUser -Force
````

12. Ablaufdiagramm

Thursday, December 11, 2025 8:52 AM

LB02-Projekt2-SystemMonitoring-Rafael_Santos > Skript_Ablaufdiagramm-M122-LB02_Projekt2.md

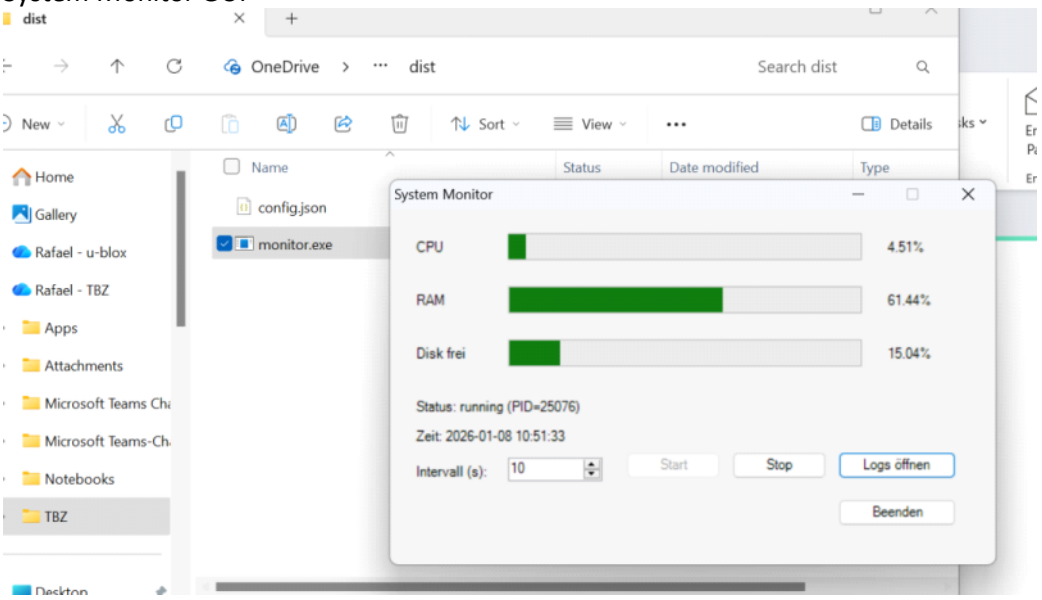
````
```mermaid
flowchart TD
    A["Start: controller.ps1"]
    B["Monitor starten"]
    C["monitor.exe (dist)"]
    D["monitor.ps1 (src)"]
    E["Messloop: CPU/RAM/Disk"]
    F["Vergleich mit config.json"]
    G["Heartbeat schreiben (logs/heartbeat.json)"]
    H["Benachrichtigung via monitor.gui.ps1"]
    I["Log schreiben (logs/system.log)"]
    J["Stop-Signal vorhanden?"]
    K["Beenden: status=stopped, Log schreiben"]
    L["Ende"]

    A --> B
    B -->|EXE vorhanden| C
    B -->|Fallback| D
    C --> E
    D --> E
    E --> F
    F -->|OK| G
    F -->|Schwelle überschritten| H
    G --> I
    H --> I
    I --> J
    J -->|Nein| E
    J -->|Ja| K
    K --> L
```
````


Start: controller.ps1
Monitor starten
EXE vorhanden
Fallback
monitor.exe (dist)
monitor.ps1 (src)
Messloop: CPU/RAM/Disk
Vergleich mit config.json
OK
Schwelle überschritten
Heartbeat schreiben
(logs/heartbeat.json)
Benachrichtigung via
monitor.gui.ps1
Nein
Log schreiben
(logs/system.log)
Stop-Signal vorhanden?
Ja
Beenden: status=stopped,
Log schreiben
Ende

13. GUI

Friday, December 12, 2025 2:17 PM

System Monitor GUI



14. Bedienungsanleitung

Thursday, December 18, 2025 8:18 AM

- Starten:
  - Öffne den Ordner dist/
  - Doppelklicke auf monitor.exe, die GUI öffnet sich und zeigt CPU-, RAM- und Festplattenstatus
- Bedienung der GUI:
  - CPU / RAM / Disk: Zeigt aktuelle Auslastung in Prozent
  - Intervall(s): Messintervall in Sekunden einstellen (Standard: 10)
  - Buttons:
    - Start: Startet den Monitor mit dem gewählten Intervall
    - Stop: Beendet den Monitor
    - Logs öffnen: Öffnet die Logdatei (logs/system.log)
    - Beenden: Schliesst die GUI
- Logs & Status:
  - Heartbeat: logs/heartbeat.json -> Status,PID, letze Werte
  - System-Log: logs/system.log -> Warnungen, Start/Stop, Fehler
- Konfiguration:
  - Datei: dist/config.json kann geändert werden von dem Benutzer.
- Stoppen:
  - Klicke auf Stop oder Beenden
  - Status wechselt auf stopped
- Fehlerbehebung:
  - Keine Anzeige? -> Prüfe logs/system.log
  - Prozess hängt? -> Manuell beenden:

Code
taskkill /F /IM monitor.exe