

Projektbeschreibung

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](https://github.com/Sweazyyy881/M122)

1. Projektstruktur

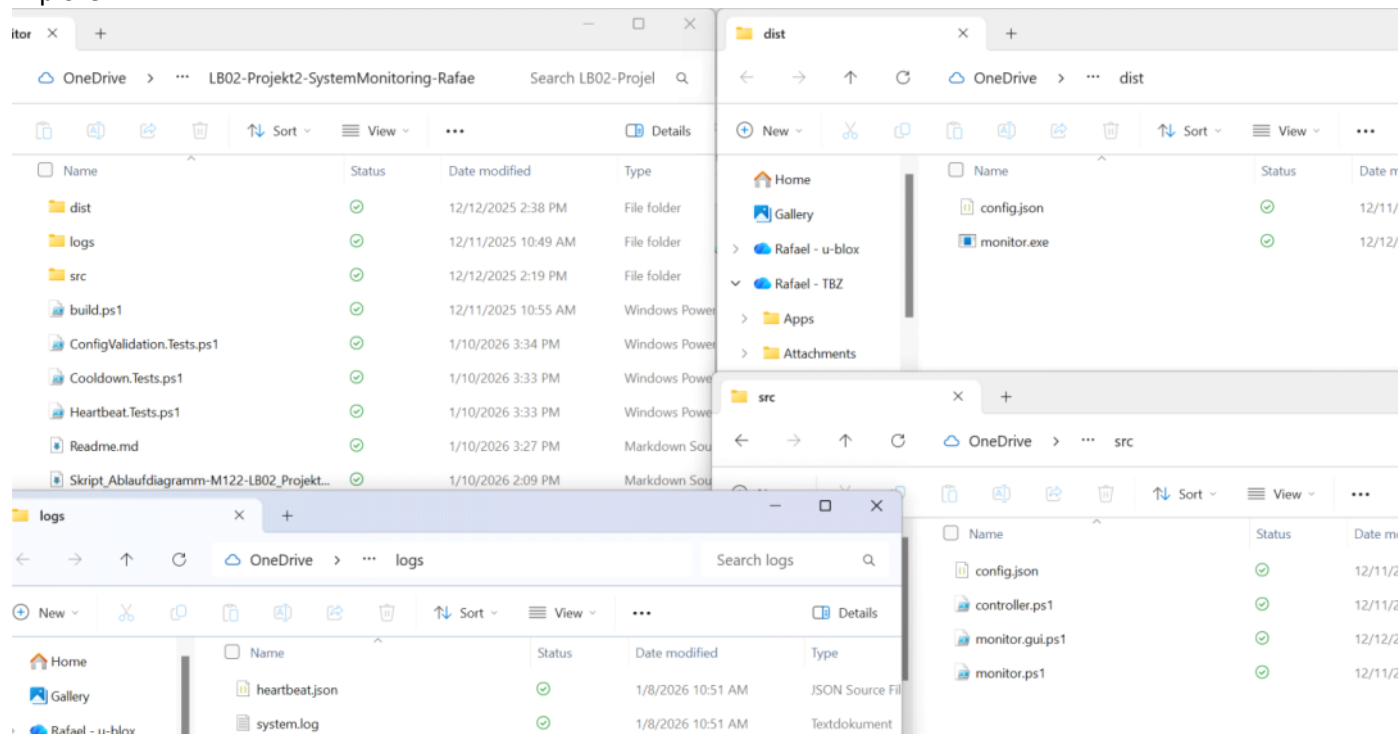
Thursday, December 11, 2025 8:51 AM

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

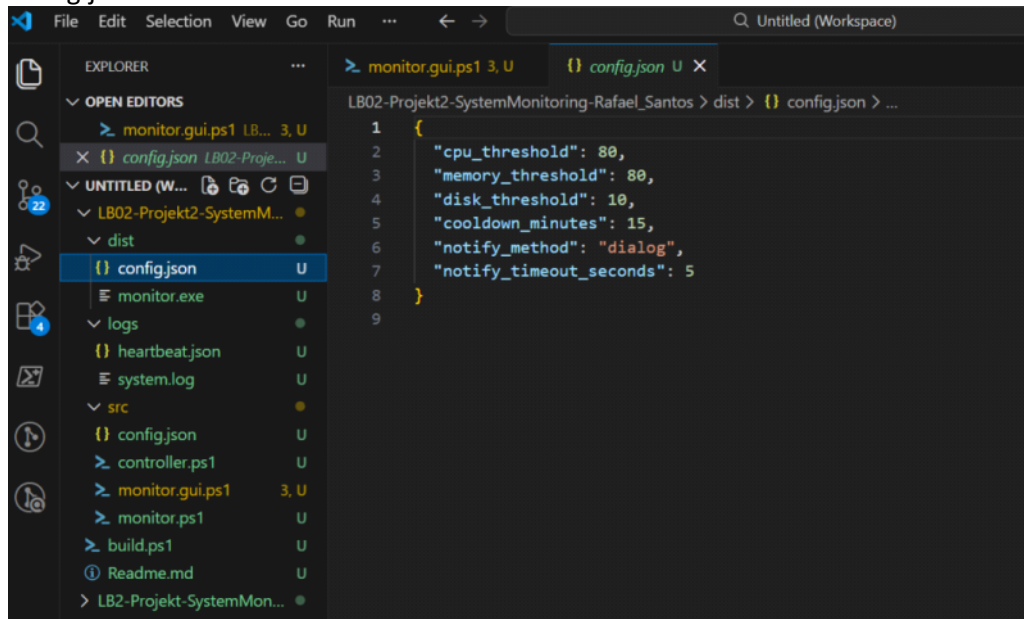
8:51 AM

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:



The screenshot shows a code editor with a dark theme. The Explorer panel on the left shows the project structure with folders like 'dist', 'logs', 'src', and 'src'. The 'dist' folder is expanded, showing 'config.json' selected. The main editor area displays the content of 'config.json' with the following JSON structure:

```
1 {  
2   "cpu_threshold": 80,  
3   "memory_threshold": 80,  
4   "disk_threshold": 10,  
5   "cooldown_minutes": 15,  
6   "notify_method": "dialog",  
7   "notify_timeout_seconds": 5  
8 }  
9
```

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"
```

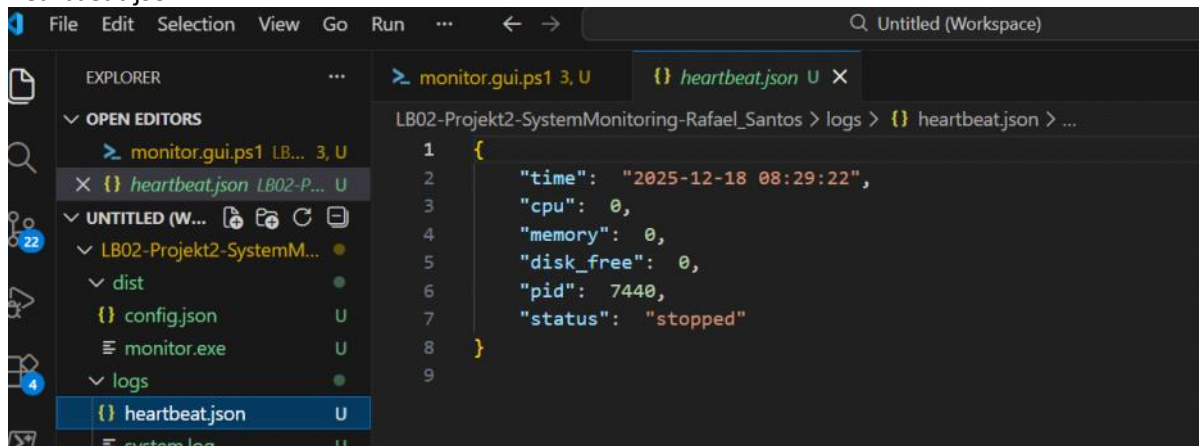
4. heartbeat.json Datei (/logs)

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:



The screenshot shows a code editor with a dark theme. The Explorer panel on the left shows a project structure with folders 'dist' and 'logs'. The 'logs' folder is expanded, showing the 'heartbeat.json' file. The main editor area displays the content of 'heartbeat.json' as a JSON object. The breadcrumb at the top of the editor indicates the path: 'LB02-Projekt2-SystemMonitoring-Rafael_Santos > logs > {} heartbeat.json > ...'.

```
1  {
2    "time": "2025-12-18 08:29:22",
3    "cpu": 0,
4    "memory": 0,
5    "disk_free": 0,
6    "pid": 7440,
7    "status": "stopped"
8  }
9
```

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
```

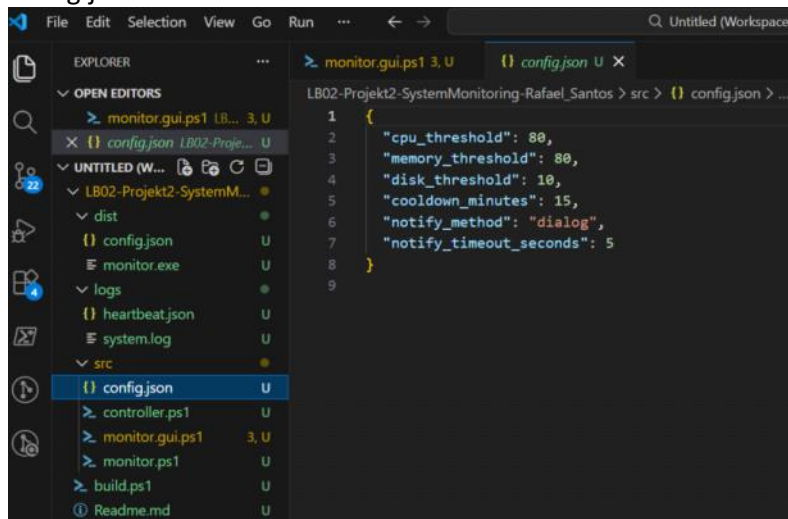
6. config.json Datei (/src)

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:



The screenshot shows a code editor with a dark theme. The Explorer panel on the left shows the project structure. The 'src' folder is expanded, and 'config.json' is selected. The main editor area displays the content of 'config.json'.

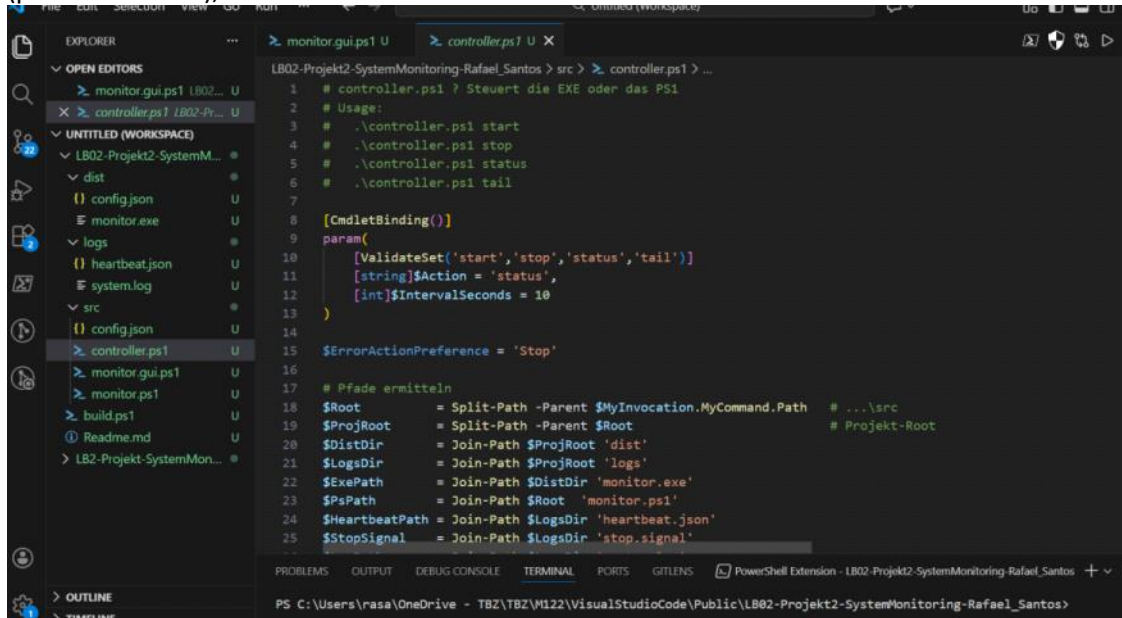
```
1 {
2   "cpu_threshold": 80,
3   "memory_threshold": 80,
4   "disk_threshold": 10,
5   "cooldown_minutes": 15,
6   "notify_method": "dialog",
7   "notify_timeout_seconds": 5
8 }
9
```

7. controller.ps1 Datei (/src)

Thursday, December 18, 2025

8:54 AM

Der controller.ps1 ist das Steuer 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.



```
1 # controller.ps1 ? Steuert die EXE oder das PS1
2 # Usage:
3 # .\controller.ps1 start
4 # .\controller.ps1 stop
5 # .\controller.ps1 status
6 # .\controller.ps1 tail
7
8 [CmdletBinding()]
9 param(
10     [ValidateSet('start','stop','status','tail')]
11     [string]$Action = 'status',
12     [int]$IntervalSeconds = 10
13 )
14
15 $ErrorActionPreference = 'Stop'
16
17 # Pfade ermitteln
18 $Root = Split-Path -Parent $MyInvocation.MyCommand.Path # ... \src
19 $ProjRoot = Split-Path -Parent $Root # Projekt-Root
20 $DistDir = Join-Path $ProjRoot 'dist'
21 $LogsDir = Join-Path $ProjRoot 'logs'
22 $ExePath = Join-Path $DistDir 'monitor.exe'
23 $PsPath = Join-Path $Root 'monitor.ps1'
24 $HeartbeatPath = Join-Path $LogsDir 'heartbeat.json'
25 $StopSignal = Join-Path $LogsDir 'stop.signal'
```

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.

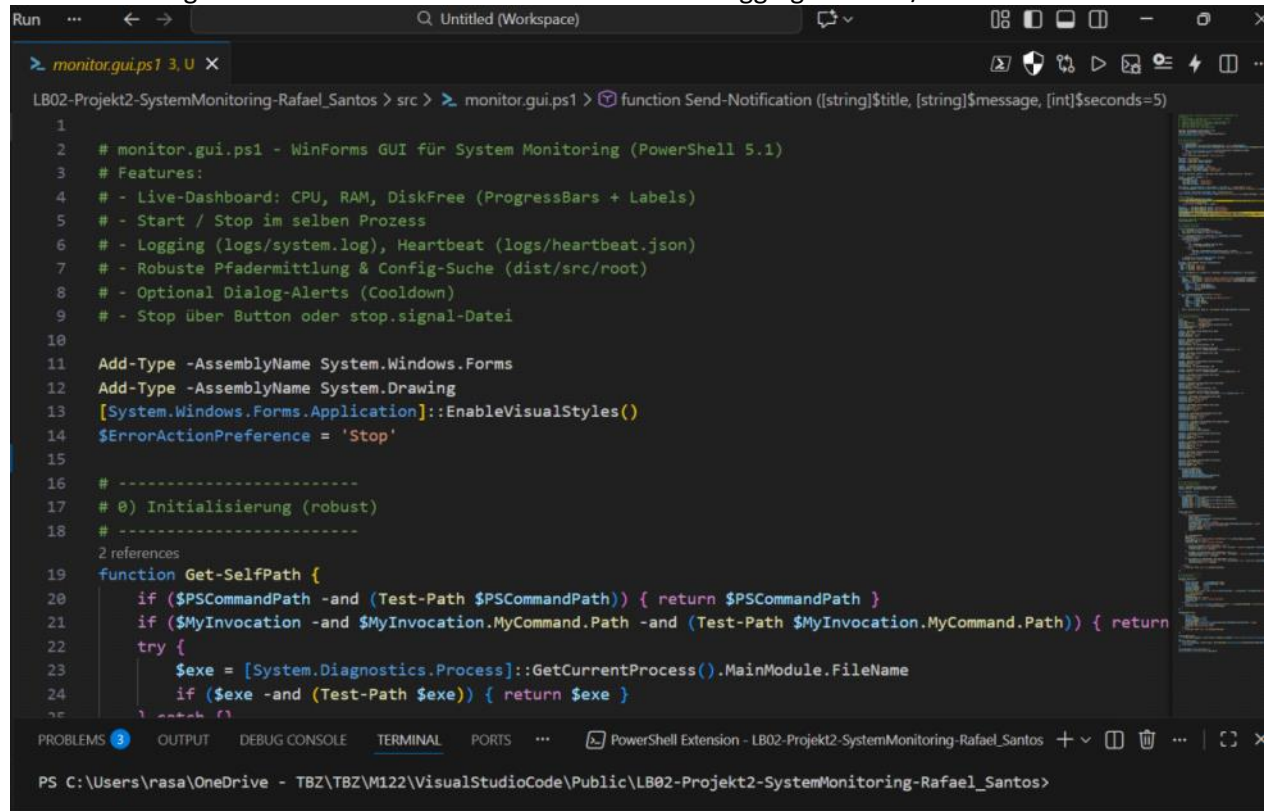
Pfade ermitteln: Dies leitet dem Controller die nötigen Pfade zum Skript ab. Er legt auch den logs Ordner automatisch an, falls der fehlen sollte.

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 Benachrichtigungen auslösen. Ausserdem macht sie sich um Logging und liest/schreibt den Heartbeat.



```
Run ... ← → Q Untitled (Workspace)
> monitor.gui.ps1 3, U X
LB02-Projekt2-SystemMonitoring-Rafael_Santos > src > > monitor.gui.ps1 > function Send-Notification ([string]$title, [string]$message, [int]$seconds=5)
1
2 # monitor.gui.ps1 - WinForms GUI für System Monitoring (PowerShell 5.1)
3 # Features:
4 # - Live-Dashboard: CPU, RAM, DiskFree (ProgressBars + Labels)
5 # - Start / Stop im selben Prozess
6 # - Logging (logs/system.log), Heartbeat (logs/heartbeat.json)
7 # - Robuste Pfadermittlung & Config-Suche (dist/src/root)
8 # - Optional Dialog-Alerts (Cooldown)
9 # - Stop über Button oder stop.signal-Datei
10
11 Add-Type -AssemblyName System.Windows.Forms
12 Add-Type -AssemblyName System.Drawing
13 [System.Windows.Forms.Application]::EnableVisualStyles()
14 $ErrorActionPreference = 'Stop'
15
16 # -----
17 # 0) Initialisierung (robust)
18 # -----
19 2 references
20 function Get-SelfPath {
21     if ($PSCommandPath -and (Test-Path $PSCommandPath)) { return $PSCommandPath }
22     if ($MyInvocation -and $MyInvocation.MyCommand.Path -and (Test-Path $MyInvocation.MyCommand.Path)) { return
23     try {
24         $exe = [System.Diagnostics.Process]::GetCurrentProcess().MainModule.FileName
25         if ($exe -and (Test-Path $exe)) { return $exe }
26     } catch {}
27 }
```

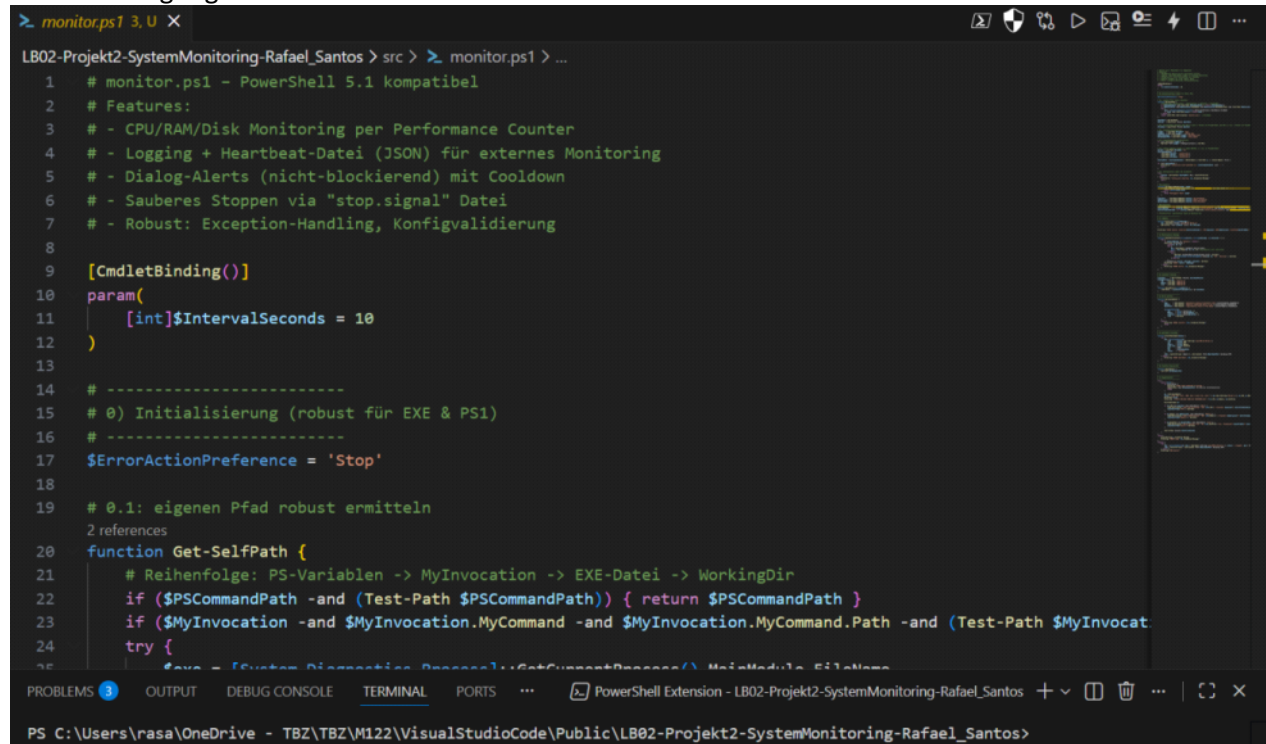
Hauptaufgaben:

- 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 selben Prozess:
 - Buttons: Start und Stop
 - Beim Stop wird die Datei logs/stop.signal gesetzt damit der Monitor sauber beenden kann
- Logging & Heartbeat:
 - Schreibt Laufzeit-Infos in logs/system.log
 - Aktualisiert logs/heartbeat.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.



```
1 # monitor.ps1 - PowerShell 5.1 kompatibel
2 # Features:
3 # - CPU/RAM/Disk Monitoring per Performance Counter
4 # - Logging + Heartbeat-Datei (JSON) für externes Monitoring
5 # - Dialog-Alerts (nicht-blockierend) mit Cooldown
6 # - Sauberes Stoppen via "stop.signal" Datei
7 # - Robust: Exception-Handling, Konfigvalidierung
8
9 [CmdletBinding()]
10 param(
11     [int]$IntervalSeconds = 10
12 )
13
14 # -----
15 # 0) Initialisierung (robust für EXE & PS1)
16 # -----
17 $ErrorActionPreference = 'Stop'
18
19 # 0.1: eigenen Pfad robust ermitteln
20 function Get-SelfPath {
21     # Reihenfolge: PS-Variablen -> MyInvocation -> EXE-Datei -> WorkingDir
22     if ($PSCommandPath -and (Test-Path $PSCommandPath)) { return $PSCommandPath }
23     if ($MyInvocation -and $MyInvocation.MyCommand -and $MyInvocation.MyCommand.Path -and (Test-Path $MyInvocation.MyCommand.Path)) { return $MyInvocation.MyCommand.Path }
24     try {
25         $Path = [System.Diagnostics.Process]::GetCurrentProcess().MainModule.FileName
26     } catch {
27         return $null
28     }
29 }
```

Hauptaufgaben:

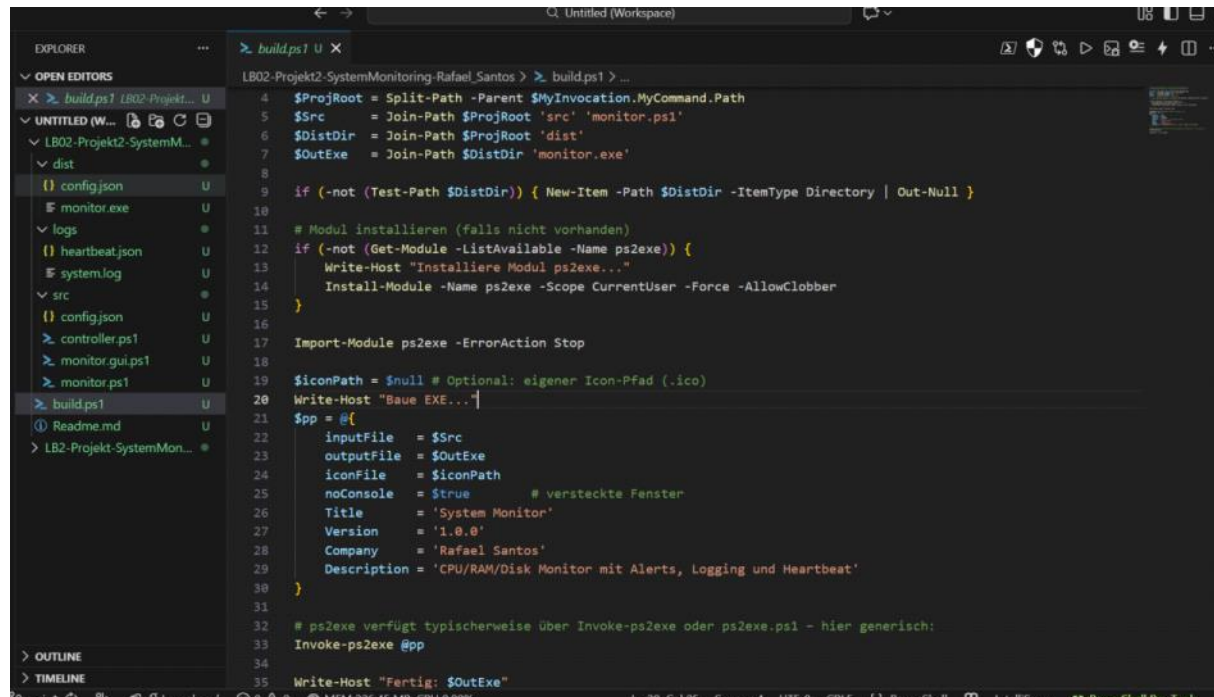
- Kompatibel & robust
 - Setzt bei Fehler ein "Stop" statt Abbrüche
 - Verwendet eine Hilfsfunktion wie "Get-SelfPath", um den richtigen Arbeitsordner zu bestimmen, damit ist es sehr zuverlässig für beide Fälle wie PS1 und EXE. -> Dadurch funktionieren die Pfade zu dist/, logs/, src/ sauber ohne Fehler.
- 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.



```
4 $ProjRoot = Split-Path -Parent $MyInvocation.MyCommand.Path
5 $Src = Join-Path $ProjRoot 'src' 'monitor.ps1'
6 $DistDir = Join-Path $ProjRoot 'dist'
7 $OutExe = Join-Path $DistDir 'monitor.exe'
8
9 if (-not (Test-Path $DistDir)) { New-Item -Path $DistDir -ItemType Directory | Out-Null }
10
11 # Modul installieren (falls nicht vorhanden)
12 if (-not (Get-Module -ListAvailable -Name ps2exe)) {
13     Write-Host "Installiere Modul ps2exe..."
14     Install-Module -Name ps2exe -Scope CurrentUser -Force -AllowClobber
15 }
16
17 Import-Module ps2exe -ErrorAction Stop
18
19 $IconPath = $null # Optional: eigener Icon-Pfad (.ico)
20 Write-Host "Baue EXE..."
21 $pp = @{
22     inputFile = $Src
23     outputFile = $OutExe
24     iconFile = $IconPath
25     noConsole = $true # versteckte Fenster
26     Title = 'System Monitor'
27     Version = '1.0.0'
28     Company = 'Rafael Santos'
29     Description = 'CPU/RAM/Disk Monitor mit Alerts, Logging und Heartbeat'
30 }
31
32 # ps2exe verfügt typischerweise über Invoke-ps2exe oder ps2exe.ps1 - hier generisch:
33 Invoke-ps2exe @pp
34
35 Write-Host "Fertig: $OutExe"
```

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 > ...
1 # Tests für die Cooldown-Logik (verhindert Popup-Spam)
2
3 Run tests | Debug tests
4 Describe 'Cooldown-Logik' {
5     3 references
6     function Should-Alert {
7         param(
8             [datetime]$LastAlertTime,
9             [int]$CooldownSeconds
10        )
11        if (-not $LastAlertTime) { return $true }
12        $elapsed = (Get-Date) - $LastAlertTime
13        return ($elapsed.TotalSeconds -ge $CooldownSeconds)
14    }
15
16    It 'blockiert Warnung innerhalb des Cooldown-Fensters' {
17        $lastAlert = (Get-Date).AddSeconds(-5)
18        $result = Should-Alert -LastAlertTime $lastAlert -CooldownSeconds 10
19        $result | Should -BeFalse
20    }
21
22    It 'erlaubt Warnung nach Ablauf des Cooldown-Fensters' {
23        $lastAlert = (Get-Date).AddSeconds(-12)
24        $result = Should-Alert -LastAlertTime $lastAlert -CooldownSeconds 10
25        $result | Should -BeTrue
26    }
27
28    It 'erlaubt Warnung wenn noch nie gewarnt wurde' {
29        $result = Should-Alert -LastAlertTime $null -CooldownSeconds 10
30        $result | Should -BeTrue
31    }
32 }
```

Heartbeat.Tests.ps1 file:

```
LB02-Projekt2-SystemMonitoring-Rafael_Santos > > Heartbeat.Tests.ps1 > Describe 'Heartbeat-Datei' {}
1 # Tests für Heartbeat-Datei (Status & Werte)
2
3 Run tests | Debug tests
4 Describe 'Heartbeat-Datei' {
5     $logsDir = Join-Path $PSScriptRoot '..\logs'
6     $hbPath = Join-Path $logsDir 'heartbeat.json'
7
8     BeforeAll {
9         if (-not (Test-Path $logsDir)) { New-Item -ItemType Directory
10     }
11     AfterAll {
12         if (Test-Path $hbPath) { Remove-Item $hbPath -Force }
13     }
14
15     7 references
16     function Write-Heartbeat {
17         param(
18             [string]$Status,
19             [double]$CPU,
20             [double]$RAM,
21             [double]$Disk,
22             [string]$Path
23         )
24         $obj = [pscustomobject]@{
25             timestamp = (Get-Date).ToString('o')
26             status = $Status
27             cpu = [math]::Round($CPU, 2)
28             ram = [math]::Round($RAM, 2)
29             disk = [math]::Round($Disk, 2)
30             pid = $PID
31         }
32         $json = $obj | ConvertTo-Json -Depth 3
33         $json | Set-Content -Path $Path -Encoding UTF8
34     }
35
36     It 'legt heartbeat.json an und enthält gültige Felder' {
37         Write-Heartbeat -Status 'running' -CPU 23.1 -RAM 41.4 -Disk 7;
38         (Test-Path $hbPath) | Should -BeTrue
39
40         $hb = Get-Content $hbPath -Raw | ConvertFrom-Json
41         $hb.status | Should -Be 'running'
42         $hb.pid | Should -Be $PID
43         [double]$hb.cpu | Should -BeGreaterThanOrEqual 0
44         [double]$hb.ram | Should -BeGreaterThanOrEqual 0
45         [double]$hb.disk | Should -BeGreaterThanOrEqual 0
46         $hb.timestamp | Should -Not -BeNullOrEmpty
47     }
48 }
```

ConfigValidation.Tests.ps1 file:


```

mMonitoring-Rafael_Santos > > ConfigValidation.Tests.ps1 > Describe 'Config-Validierung' {} > It 'lädt eine g
1  # Tests für das Laden und Validieren der config.json
2
3  Run tests | Debug tests
4  Describe 'Config-Validierung' {
5      $distDir = Join-Path $PSScriptRoot '..\dist'
6      $configPath = Join-Path $distDir 'config.json'
7
8      BeforeAll {
9          if (-not (Test-Path $distDir)) { New-Item -ItemType Directory
10             @"
11             "cpu_threshold": 85,
12             "memory_threshold": 80,
13             "disk_threshold": 15,
14             "notify_timeout_seconds": 5
15         }
16         @" | Set-Content -Path $configPath -Encoding UTF8
17     }
18     AfterAll {
19         if (Test-Path $configPath) { Remove-Item $configPath -Force }
20     }
21
22     2 references
23     function Load-Config {
24         param([string]$Path)
25
26         if (-not (Test-Path $Path)) {
27             throw "Konfiguration nicht gefunden: $Path"
28         }
29         $cfg = Get-Content $Path -Raw | ConvertFrom-Json
30
31         foreach ($k in 'cpu_threshold', 'memory_threshold', 'disk_thres
32             if (-not ($cfg.PSObject.Properties.Name -contains $k)) {
33                 throw "Fehlender Schlüssel: $k"
34             }
35             $v = $cfg.$k
36             if (-not ($v -is [int])) { throw "Ungültiger Typ für $k: $
37             if ($v -lt 1 -or $v -gt 99) { throw "$k außerhalb 1..99: $
38         }
39
40         if (-not ($cfg.PSObject.Properties.Name -contains 'notify_time
41             $cfg | Add-Member -NotePropertyName 'notify_timeout_secon
42         } elseif (-not ($cfg.notify_timeout_seconds -is [int])) {
43             $cfg.notify_timeout_seconds = 5
44         }
45
46         return $cfg
47     }
48     It 'lädt eine gültige Konfiguration' {
49         $cfg = Load-Config -Path $configPath
50         $cfg.cpu_threshold | Should -Be 85
51         $cfg.memory_threshold | Should -Be 80
52         $cfg.disk_threshold | Should -Be 15
53         $cfg.notify_timeout_seconds | Should -Be 5
54     }
55
56     It 'wirft Fehler bei fehlenden Keys' {
57         $bad = Join-Path $distDir 'bad.json'
58         @"
59         { "cpu_threshold": 90, "memory_threshold": 80 }
60     @" | 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
```

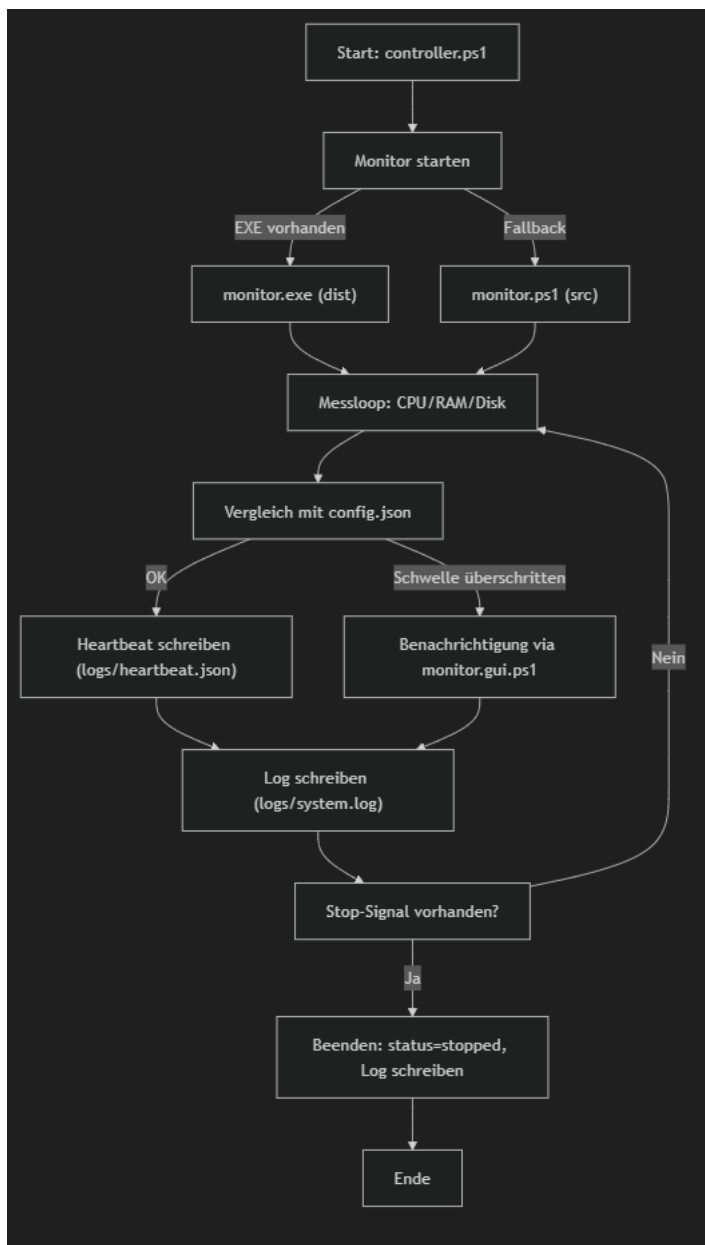
```
Set-ExecutionPolicy RemoteSigned -Scope CurrentUser -Force
```

12. Ablaufdiagramm

Thursday, December 11, 2025

8:52 AM

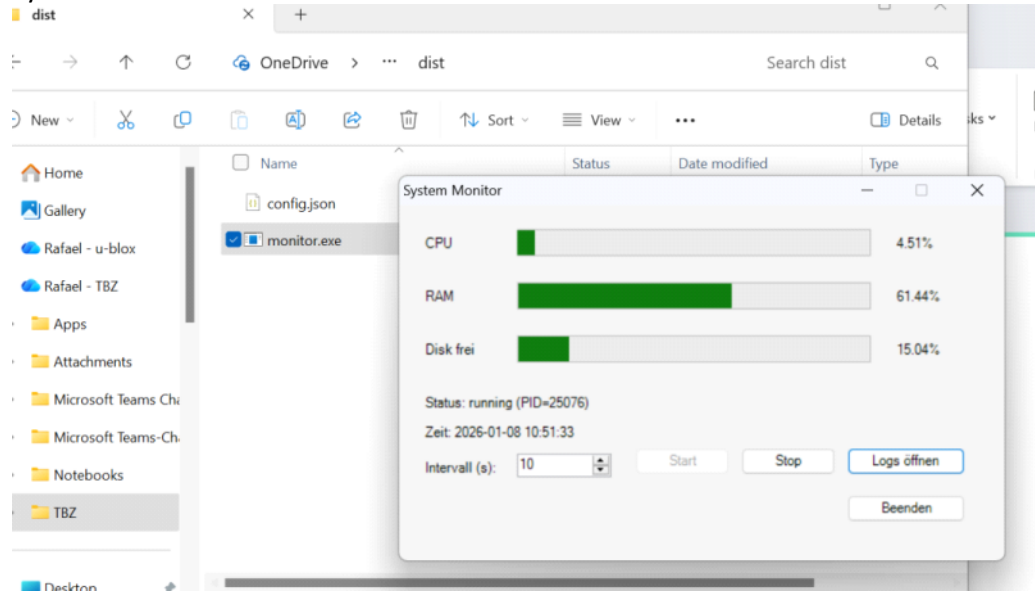
```
LB02-Projekt2-SystemMonitoring-Rafael_Santos > Skript_Ablaufdiagramm-M122-LB02_Projekt2.md
1  ```mermaid
2  flowchart TD
3      A["Start: controller.ps1"]
4      B["Monitor starten"]
5      C["monitor.exe (dist)"]
6      D["monitor.ps1 (src)"]
7      E["Messloop: CPU/RAM/Disk"]
8      F["Vergleich mit config.json"]
9      G["Heartbeat schreiben (logs/heartbeat.json)"]
10     H["Benachrichtigung via monitor.gui.ps1"]
11     I["Log schreiben (logs/system.log)"]
12     J["Stop-Signal vorhanden?"]
13     K["Beenden: status=stopped, Log schreiben"]
14     L["Ende"]
15
16     A --> B
17     B -->|EXE vorhanden| C
18     B -->|Fallback| D
19     C --> E
20     D --> E
21     E --> F
22     F -->|OK| G
23     F -->|Schwelle überschritten| H
24     G --> I
25     H --> I
26     I --> J
27     J -->|Nein| E
28     J -->|Ja| K
29     K --> L
```

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, letzte 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
```