Overvågning af netværk og webtjenester med PowerShell

IT-Teknolog – Afsluttende Eksamensprojekt

Navn: Latif Caliskan

Uddannelse: IT-Teknolog

Skole: Zealand Køge

Vejleder: Zuhair Haroon Khan



Indhold

[1. Indledning 3](#_Toc199428084)

[2. Problemformulering 4](#_Toc199428085)

[3. Formål med projektet 5](#_Toc199428086)

[4. Teori og teknik 6](#_Toc199428087)

[5. Metode og udviklingsproces 7](#_Toc199428088)

[6. Produktbeskrivelse 11](#_Toc199428089)

[6.1 Script 12](#_Toc199428090)

[6.2 GUI 14](#_Toc199428091)

[7. Test og validering 19](#_Toc199428092)

[8. Brugervejledninger og bilag 20](#_Toc199428093)

[9. Brugeroplevelse og tilgængelighed 21](#_Toc199428094)

[10. Sikkerhed og databeskyttelse 21](#_Toc199428095)

[11. Fejlhåndtering og sikkerhed 22](#_Toc199428096)

[12. Design og implementering 25](#_Toc199428097)

[13. Analyse og refleksion 26](#_Toc199428098)

[14. Konklusion 27](#_Toc199428099)

[15. Perspektivering 28](#_Toc199428100)

[16. Litteraturliste 29](#_Toc199428101)

[17. Bilag 30](#_Toc199428102)

[Bilag 1.1 – Script: overvaagning.ps1 30](#_Toc199428103)

[Bilag 1.2 – GUI-script 35](#_Toc199428104)

[Bilag 1.3 – config.json 44](#_Toc199428105)

[Bilag 1.4 – Log-eksempel 47](#_Toc199428106)

[Bilag 1.5 – Screenshots 47](#_Toc199428107)

[Bilag 1.6 – Anvendelse af ChatGPT i projektet 51](#_Toc199428108)

[Bilag 1.7 – index.html 51](#_Toc199428109)

[Bilag 1.8 – server.js 55](#_Toc199428110)

# 1. Indledning

I små og store virksomheder er det vigtigt, at hjemmesider og andre online-adresser kan tilgås uden problemer. Det kan få konsekvenser for både arbejdet og økonomiet, hvis tjenesten eller forbindelsen ikke virker, at opdage fejlen så tidligt som muligt er en kæmpe fordel

Dette projekt går ud på at lave et værktøj, som skal automatisk tjekke om IP-adresser eller webadresser virker, dermed skal den give fejlkode, hvis der går noget galt. Overvågningsværktøjet er også bygget på den måde, som gøre at den kan køres automatisk og bruges manuelt, hvilket gøre at den er brugervenlig og nem at bruge for IT-folk.

PowerShell er blevet brugt, fordi det er stærkt og fleksibelt værktøj, den er også egnet til scripting og automatisering. For at sikre brugervenligheden er der blevet lavet en GUI og en webinterface. Dette er med til at understøtte at man ikke behøver at være en teknisk bruger for at kunne bruge værktøjet.

Projekt giver indblik i, hvordan man med teknologier som ping og http og andre metoder som log filer og konfig-filer, kan bygge et system, der gøre at man kan holde overblik og sørge for tryghed i hverdagen, hvor man slipper for at købe dyre programmer, dermed kan man virksomhederne spare økonomisk.

# 2. Problemformulering

Hvordan kan jeg med PowerShell udvikle et script, der automatisk overvåger IP-adresser og HTTP/HTTPS-tjenester, gemmer resultater, og sender notifikationer via e-mail, hvis der opstår fejl?

# 3. Formål med projektet

Formålet med dette projekt er at udvikle et PowerShell-baseret overvågningssystem, som kan bruges til at teste tilgængelighed af netværksressourcer som IP-adresser og webtjenester. Målet er at skabe et script, der kører automatisk, registrerer resultater og reagerer hurtigt på fejl. Løsningen skal være både funktionel og brugervenlig, og kunne anvendes af personer uden teknisk erfaring med scripting, bl.a. via en simpel grafisk brugergrænseflade (GUI).

Systemet skal understøtte følgende funktioner:

✔ Læse overvågningsopgaver fra en konfigurationsfil i JSON-format

✔ Udføre både ping-tests (ICMP) og HTTP/HTTPS-tests

✔ Logge resultaterne i en CSV-logfil til senere analyse

✔ Gemme tidspunktet for seneste kørsel i en separat state-fil

✔ Køre automatisk og periodisk via Windows Task Scheduler

✔ Udsende fejlnotifikationer via e-mail, og valgfrit via en Microsoft Teams-webhook

Projektet demonstrerer mine færdigheder inden for scripting, netværksteknologi, automatisering, filhåndtering og simpel brugergrænseflade-udvikling. Det afspejler et realistisk behov i it-drift og support, og er inspireret af erfaringer fra min praktik

# 4. Teori og teknik

Projektet tager udgangspunkt i netværksovervågning og fejlhåndtering ved hjælp af automatisering og scripting i PowerShell. Løsningen kombinerer forskellige teknologier, som samlet set muliggør et fleksibelt og udvideligt system til overvågning af både netværksinfrastruktur og webtjenester.

ICMP (Ping)

ICMP også kendt som Internet Control Message Protocol bliver brugt til at sende ping-forespørgsler. En ping-test sender en echo-request til den ip adresse der er specificeret, hvis man for et svar beviser det at forbindelsen er aktiv. I PowerShell bruges funktionen Test-Connection, der returnere svartid og netværksfejl, som nedetid eller andre utilgængelig enheder. [[1]](#footnote-1)

HTTP/HTTPS

HTTP/HTTPS bliver anvendt til at teste tilgængeligheden af de webtjenester. Man kan bruge HTTP/HTTPS til at tjekke om hjemmesiden eller webtjenesten virker, hvor man vil modtage et svar, eksempel 200, der indikerer at alt virker. Invoke-WebRequest bruges I PowerShell, der skal vise om siden svarer korrekt eller om der er fejl.[[2]](#footnote-2)

Scripting og automatisering i PowerShell

PowerShell er blevet anvendt i dette projekt, fordi det fungerer godt med Windows den er også nem at bruge til systemarbejde, filer og netværk. Man kan bruge PowerShell til simple script og avanceret. PowerShell kan også bruges til at lave GUI, som gøre at programmet er nemt at bruge.

Dataformater: JSON og CSV

Systemet bruger JSON-filer til konfiguration. JSON er nem at oprette, ændre og udvide overvågningsopgaver uden at ændre koden. Opgaverne defineres med ping eller http, hvor den bruger email adresse til notifikation. Alt logning i overvågningen bliver logget i CSV-format, som gemmer data og kan åbnes direkte i noter eller databaser.

E-mailnotifikation via SMTP

I koden er der blevet brugt PowerShell-cmdleten Send-MailMessage, der automatisk sender en e-mail til adressen der er henvist til i konfig-filen, hvis testen slår fejl. Email sendes gennem gmail SMTP-server, hvor der er SSL-kryptering og autentificering inkluderet. Dette gøre at de relevante personer for besked så fejlen kan opdages.[[3]](#footnote-3)[[4]](#footnote-4)

Webteknologier

I projektet er der også udviklet et webinterface gennem Node.js og Express, hvor man kan styre overvågningsværktøjet fra en browser. Gennem browseren kan køre test, se resultater og redigere i logfilen. Websiden bruger JavaScript til at sende beskeder til serveren, som så kører PowerShell-scriptet. For at gøre det mere brugervenligt, er der lavet en bat-fil, der starter webserveren. Herefter skal man selv tilgå browseren gennem localhost. [[5]](#footnote-5)

# 5. Metode og udviklingsproces

I projektet er Scrum blevet brugt som udviklingsmetode, der er tilpasset til en individuel arbejdsform. Scrum er en agil arbejdsmetode, der fremmer løbende forbedring, udvikling, dermed er den med til at skabe en fleksibel planlægning. Selvom Scrum normalt benyttes i teams, er det blevet justeret så det er anvendt effektivt i et solo-projekt som denne.

Anvendelse af Scrum-elementer

Produktbacklog og Kanban:

De ting, som skulle laves blev skrevet, som opgaver i produktbacklog, der løbende blev justeret. En visuel kanban-tavle, som jeg har brugt fra (Monday.com) blev anvendt til at dele opgaver i “To do”, “Doing” og “Done”. Dette gav et godt overblik over prioriteringer. Eksempler på backlog-items:

• JSON-konfiguration og opgavelæsning

• Ping- og HTTP/HTTPS-test

• Automatisk fejlnotifikation via e-mail

• Logging til CSV

• Windows Forms GUI med søgning og redigering

• Webinterface (Node.js + PowerShell-integration)

• Redigering og eksport af logfil

Sprintstruktur:

Scrum-processen blev opdelt i sprints af 2–4 dage. Udviklingen var trinvis, hvor én funktion blev færdiggjort og testet, før man begyndte på den næste. Hvert sprint havde mål og afsluttet med test og evaluering.

Daily Scrum / Refleksion:

Der blev lavet daglig logbog, hvor fremdrift og blokering blev skrevet ned inklusiv de næste skridt, der skulle foretages. Dette gjorde at man kunne holde fokus og derefter vurdere, hvad der fungerede og hvad der ellers skulles justeres

Sprint Review og Retrospective:

Når et sprint var færdigt, gik jeg igennem det og testede det, jeg havde lavet, i et kort review. Derefter lavede jeg en retrospektiv, hvor jeg evalueret, hvad der gik godt, hvilke udfordringer der opstod, og hvordan arbejdsprocessen kunne forbedres.

Definition of Done:

Når en opgave blev registreret som done skulle den eksempelvis være testet, virke stabilt og logge korrekt. Når opgaven lever op til kravene kan den anses som ”Done”.

Tilpasning til ændringer:

I processen kom nye ønsker til foreksempel logredigering i GUI’en og gemme log som en separat fil. De blev tilføjet til backlog og dermed prioriteret ud fra deres relevans.

Det afsluttende resultat:

I det sidste sprint samles alle funktioner i et system. Løsningen blev testet, lavede en rapport og en brugervejledning. Fokuset i slutfasen var at systemet fungerede korrekt og var nemt at bruge, med andre ord var der fokus på stabilitet og brugervenlighed.[[6]](#footnote-6)

Et billede, der indeholder tekst, skærmbillede, nummer/tal, software

Indhold genereret af kunstig intelligens kan være forkert.

Status for dokumentations- og udviklingsopgaver er opdelt i "Done", "Working on it", "Stuck" og "Not Started". Værktøjet blev brugt til at planlægge, organisere og følge projektets fremdrift.

Et billede, der indeholder tekst, skærmbillede, nummer/tal, Font/skrifttype

Indhold genereret af kunstig intelligens kan være forkert.

Til styring af opgaver og overblik over fremdrift anvendte jeg Monday.com som digital Kanban-tavle. Her blev både tekniske opgaver oprettet og opdateret løbende. Tavlen blev hermed eksporteret til Excel og indsat i rapporten som dokumentation.

# 6. Produktbeskrivelse

Hovedkomponenten i systemet er et PowerShell-script, som automatisk overvåger en række netværksadresser og hjemmesider. Scriptet læser sine opgaver fra en ekstern JSON-konfigurationsfil, og bestemmer ud fra et interval, hvornår hver opgave skal køres.

Scriptet understøtter to typer tests:

* **Ping-test (ICMP):** Tjekker om der er svar fra en IP-adresse
* **HTTP/HTTPS-test:** Sender webforespørgsel og tjekker statuskode

Hver test logges i en CSV-logfil med følgende information:

* Tidsstempel
* Adresse
* Resultat (success/fail)
* Fejlbesked (hvis relevant)

Ved fejl sendes en e-mail til den angivne kontaktperson. E-mailen indeholder detaljer om opgaven og fejlen. SMTP-indstillinger og modtager angives i konfigurationsfilen.

Scriptet kan køre automatisk i baggrunden via **Windows Task Scheduler**, så det fx aktiveres hvert 5. minut.

# 6.1 Script

Scriptet er lavet i PowerShell og er hjernen bag hele overvågningssystemet. Det sørger for at teste, om bestemte IP-adresser og hjemmesider virker, og gemmer resultaterne i en logfil. Hvis noget fejler, sender scriptet automatisk en e-mail med information om fejlen.

Scriptet gør følgende:

1. **Læser opgaver fra en konfigurationsfil** (f.eks. hvad der skal pinges eller testes med HTTP) ”$configPath = "C:\IT-Projekt\config\config.json”
2. **Tjekker, om det er tid til at udføre testen** (baseret på hvor lang tid der er gået siden sidste gang) ”"intervalMinutes": 10,”
3. **Kører testen**, enten en ping eller en HTTP/HTTPS
4. **Logger resultatet** i en CSV-fil
5. **Sender e-mail**, hvis noget går galt

Her er et kort eksempel fra scriptet, som viser hvordan det vælger og kører en test:

foreach ($task in $config) {

$taskId = $task.id

$lastRun = if ($state[$taskId]) { Get-Date $state[$taskId] } else { [datetime]::MinValue }

$minutesSinceLast = ($now - $lastRun).TotalMinutes

if ($minutesSinceLast -ge $task.intervalMinutes) {

Write-Host "`nRunning task: $($task.id)"

if ($task.type -eq "ping") {

Run-PingTest -task $task

} elseif ($task.type -eq "http" -or $task.type -eq "https") {

Run-HttpTest -task $task

}

$state[$taskId] = $now.ToString("o")

}

}

Scriptet består af flere funktioner, som gør det nemmere at holde styr på det hele:

* **Run-PingTest**: Pinger en IP 10 gange og ser, hvor mange svar der kommer tilbage
* **Run-HttpTest**: Besøger en hjemmeside og ser, hvilken statuskode der kommer
* **Log-Result**: Skriver testens resultat i logfilen
* **Send-Fejlmail**: Sender e-mail ved fejl baseret på e-mailadresse fra JSON-konfigurationen

**Et billede, der indeholder tekst, skærmbillede, menu, Font/skrifttype

Indhold genereret af kunstig intelligens kan være forkert.**

Billedet viser resultatet af en manuel kørsel af overvaagning.ps1-scriptet i PowerShell. Scriptet henter konfigurationsdata fra en JSON-fil og gennemfører en række netværks- og HTTP/HTTPS-tests. I dette tilfælde er der både ping- og webtestopgaver.

Hver opgave vises med opgave-id og status. Eksempelvis viser ping-studerendeonline, at der er opnået svar i 10 ud af 10 forsøg, som også er specificeret i scriptet ”test-connection”. For de øvrige opgaver vises HTTP-statuskoder (200 OK), hvilket indikerer, at webserverne har svaret korrekt og er tilgængelige ”inwoke-webrequest”. [[7]](#footnote-7)[[8]](#footnote-8)

Testen bekræfter, hermed at scriptet fungerer, som forventet og kan anvendes både manuelt og automatiseret. Dette gøre at scriptet er fleksibelt i forhold til fejlfindingsarbejde og overvågning.

# 6.2 GUI

* For at gøre systemet brugervenligt – også for personer uden PowerShell-erfaring – er der udviklet en simpel grafisk brugerflade (GUI) ved hjælp af Windows Forms i PowerShell. Formålet med GUI’en er at give mulighed for manuelt at teste netværksforbindelser og webtjenester uden at åbne PowerShell-terminalen.
* **Funktionalitet i GUI:**
* Viser en liste over alle overvågningsopgaver defineret i config.json
* Giver mulighed for at vælge og køre en enkelt test manuelt
* Viser resultatet af testen (ping eller HTTP) i et tekstfelt
* Giver adgang til at åbne og redigere logfilen
* Har en søgefunktion i logfilen
* Har en funktion til at slette linjer fra logfilen
* GUI’en er udviklet med fokus på enkelhed og funktionalitet. Den understøtter også danske bogstaver korrekt og viser statusmeddelelser under og efter hver test, så brugeren ved, hvornår testen er færdig.

**Kodeudsnit: Oprettelse af GUI-komponenter**

$form = New-Object Windows.Forms.Form

$form.Text = "Overvågning GUI"

$form.Size = New-Object Drawing.Size(400, 300)

$listbox = New-Object Windows.Forms.ListBox

$listbox.Size = New-Object Drawing.Size(360, 100)

$form.Controls.Add($listbox)

$button = New-Object Windows.Forms.Button

$button.Text = "Kør Test"

$form.Controls.Add($button)

$resultBox = New-Object Windows.Forms.TextBox

$resultBox.Multiline = $true

$form.Controls.Add($resultBox)

**Kodeudsnit: Testfunktion i GUI**

$button.Add\_Click({

if ($listbox.SelectedItem -ne $null) {

$task = ... # Find opgaven

if ($task.type -eq "ping") {

# Udfør ping og vis resultat

} elseif ($task.type -eq "http" -or $task.type -eq "https") {

# Udfør HTTP-test og vis resultat

}

}

})

Der er tilføjet knapper til at åbne logfilen i standardeditor, hvor man kan søge i logfilen og slette linjer. Dette gør at GUI’en er praktisk og fleksibel – selv for brugere uden teknisk baggrund.

Et billede, der indeholder tekst, software, Computerikon, Webside

Indhold genereret af kunstig intelligens kan være forkert.

Skærmbilledet ovenfor viser den grafiske brugerflade (GUI), som er udviklet via PowerShell ved hjælp af Windows Forms. Brugeren kan vælge opgaver fra listen over overvågningsopgaver og trykke på “Start Test” for at begynde en netværks- eller webtest. Testresultatet vil derefter blive vist i resultatfeltet enten med succes eller fejl.

Når man trykker på rediger log vil der åbne et nyt vindue, hvor man kan se logfilens indhold i et format der viser nærmere oplysninger, som tid, adresse, status, detaljer og taskid. Brugeren kan søge i loggen, slette, gemme ændringer og gemme en kopi. Disse funktioner skaber en brugervenlig måde at arbejde med overvågningsdata så man slipper for at manuelt åbne CSV-filer.

GUI’en er designet til at være simpel, så også brugere uden PowerShell-erfaring nemt kan udføre overvågning og fejlsøgning. Dette gør at løsningen kan anvendes i mindre IT-afdelinger, hvor driftsovervågning skal kunne udføres hurtigt og tilgængeligt.

**6.3 Webbaseret brugerflade (Webapp)**

Der er også udviklet en webbaseret brugerflade, hvor man kan styre og få adgang til systemet fra en browser. Dette sikrer at fleksibiliten er værdifuld, fordi overvågning og logkontrol nu kan foretages fra enhver på netværket, hvor man ikke behøver PowerShell installeret.

Webapplikation blev udviklet ved hjælp af Node.js, der bruger Express-rammeværket, som fungerer som et REST-lignende interfade, som kommunikerer med PowerShell-script og logfiler. Webinterface er brugervenlig og fleksibelt, hvor man kan køre overvågningstest og se eller redigere resultaterne.

Webappens funktionalitet omfatter:

• Kørsel af overvågningstest direkte fra browseren

• Visning af testresultater i realtid (uden behov for genindlæsning af siden)

• Visning af tidspunkt for seneste testkørsel

• Indlæsning og visning af hele logfilens indhold i et redigerbart tekstfelt

• Redigering og gemning af logfilens indhold direkte fra webinterfacet

• Sletning af logindhold via et simpelt klik (beholder kolonneheaders)

• Mulighed for at downloade og eksportere logfil som backup

Applikationen består teknisk set af følgende komponenter:

• server.js – Node.js-backend, som håndterer HTTP-forespørgsler, kalder PowerShell-scriptet via child\_process.exec() og læser/skriver logfiler

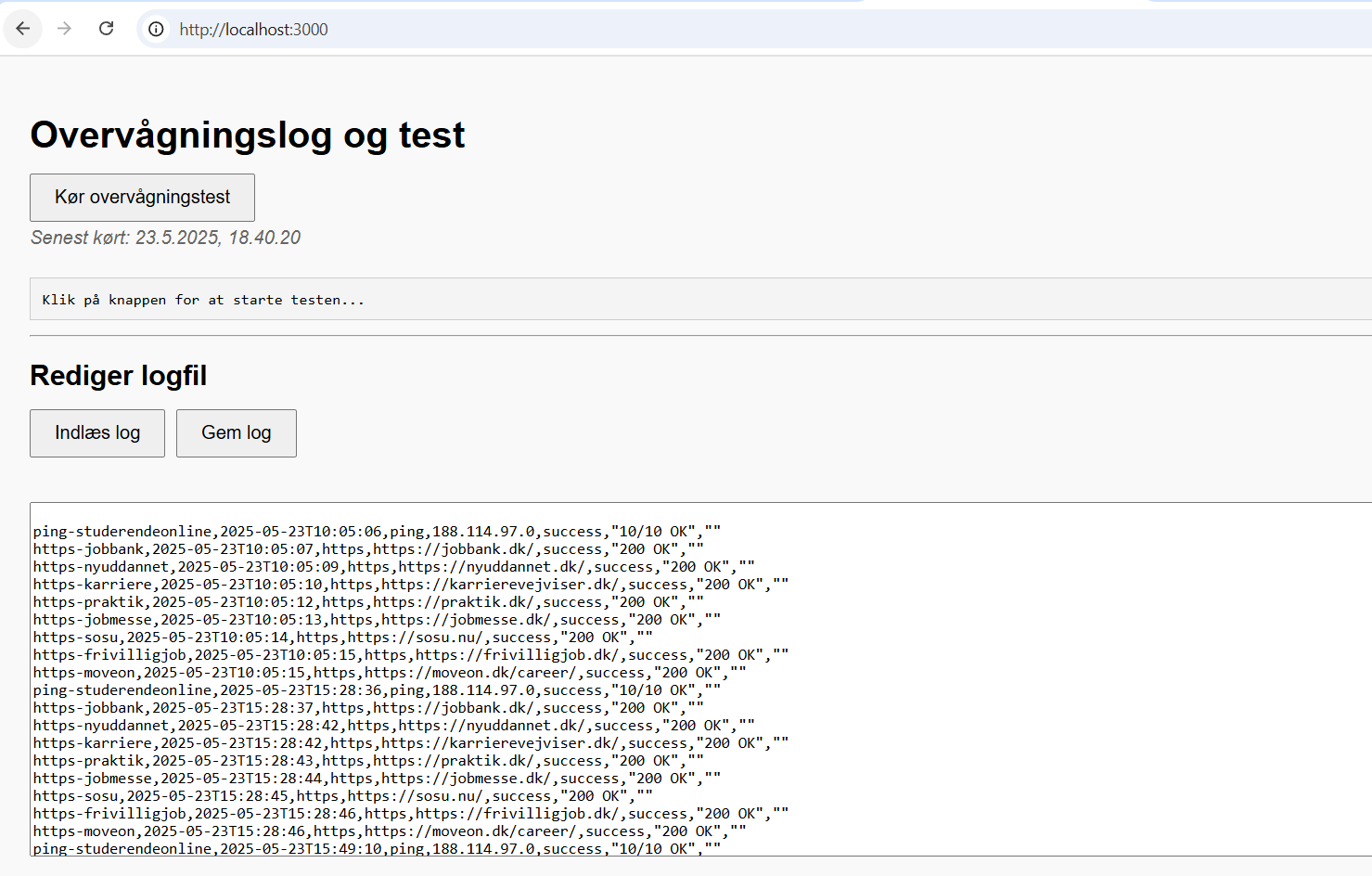
• index.html – Frontend-grænseflade skrevet i HTML/CSS/JavaScript, der kommunikerer med backend via fetch()

• overvaagning.ps1 – Det oprindelige PowerShell-script, som er genbrugt og nu kaldt eksternt via webserveren

Brugeroplevelsen er holdt simpel og let at bruge, med knapper til “Kør test”, “Indlæs log” og “Gem log”, samt mulighed for at redigere logdata. Systemet viser også statusoplysninger og fejl direkte i browseren.

Denne løsning er velegnet til mindre IT-afdelinger eller supportteams, hvor flere brugere kan følge med i status på systemer – uden at man behøver adgang til kommandolinjen eller PowerShell. Webinterfacet kan let udvides med sikkerhedsforanstaltninger som f.eks. adgangskodebeskyttelse, IP-filtrering eller HTTPS, hvis det skal gøres tilgængeligt uden for lokalnetværket.

Udviklingen af webappen har også givet indblik i samspillet mellem PowerShell og moderne webteknologier, og har vist hvordan selv et klassisk script-baseret system kan forenes med en browserbaseret brugeroplevelse.



Dette skærmbillede viser den webbaserede brugerflade, som er udviklet i Node.js og Express. Webappen gør det muligt at køre netværks- og HTTP-tests direkte fra browseren samt læse og redigere overvågningsloggen.

Øverst kan man se en knap med teksten “Kør overvågningstest”, hvor den sender en forespørgsel til serveren og udfører PowerShell-scriptet i baggrunden. Resultatet vises i realtid, og tidspunktet for seneste kørsel bliver opdateret automatisk..

Man kan også finde et redigeringsknap nederst på siden, hvor man kan se logfilen i CSV-format. For at hente data kan man trykke ”indlæs log” eller ”gem log” for at gemme logfilen med ændringer. Ved hjælp af denne funktion kan man overvåge og rette i den uden man bruger PowerShell.

# 7. Test og validering

For at sikre at både scriptet, den grafiske brugerflade (GUI) og webinterfacet fungerede korrekt, blev der udført tests. Formålet var at verificere, at hver enkelt funktion – ping-test, HTTP/HTTPS-test, logning, fejlhåndtering og brugerinteraktion virkede som forventet.

**Manuel testning:**

* Scriptet blev kørt direkte fra PowerShell for at observere output og sikre logning er korrekt og e-mailafsendelse sendes korrekt ved fejl.
* JSON-konfigurationen blev ændret for at inkludere både korrekte og fejlagtige IP-adresser og URL’er. Hvor systemet blev testet for at se om den identificerer fejlen.

**GUI-testning:**

* Brugeren kunne vælge en opgave fra listen og trykke på “Start Test”. Derefter blev resultater vist enten ”fejl, ok, 200, 0/10”
* Funktionen “Rediger Log” blev testet ved at åbne og søge i logfilen samt gemme ændringer direkte via GUI’en.
* Funktionen for at rydde logfilen og oprette en ny kopi blev også testet og fungerede som forventet.

**Webapp-testning:**

* Webinterfacet blev testet i browser ved at tilgå <http://localhost:3000>. Her kunne overvågningstests køres via en “Kør Test”-knap, og resultaterne blev vist direkte.
* “Indlæs log” og “Gem log” blev testet til at hente og gemme logfilen via webappen. Der blev indlæst både små og store filer uden fejl.
* Funktionen for at downloade en kopi af logfilen samt slette loggens indhold blev verificeret.
* Webappen blev testet i forskellige browsere for at sikre at den virkede korrekt.

**Task Scheduler test:**

* Scriptet blev sat op til at køre automatisk via Windows Task Scheduler med et interval på 5-10 minutter.
* Funktionen blev bekræftet ved at kontrollere tidspunktet i logfilen og verificere, at fejl-e-mails blev sendt ved fejl.

**Screenshots og dokumentation:**

* Skærmbilleder af testresultater, GUI, webinterface og logfiler er inkluderet som bilag for at dokumentere og validere testforløbet.

# 8. Brugervejledninger og bilag

For at understøtte implementeringen og den daglige anvendelse af systemet er der udarbejdet en række brugervejledninger, som er vedlagt som bilag. Formålet med vejledningerne er at sikre, at både tekniske og ikke-tekniske brugere kan betjene systemet korrekt og forstå dets funktionalitet – uanset om de arbejder direkte i PowerShell, via den grafiske brugerflade eller i webinterfacet.

Brugervejledningerne dækker alle relevante dele af systemet:

* **Bilag A – GUI**: Gennemgang af den lokale grafiske brugerflade, herunder hvordan man starter en test, ser resultater og redigerer logfiler.
* **Bilag B – Webapp**: Beskriver den browserbaserede adgang til systemet, herunder hvordan man starter overvågningstests, læser og redigerer logfiler samt ser status i realtid.
* **Bilag C – PowerShell**: Forklarer hvordan scriptet kan køres manuelt, hvilke filer det benytter, og hvordan man kan tilpasse konfigurationen.
* **Bilag D – Automatisk kørsel**: Viser trin for trin hvordan systemet sættes op til at køre automatisk via Windows Task Scheduler, hvilket er relevant i driftssammenhænge.

Vejledningerne er opbygget som praktiske trin-for-trin-instruktioner med fokus på brugervenlighed. De er tiltænkt IT-teknikere, supportmedarbejdere eller systemadministratorer, som skal vedligeholde eller drifte løsningen. Ved at dokumentere anvendelsen på denne måde, understøtter projektet både overdragelse, skalering og videreudvikling.

Dokumentation og brugervejledninger er ofte undervurderede elementer i IT-projekter, men de spiller en afgørende rolle for systemets levetid og vedligeholdelse. I dette projekt sikrer vejledningerne, at løsningen kan overdrages til en ny tekniker uden oplæring, hvilket øger robustheden i praksis og mindsker afhængighed af den oprindelige udvikler.

# 9. Brugeroplevelse og tilgængelighed

Et af de store nøglefaktor af dette projekt er at gøre systemet brugervenligt og tilgængeligt for personer uden teknisk baggrund. Overvågningsløsninger kan være svære, fordi det kræver at man har teknisk kendskab eller kan bruge terminalen, hvilket gøre det svært for mange. Dette er et af grundene til der er sat vægt på GUI og den webbaseret version.

GUI’en er lavet med synlige knapper og opgavestyring i listeform, hvor man nemt kan navigere rundt i GUI’en. GUI’en er nem for brugere uden at man behøver særlig oplæring.

Den webbaserede version er endnu mere brugervenlig, fordi den kan åbnes fra enhver browser og mobile enheder. Der er ikke krav om PowerShell-kendskab. Dette gør at systemet er relevant for eksempelvis helpdesk-medarbejdere eller driftspersonale, der hurtigt skal danne sig et overblik uden at skulle logge ind på servere.

Projektet viser at man kan gøre overvågning og systemadministration nemt uden at man skal være ekspert i området. Ved at integrere de fukntioner med fokus på brugervenlighed vil dette blive et effektivt redskab i mange IT-miljøer, hvor man sætter fokus på hurtig adgang og overblik.

# 10. Sikkerhed og databeskyttelse

Projektet handler først og fremmest om overvågning og automatisering, da er det vigtigt at tænke på datasikkerhed. Sytemet bruger data som netværksadresser, links, fejlkoder og statusbeskeder, der kan afsløre data i det interne system, derfor er det vigtigt man tager højde for sikkerhed.

Loginoplysninger til e-mailserveren (Gmail) er gemt direkte i PowerShell-scriptet. Det er en praktisk løsning til udvikling og test, men i professionelle omgivelser bør sådanne oplysninger håndteres gennem sikre metoder. Det kunne f.eks. være Windows Credential Manager, eller sikre konfigurationssystemer. På disse måder undgår man, at brugernavne og adgangskoder ligger i klartekst.

Logfilen, som indeholder testresultater og fejl, bliver gemt i CSV-format og er frit tilgængelig for brugeren. I en større organisation bør adgang til denne fil begrænses til autoriserede brugere, f.eks. via filsystemets adgangskontroller (NTFS-rettigheder) eller ved at flytte logdata til en sikret database. Derudover bør man overveje, om logningen overholder virksomhedens interne retningslinjer for datalagring og GDPR-compliance, særligt hvis IP-adresser eller personrelaterede oplysninger indgår. I større virksomheder bør man sørge for at det kun er de rigtige personer der har adgang til log-filen

Hvis løsningen en dag skal anvendes uden for lokalnetværket, bør webserveren sikres med autentificering, HTTPS og rate-limiting for at undgå misbrug.

Hvis scriptet kører hvert minut kan det overbelaste det overvågede system, hvilket kan i værste fald ligne DoS angreb, hvor servere eller firewalls kan blokere forbindelsen.

For at ikke sende for mange forespørgsler på kort tid er der anvendt intervaller på 5-10 minutter. På denne måde sikrer man stabil overvågning.

Samlet set viser projektet, at også simple scripts kræver overvejelser om sikkerhed – især når de interagerer med netværk, filer og tredjepartstjenester som e-mail. Gennem refleksion og afgrænsning er der i dette projekt taget bevidste valg, og det er tydeligt, at en fremtidig implementering med fokus på drift og sikkerhed vil kræve yderligere foranstaltninger.

# 11. Fejlhåndtering og sikkerhed

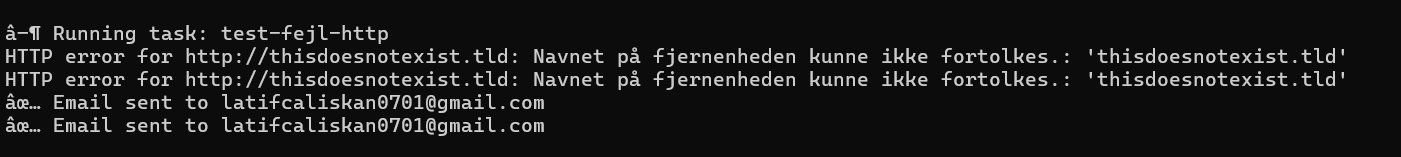
Systemet er udviklet med særlig fokus på robusthed og håndtering af fejl, hvilket er essentielt i et overvågningsmiljø. Fejlhåndtering og sikker overførsel af information spiller en central rolle i at sikre, at eventuelle problemer opdages og rapporteres hurtigt og pålideligt.

**Fejlhåndtering i scriptet**

Hver opgave, som scriptet udfører – uanset om det er en ping-test eller en HTTP/HTTPS-forbindelse – bliver kontrolleret for fejl. Hvis en opgave fejler, bliver dette ikke kun markeret internt, men også registreret i systemets logfil. En sådan fejl kan fx være, at en IP-adresse ikke svarer, eller at en webserver returnerer en fejlstatus (som 404 eller 500).

Fejlen bliver automatisk skrevet til logfilen sammen med oplysninger som tidspunkt, opgavens ID, adresse og en detaljeret fejlbeskrivelse. Ud over at gemme informationen lokalt bliver der også sendt en e-mailnotifikation til den ansvarlige systemadministrator.

**Figur 1** viser et konkret eksempel fra PowerShell-terminalen, hvor en HTTP-test mislykkes, fordi adressen "thisdoesnotexist.tld" ikke kan fortolkes. Systemet registrerer fejlen og sender derefter automatisk en e-mail til den opsatte adresse.



**Figur 1 – Eksempel på fejlhåndtering i PowerShell-scriptet**  
En ugyldig HTTP-adresse udløser fejl, som fanges og rapporteres med det samme via e-mail.

**Automatisk e-mailnotifikation**

Når en fejl opstår, genererer scriptet automatisk en e-mail, som sendes til den kontaktperson, der er defineret i konfigurationen. Denne besked indeholder alle relevante informationer om fejlen – herunder typen af opgave, tidspunktet for fejlen, den pågældende adresse samt selve fejlbeskeden.

Et billede, der indeholder tekst, skærmbillede, Font/skrifttype, nummer/tal

Indhold genereret af kunstig intelligens kan være forkert.

Som illustreret i **Figur 2**, modtages e-mailen med en overskuelig struktur, som gør det nemt at identificere og reagere på problemet. Dette er en stor fordel i miljøer, hvor hurtig fejlfinding og reaktionstid er vigtig.

**Figur 2 – Fejlnotifikation sendt til systemadministratorens e-mail**  
E-mailen indeholder detaljeret information om fejlopgaven og gør det muligt at reagere hurtigt på problemer.

**Sikkerhedsforanstaltninger**

For at beskytte følsomme oplysninger og undgå uautoriseret adgang, er der indført flere sikkerhedsforanstaltninger:

* Adgangskoden til den anvendte Gmail-konto gemmes som en SecureString i PowerShell for at forhindre, at den vises i klartekst i systemets hukommelse.
* Scriptet og de tilhørende logfiler gemmes i en dedikeret mappe (C:\IT-Projekt) med korrekte fil- og mappetilladelser.
* Konfigurationsfilen (JSON) kan ikke ændres fra GUI’en, hvilket forhindrer ikke-tekniske brugere i at ændre i scriptets kernefunktioner.

Disse tiltag er med til at sikre, at scriptet både fungerer stabilt og sikkert – også i miljøer hvor flere brugere har adgang til systemet.

**HTTP-fejl uden statuskode**  
Systemet håndterer ikke kun klassiske HTTP-fejlkoder som 404 eller 500, men også tekniske forbindelsesfejl, hvor ingen statuskode returneres. Eksempler på dette kan være, hvis en server ikke svarer, en webadresse er ugyldig, eller netværket er nede. I disse tilfælde udløser PowerShell en undtagelse (exception), som opfanges af scriptets fejlhåndtering. Resultatet logges, og der sendes automatisk en fejlnotifikation via e-mail. Dette sikrer, at selv alvorlige forbindelsesproblemer – som ikke returnerer en HTTP-kode – stadig bliver opdaget og håndteret.

**Udvidelsesmuligheder**

På sigt kan sikkerheden og funktionaliteten udvides yderligere. Eksempler på dette kunne være:

* Integration med Microsoft Teams eller Slack via webhook for hurtigere notifikationer i teammiljøer
* Kryptering af logfiler eller eksport af logdata til en database, hvilket vil muliggøre mere avanceret analyse og datavisualisering
* Implementering af adgangskontrol i webinterfacet, så kun godkendte brugere kan foretage visse handlinger (som fx at gemme ændringer i logfilen)

Denne helhedstilgang til fejl- og sikkerhedshåndtering understøtter systemets driftssikkerhed og gør det mere egnet til brug i professionelle IT-miljøer.

# 12. Design og implementering

Løsningen er teknisk opdelt i tre lag: et scriptlag (PowerShell), et brugergrænsefladelag (GUI i Windows Forms) og et webinterface (Node.js med Express). PowerShell-scriptet er struktureret i funktioner med klart ansvar: konfiguration indlæses fra JSON, tests udføres med Test-Connection og Invoke-WebRequest, og resultater logges til en CSV-fil. E-mailnotifikationer implementeres via Send-MailMessage med SMTP og autentificering, hvilket krævede korrekt håndtering af credentials.

GUI’en er bygget med .NET Windows Forms via PowerShell og giver brugeren mulighed for at vælge en opgave, afvikle testen og se resultatet med farvekodet feedback. Derudover er der implementeret funktioner til at åbne, søge i og redigere logfilen direkte fra brugergrænsefladen.

Det webbaserede interface anvender Node.js og Express og kommunikerer med både filsystemet og PowerShell-scriptet. Brugeren kan afvikle en test via et HTTP-kald til /koertest, som kører scriptet med child\_process.exec, og modtage resultatet i browseren. Logfiler kan hentes, redigeres og gemmes via endpoints som /log og /save-log. Webappen og scriptet deler konfigurations- og logfiler, så begge grænseflader er koblet til samme backend-logik.

# 13. Analyse og refleksion

Projektet har givet en værdifuld mulighed for at arbejde med et praktisk og virkelighedsnært problem inden for IT-infrastruktur og scripting. Gennem arbejdet med overvågningsscriptet og udviklingen af en brugervenlig grafisk brugerflade (GUI) er der opnået en dybere forståelse af både PowerShell-programmering, automatisering og systemadministration.

**Læring og erfaringer:**

* Det har været lærerigt at arbejde med JSON-filer til konfiguration og forstå, hvordan eksterne data kan bruges til at styre scriptets funktionalitet.
* Udviklingen af GUI’en har givet indsigt i Windows Forms, og hvordan man skaber en applikation, der er let at anvende – også for brugere uden teknisk baggrund.
* Fejlhåndtering og e-mailnotifikation har øget forståelsen for, hvordan man kan bygge scripts, der både er pålidelige og informative.
* Udvidelsen med et webbaseret interface bygget i Node.js og Express har givet erfaring med integration mellem PowerShell og webteknologier, herunder brug af server.js, REST-lignende endpoints og frontend-adgang til logfiler og testfunktioner via browser.

**Udfordringer undervejs:**

* En af de største udfordringer var at få e-mailfunktionen til at fungere korrekt med Gmail og SMTP-indstillinger, hvilket krævede tålmodighed og fejlsøgning.
* Derudover skulle GUI’en tilpasses, så den både var overskuelig og funktionel, hvilket tog flere iterationer at opnå.
* I webdelen opstod mindre udfordringer med stihåndtering og eksekvering af PowerShell fra Node.js, som blev løst gennem test og tilpasning.

**Fremtidsperspektiv:**

* I fremtiden kunne systemet udvikles videre med grafiske rapporter, databaseintegration og brugerstyring i webappen.
* Webinterfacet åbner mulighed for, at løsningen kan tilgås fra mobile enheder eller deles internt i organisationer uden krav om PowerShell-adgang.
* Projektet viser, hvordan selv et enkelt PowerShell-script kan give stor værdi og skabe automatisering i en professionel IT-driftssammenhæng – især når det kombineres med en moderne brugerflade.

# 14. Konklusion

Projektets mål var at udvikle et overvågningsværktøj i PowerShell, der automatisk kan overvåge netværksforbindelser og hjemmesider ved hjælp af ping- og HTTP/HTTPS-tests. Dette blev opnået gennem et script, som læser en konfigurationsfil, logger testresultater, sender fejlnotifikationer via e-mail og kan afvikles automatisk med Windows Task Scheduler.

For at gøre løsningen mere brugervenlig er der desuden udviklet en grafisk brugerflade (GUI), som gør det muligt at køre tests manuelt, se resultater og redigere logfiler – alt sammen uden at brugeren behøver kende til PowerShell. Som en ekstra udvidelse blev der i projektets afslutning også udviklet et simpelt webinterface baseret på Node.js og Express, hvor man via en browser kan afvikle tests, hente logfiler og redigere indhold. Dette bidrager til øget fleksibilitet og tilgængelighed i løsningen.

Projektet opfylder alle de krav, der er opstillet i problemformuleringen og kravspecifikationen. Det demonstrerer praktisk anvendelse af teori fra undervisningen, herunder automatisering, scriptprogrammering, fejlhåndtering, brugergrænsefladedesign og systemintegration.

Det færdige produkt kan bruges i en reel driftssammenhæng og kan tilpasses andre miljøer og behov. Projektet viser derfor, at der er arbejdet med et praksisnært og relevant problem, og at den studerende har opnået kompetencer til at udvikle komplette og professionelle IT-løsninger med både backend- og brugerfokus.

# 15. Perspektivering

Projektet har givet et værdifuldt indblik i, hvordan automatisering og overvågning med PowerShell kan spille en central rolle i den daglige IT-drift. Løsningen er et godt eksempel på, hvordan simple scripts og lettilgængelige værktøjer kan effektivisere opgaver, der ellers ville kræve manuelt arbejde og stor opmærksomhed fra IT-teknikere. Både mindre virksomheder og større organisationers IT-afdelinger vil kunne tilpasse og anvende løsningen, da den er fleksibel, let at konfigurere og ikke afhænger af kommerciel software.

Et oplagt næste skridt i videreudviklingen kunne være at udvide notifikationssystemet med SMS-tjenester eller push-notifikationer via mobilapps, så fejlmeldinger når brugeren hurtigere og mere direkte. På den måde kan man reagere på nedetid eller netværksfejl med minimal forsinkelse, hvilket især er vigtigt i driftskritiske miljøer.

Yderligere udvikling kunne også inkludere overvågning af ressourcer som CPU-forbrug, RAM-brug og diskplads, så scriptet dækker bredere aspekter af server- og systemovervågning. På sigt ville man kunne integrere andre PowerShell-moduler til systemmålinger og supplere med Windows Management Instrumentation (WMI) eller performance counters.

På et mere teknisk niveau er der også mulighed for at erstatte den nuværende CSV-baserede logning med en database, f.eks. SQLite eller en cloud-baseret løsning. Det ville muliggøre datavisualisering over tid, statistik og filtrering – fx gennem dashboards i en webapplikation. På den måde bliver det muligt at analysere trends og fejlmønstre, hvilket kan være værdifuldt i forebyggende vedligehold og optimering.

Det samlede system – med både PowerShell, GUI og webinterface – viser, hvordan en enkel løsning kan danne grundlag for mere avancerede systemer, og hvordan kompetencer inden for scripting, webteknologi og brugergrænseflade-design kan kombineres for at skabe reelle, anvendelige værktøjer i praksis.

# 16. Litteraturliste

<https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/test-connection>

https://learn.microsoft.com/en-us/powershell/module/?view=powershell-7.4

Microsoft. (2024). Test-Connection (PowerShell). Hentet fra: https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/test-connection

Microsoft. (2024). Invoke-WebRequest. Hentet fra: https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/invoke-webrequest

Microsoft. (2024). Send-MailMessage. Hentet fra: https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/send-mailmessage

Microsoft. (2024). Windows Forms overview. Hentet fra: https://learn.microsoft.com/en-us/dotnet/desktop/winforms/

JSON.org. (n.d.). Introducing JSON. Hentet fra: https://www.json.org/json-en.html

Atlassian. (n.d.). What is Scrum?. Hentet fra: https://www.atlassian.com/agile/scrum

Monday.com. (2024). Getting started with Kanban boards. Hentet fra: https://support.monday.com/hc/en-us/articles/360002197259-Kanban-View

Stack Overflow. (2023). Discussion threads on PowerShell scripting and automation. Hentet fra: <https://stackoverflow.com/questions/tagged/powershell>

OpenAI. (2025). *ChatGPT* <https://chat.openai.com>

Scrumguiden  
*Jeff Sutherland & Ken Schwaber*  
**Link:** https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf#zoom=100

# 17. Bilag

# Bilag 1.1 – Script: overvaagning.ps1

# Script: monitoring.ps1

# === File paths ===

$configPath = "C:\IT-Projekt\config\config.json"

$statePath  = "C:\IT-Projekt\config\state.json"

$logPath    = "C:\IT-Projekt\logs\overvaagning\_log.csv"

# === Email settings ===

$gmailUser     = "latifcaliskan0701@gmail.com"

$gmailPass     = ConvertTo-SecureString "borw fklh mhgv bcqu" -AsPlainText -Force

$cred          = New-Object System.Management.Automation.PSCredential ($gmailUser, $gmailPass)

$smtpServer    = "smtp.gmail.com"

$smtpPort      = 587

$smtpUseSsl    = $true

# === Helper: write to console and output ===

function Write-Both {

    param([string]$text)

    Write-Output $text

    Write-Host   $text

}

# === Load JSON from file ===

function Load-JsonFile {

    param ($path)

    if (Test-Path $path) {

        return Get-Content $path | ConvertFrom-Json

    } else {

        return @()

    }

}

# === Log result to file ===

function Log-Result {

    param ($task, $status, $details, $error)

    $line = "$($task.id),$((Get-Date).ToString("s")),$($task.type),$($task.address),$status,""$details"",""$error"""

    Add-Content -Path $logPath -Value $line

}

# === Send error email ===

function Send-ErrorMail {

    param ($task, $details, $error)

    $body = @"

Task: $($task.id)

Type: $($task.type)

Address: $($task.address)

Timestamp: $(Get-Date)

Result: $details

Error: $error

"@

    try {

        Send-MailMessage -To $task.alertEmail -From $gmailUser `

            -Subject "Monitoring task failed: $($task.id)" `

            -Body $body `

            -SmtpServer $smtpServer `

            -Port $smtpPort `

            -UseSsl:$smtpUseSsl `

            -Credential $cred

        Write-Both "✅ Email sent to $($task.alertEmail)"

    } catch {

        Write-Both "❌ Email sending failed: $($\_.Exception.Message)"

    }

}

# === Ping test ===

function Run-PingTest {

    param ($task)

    $address = $task.address

    $successCount = 0

    for ($i = 1; $i -le 10; $i++) {

        if (Test-Connection -ComputerName $address -Count 1 -Quiet -ErrorAction SilentlyContinue) {

            $successCount++

        }

    }

    $summary = "$successCount/10 OK"

    $status = if ($successCount -eq 10) { "success" } else { "fail" }

    Write-Both "Ping result for ${address}: $summary"

    Log-Result -task $task -status $status -details $summary -error ""

    if ($status -eq "fail") {

        Send-ErrorMail -task $task -details $summary -error "No response to ping"

    }

}

# === HTTP(S) test ===

function Run-HttpTest {

    param ($task)

    $address = $task.address

    $protocol = if ($address.StartsWith("https")) { "HTTPS" } else { "HTTP" }

    try {

        $response = Invoke-WebRequest -Uri $address -UseBasicParsing -TimeoutSec 5

        $statusCode = $response.StatusCode

        if ($statusCode -ge 200 -and $statusCode -lt 300) {

            Write-Both "$protocol result for ${address}: $statusCode OK"

            Log-Result -task $task -status "success" -details "$statusCode OK" -error ""

        } else {

            Write-Both "$protocol error for ${address}: $statusCode"

            Log-Result -task $task -status "fail" -details "$statusCode Error" -error "HTTP status not in 200-299"

            Send-ErrorMail -task $task -details "$statusCode Error" -error "HTTP status not in 200-299"

        }

    } catch {

        Write-Both "$protocol error for ${address}: $\_"

        Log-Result -task $task -status "fail" -details "Request failed" -error $\_.Exception.Message

        Send-ErrorMail -task $task -details "Request failed" -error $\_.Exception.Message

    }

}

# === Load config and state ===

$config = Load-JsonFile $configPath

$state = @{}

if (Test-Path $statePath) {

    $rawState = Get-Content $statePath -Raw | ConvertFrom-Json

    foreach ($entry in $rawState.PSObject.Properties) {

        $state[$entry.Name] = $entry.Value

    }

}

# === Current time ===

$now = Get-Date

# === Process each task ===

foreach ($task in $config) {

    $taskId = $task.id

    $lastRun = if ($state[$taskId]) { Get-Date $state[$taskId] } else { [datetime]::MinValue }

    $minutesSinceLast = ($now - $lastRun).TotalMinutes

    if ($minutesSinceLast -ge $task.intervalMinutes) {

        Write-Both "`n▶ Running task: $($task.id)"

        if ($task.type -eq "ping") {

            Run-PingTest -task $task

        } elseif ($task.type -eq "http" -or $task.type -eq "https") {

            Run-HttpTest -task $task

        }

        $state[$taskId] = $now.ToString("o")

    } else {

        Write-Both "⏭ Skipping $($task.id) (only $([math]::Round($minutesSinceLast)) min since last run)"

    }

}

# === Save updated state ===

$state | ConvertTo-Json | Set-Content $statePath

# Bilag 1.2 – GUI-script

Add-Type -AssemblyName System.Windows.Forms

Add-Type -AssemblyName System.Drawing

[Console]::OutputEncoding = [System.Text.Encoding]::UTF8

# === Indlæs config ===

$configPath = "C:\IT-Projekt\config\config.json"

$config = Get-Content $configPath -Encoding UTF8 | ConvertFrom-Json

# === Funktion: Log resultat ===

function Log-Result {

param ($taskId, $type, $address, $status, $details, $error)

$logPath = "C:\IT-Projekt\logs\overvaagning\_log.csv"

if (-not (Test-Path $logPath)) {

"TaskId,Tidspunkt,Type,Adresse,Status,Detaljer,Fejl" | Out-File -FilePath $logPath -Encoding UTF8

}

$line = "$taskId,$((Get-Date).ToString("s")),$type,$address,$status,""$details"",""$error"""

Add-Content -Path $logPath -Value $line

}

# === GUI Setup ===

$form = New-Object Windows.Forms.Form

$form.Text = "Netværks- og HTTP Overvågning"

$form.Size = New-Object Drawing.Size(600,500)

$form.Font = New-Object Drawing.Font("Segoe UI", 10)

$form.BackColor = [System.Drawing.Color]::WhiteSmoke

$welcomeLabel = New-Object Windows.Forms.Label

$welcomeLabel.Location = New-Object Drawing.Point(10,10)

$welcomeLabel.Size = New-Object Drawing.Size(560,30)

$welcomeLabel.Font = New-Object Drawing.Font("Segoe UI", 14, [System.Drawing.FontStyle]::Bold)

$welcomeLabel.ForeColor = [System.Drawing.Color]::DarkSlateBlue

$welcomeLabel.Text = ":) Velkommen til Overvågningsværktøj :)"

$form.Controls.Add($welcomeLabel)

# Gruppe: Opgaver

$groupTasks = New-Object Windows.Forms.GroupBox

$groupTasks.Location = New-Object Drawing.Point(10,50)

$groupTasks.Size = New-Object Drawing.Size(560,140)

$groupTasks.Text = "Overvågningsopgaver"

$form.Controls.Add($groupTasks)

$listbox = New-Object Windows.Forms.ListBox

$listbox.Location = New-Object Drawing.Point(10,20)

$listbox.Size = New-Object Drawing.Size(540,100)

$listbox.Font = New-Object Drawing.Font("Segoe UI", 10)

foreach ($task in $config) {

$listbox.Items.Add($task.id + " (" + $task.type + ")")

}

$groupTasks.Controls.Add($listbox)

# Gruppe: Resultat

$groupResult = New-Object Windows.Forms.GroupBox

$groupResult.Location = New-Object Drawing.Point(10,200)

$groupResult.Size = New-Object Drawing.Size(560,120)

$groupResult.Text = "Resultater"

$form.Controls.Add($groupResult)

$resultBox = New-Object Windows.Forms.TextBox

$resultBox.Location = New-Object Drawing.Point(10,20)

$resultBox.Size = New-Object Drawing.Size(540,80)

$resultBox.Multiline = $true

$resultBox.Font = New-Object Drawing.Font("Segoe UI", 10)

$groupResult.Controls.Add($resultBox)

$statusLabel = New-Object Windows.Forms.Label

$statusLabel.Location = New-Object Drawing.Point(10,430)

$statusLabel.Size = New-Object Drawing.Size(570,20)

$statusLabel.Text = "Status: Klar"

$form.Controls.Add($statusLabel)

# Knapper

$button = New-Object Windows.Forms.Button

$button.Location = New-Object Drawing.Point(10,330)

$button.Size = New-Object Drawing.Size(150,40)

$button.Text = "▶ Start Test"

$button.BackColor = [System.Drawing.Color]::LightGreen

$form.Controls.Add($button)

$editButton = New-Object Windows.Forms.Button

$editButton.Location = New-Object Drawing.Point(170,330)

$editButton.Size = New-Object Drawing.Size(150,40)

$editButton.Text = "📝 Rediger Log"

$editButton.BackColor = [System.Drawing.Color]::LightSkyBlue

$form.Controls.Add($editButton)

# === Rediger Log korrekt ===

$editButton.Add\_Click({

$logPath = "C:\IT-Projekt\logs\overvaagning\_log.csv"

if (-not (Test-Path $logPath)) {

[System.Windows.Forms.MessageBox]::Show("Logfil ikke fundet.","Fejl","OK","Error")

return

}

$logForm = New-Object Windows.Forms.Form

$logForm.Text = "Rediger Log"

$logForm.Size = New-Object Drawing.Size(800, 600)

$logForm.StartPosition = "CenterScreen"

$searchBox = New-Object Windows.Forms.TextBox

$searchBox.Location = New-Object Drawing.Point(10,10)

$searchBox.Size = New-Object Drawing.Size(500,20)

$logForm.Controls.Add($searchBox)

$searchButton = New-Object Windows.Forms.Button

$searchButton.Text = "Søg"

$searchButton.Location = New-Object Drawing.Point(520,8)

$searchButton.Size = New-Object Drawing.Size(60,24)

$logForm.Controls.Add($searchButton)

$logTextbox = New-Object Windows.Forms.RichTextBox

$logTextbox.Location = New-Object Drawing.Point(10,40)

$logTextbox.Size = New-Object Drawing.Size(760,460)

$logTextbox.Font = New-Object Drawing.Font("Consolas", 10)

$logTextbox.WordWrap = $false

$logForm.Controls.Add($logTextbox)

$global:logData = @()

try {

$global:logData = Import-Csv $logPath

if ($global:logData.Count -eq 0) {

$logTextbox.Text = "Logfilen er tom."

} else {

$formatted = $global:logData | Format-Table TaskId, Tidspunkt, Type, Adresse, Status, Detaljer -AutoSize | Out-String

$logTextbox.Text = $formatted

}

} catch {

$logTextbox.Text = "Fejl ved indlæsning af logfil.`n$\_"

}

$saveButton = New-Object Windows.Forms.Button

$saveButton.Text = "💾 Gem"

$saveButton.Location = New-Object Drawing.Point(10,510)

$saveButton.Size = New-Object Drawing.Size(100,30)

$logForm.Controls.Add($saveButton)

$clearButton = New-Object Windows.Forms.Button

$clearButton.Text = "🧹 Ryd"

$clearButton.Location = New-Object Drawing.Point(120,510)

$clearButton.Size = New-Object Drawing.Size(100,30)

$logForm.Controls.Add($clearButton)

$exportButton = New-Object Windows.Forms.Button

$exportButton.Text = "📁 Gem en kopi"

$exportButton.Location = New-Object Drawing.Point(230,510)

$exportButton.Size = New-Object Drawing.Size(120,30)

$logForm.Controls.Add($exportButton)

$closeButton = New-Object Windows.Forms.Button

$closeButton.Text = "Luk"

$closeButton.Location = New-Object Drawing.Point(360,510)

$closeButton.Size = New-Object Drawing.Size(100,30)

$logForm.Controls.Add($closeButton)

$searchButton.Add\_Click({

$searchTerm = $searchBox.Text

if ($searchTerm -ne "") {

$index = $logTextbox.Find($searchTerm)

if ($index -ge 0) {

$logTextbox.Select($index, $searchTerm.Length)

$logTextbox.ScrollToCaret()

$logTextbox.Focus()

} else {

[System.Windows.Forms.MessageBox]::Show("Intet fundet.","Søg","OK","Information")

}

}

})

$saveButton.Add\_Click({

try {

$global:logData | Export-Csv -Path $logPath -Encoding UTF8 -NoTypeInformation

[System.Windows.Forms.MessageBox]::Show("Logfil gemt korrekt som CSV.","Gem","OK","Information")

} catch {

[System.Windows.Forms.MessageBox]::Show("Fejl ved gemning: $\_","Fejl","OK","Error")

}

})

$clearButton.Add\_Click({

"TaskId,Tidspunkt,Type,Adresse,Status,Detaljer,Fejl" | Out-File -FilePath $logPath -Encoding UTF8

$logTextbox.Text = "Logfil ryddet – kun header bevaret."

[System.Windows.Forms.MessageBox]::Show("Logfil er ryddet, og klar til ny test.","Ryd","OK","Information")

})

$exportButton.Add\_Click({

$saveDialog = New-Object System.Windows.Forms.SaveFileDialog

$saveDialog.Title = "Gem en kopi af logfilen"

$saveDialog.Filter = "CSV-filer (\*.csv)|\*.csv"

$saveDialog.FileName = "overvaagning\_log\_kopi.csv"

if ($saveDialog.ShowDialog() -eq "OK") {

try {

Copy-Item -Path $logPath -Destination $saveDialog.FileName -Force

[System.Windows.Forms.MessageBox]::Show("Kopi gemt: $($saveDialog.FileName)","Eksporteret","OK","Information")

} catch {

[System.Windows.Forms.MessageBox]::Show("Fejl ved kopiering: $\_","Fejl","OK","Error")

}

}

})

$closeButton.Add\_Click({ $logForm.Close() })

[void]$logForm.ShowDialog()

})

# === Test funktion ===

$button.Add\_Click({

if ($listbox.SelectedItem -ne $null) {

$resultBox.BackColor = [System.Drawing.Color]::White

$resultBox.Text = "Tester... Vent venligst..."

$statusLabel.Text = "Status: Tester..."

$selectedTaskId = $listbox.SelectedItem.Split(" ")[0]

$task = $config | Where-Object { $\_.id -eq $selectedTaskId }

if ($task.type -eq "ping") {

$successCount = 0

for ($i = 1; $i -le 10; $i++) {

if (Test-Connection -ComputerName $task.address -Count 1 -Quiet -ErrorAction SilentlyContinue) {

$successCount++

}

}

$summary = "$successCount/10 OK"

$resultBox.Text = "Ping resultat: $summary`nTest færdig."

if ($successCount -eq 10) {

$resultBox.BackColor = [System.Drawing.Color]::LightGreen

} else {

$resultBox.BackColor = [System.Drawing.Color]::LightSalmon

}

Log-Result -taskId $task.id -type "ping" -address $task.address -status "success" -details $summary -error ""

} elseif ($task.type -eq "http" -or $task.type -eq "https") {

try {

$response = Invoke-WebRequest -Uri $task.address -UseBasicParsing -TimeoutSec 5

$statusCode = $response.StatusCode

if ($statusCode -ge 200 -and $statusCode -lt 300) {

$resultBox.Text = "$($task.type.ToUpper()) resultat: $statusCode OK`nTest færdig."

$resultBox.BackColor = [System.Drawing.Color]::LightGreen

Log-Result -taskId $task.id -type $task.type -address $task.address -status "success" -details "$statusCode OK" -error ""

} else {

$resultBox.Text = "$($task.type.ToUpper()) fejl: $statusCode`nTest færdig."

$resultBox.BackColor = [System.Drawing.Color]::LightSalmon

Log-Result -taskId $task.id -type $task.type -address $task.address -status "fail" -details "$statusCode fejl" -error "HTTP-status uden for 200-299"

}

} catch {

$resultBox.Text = "Fejl ved $($task.type.ToUpper()) forespørgsel: $\_`nTest færdig."

$resultBox.BackColor = [System.Drawing.Color]::LightSalmon

Log-Result -taskId $task.id -type $task.type -address $task.address -status "fail" -details "Fejl ved forespørgsel" -error $\_.Exception.Message

}

}

$statusLabel.Text = "Status: Færdig | Bruger: $env:USERNAME | Klokken: $(Get-Date -Format HH:mm:ss)"

} else {

$resultBox.Text = "Vælg en opgave fra listen først!"

}

})

# === Vis GUI ===

[void]$form.ShowDialog()

# Bilag 1.3 – config.json

[

{

"id": "ping-studerendeonline",

"type": "ping",

"address": "188.114.97.0",

"intervalMinutes": 5,

"alertEmail": "latifcaliskan0701@gmail.com"

},

{

"id": "https-jobbank",

"type": "https",

"address": "https://jobbank.dk/",

"intervalMinutes": 10,

"alertEmail": "latifcaliskan0701@gmail.com"

},

{

"id": "https-nyuddannet",

"type": "https",

"address": "https://nyuddannet.dk/",

"intervalMinutes": 10,

"alertEmail": "latifcaliskan0701@gmail.com"

},

{

"id": "https-karriere",

"type": "https",

"address": "https://karrierevejviser.dk/",

"intervalMinutes": 10,

"alertEmail": "latifcaliskan0701@gmail.com"

},

{

"id": "https-praktik",

"type": "https",

"address": "https://praktik.dk/",

"intervalMinutes": 10,

"alertEmail": "latifcaliskan0701@gmail.com"

},

{

"id": "https-jobmesse",

"type": "https",

"address": "https://jobmesse.dk/",

"intervalMinutes": 10,

"alertEmail": "latifcaliskan0701@gmail.com"

},

{

"id": "https-sosu",

"type": "https",

"address": "https://sosu.nu/",

"intervalMinutes": 10,

"alertEmail": "latifcaliskan0701@gmail.com"

},

{

"id": "https-frivilligjob",

"type": "https",

"address": "https://frivilligjob.dk/",

"intervalMinutes": 10,

"alertEmail": "latifcaliskan0701@gmail.com"

},

{

"id": "https-moveon",

"type": "https",

"address": "https://moveon.dk/career/",

"intervalMinutes": 10,

"alertEmail": "latifcaliskan0701@gmail.com"

}

]

# Bilag 1.4 – Log-eksempel

https-karriere,2025-05-06T14:04:43,https,https://karrierevejviser.dk/,success,"200 OK",""

https-praktik,2025-05-06T14:04:44,https,https://praktik.dk/,success,"200 OK",""

https-jobmesse,2025-05-06T14:04:45,https,https://jobmesse.dk/,success,"200 OK",""

https-sosu,2025-05-06T14:04:46,https,https://sosu.nu/,success,"200 OK",""

https-frivilligjob,2025-05-06T14:04:46,https,https://frivilligjob.dk/,success,"200 OK",""

https-moveon,2025-05-06T14:04:47,https,https://moveon.dk/career/,success,"200 **OK**","

# Bilag 1.5 – Screenshots

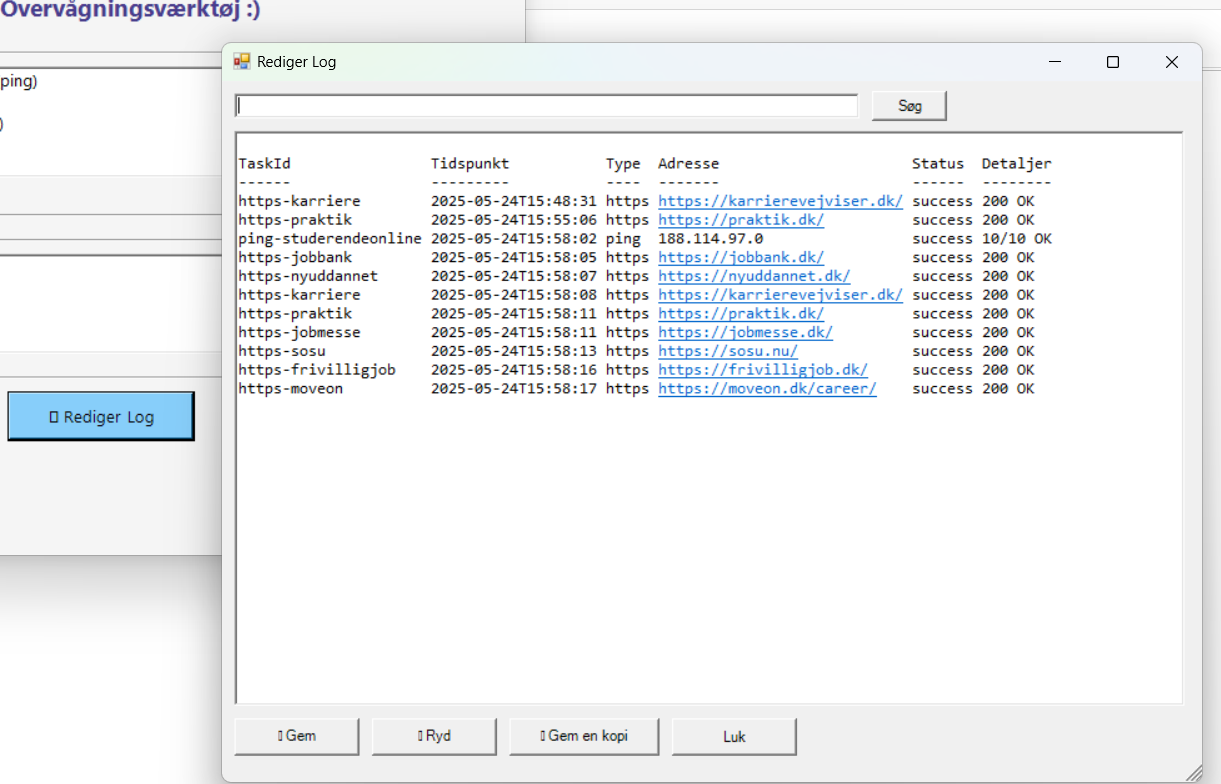
Gui’en i brug:

Et billede, der indeholder tekst, skærmbillede, software, display/skærm/fremvisning

Indhold genereret af kunstig intelligens kan være forkert.

Et billede, der indeholder tekst, skærmbillede, display/skærm/fremvisning, software

Indhold genereret af kunstig intelligens kan være forkert.



Scriptet kørt i Powershell:

**Et billede, der indeholder tekst, skærmbillede, menu, Font/skrifttype

Indhold genereret af kunstig intelligens kan være forkert.**

Opsætning af alle mapper inklusive filer der er inkluderet i projekt.

Et billede, der indeholder tekst, skærmbillede, software, Multimediesoftware

Indhold genereret af kunstig intelligens kan være forkert.

Webapp i brug:

Et billede, der indeholder tekst, skærmbillede, Font/skrifttype, nummer/tal

Indhold genereret af kunstig intelligens kan være forkert.

# Bilag 1.6 – Anvendelse af ChatGPT i projektet

I dette projekt er ChatGPT anvendt som støtteværktøj i både udviklings- og dokumentationsfasen.

ChatGPT er primært brugt til:

* Udformning og tilpasning af PowerShell-scriptet til netværksovervågning
* Generering og forbedring af HTML/JavaScript til webinterfacet
* Strukturering og sproglig optimering af rapporttekster
* Fejlfinding og forslag til forbedringer

Al kode er efterfølgende blevet tilpasset, testet og forstået af mig selv, og modellen er udelukkende brugt som et assisterende værktøj – ikke som erstatning for egen udvikling.

# Bilag 1.7 – index.html

<!DOCTYPE html>

<html lang="da">

<head>

  <meta charset="UTF-8">

  <title>Overvågningslog</title>

  <style>

    body {

      font-family: sans-serif;

      padding: 20px;

      background-color: #f9f9f9;

    }

    h1, h2 {

      margin-bottom: 10px;

    }

    pre {

      background: #f4f4f4;

      padding: 10px;

      border: 1px solid #ccc;

      white-space: pre-wrap;

    }

    textarea {

      width: 100%;

      height: 300px;

      font-family: monospace;

    }

    button {

      padding: 10px 20px;

      font-size: 16px;

      margin: 5px 5px 20px 0;

      cursor: pointer;

    }

    #sidstKoert {

      margin-top: -15px;

      margin-bottom: 25px;

      color: #666;

      font-style: italic;

    }

  </style>

</head>

<body>

  <h1>Overvågningslog og test</h1>

  <button onclick="koerTest()">Kør overvågningstest</button>

  <p id="sidstKoert">Senest kørt: ...</p>

  <pre id="resultat">Klik på knappen for at starte testen...</pre>

  <hr>

  <h2>Rediger logfil</h2>

  <button onclick="loadLog()">Indlæs log</button>

  <button onclick="saveLog()">Gem log</button>

  <br><br>

  <textarea id="logEditor" placeholder="Logfilen vises her..."></textarea>

  <script>

    function koerTest() {

      fetch('/koertest')

        .then(res => res.text())

        .then(data => {

          document.getElementById('resultat').textContent = data;

          opdaterTidspunkt(); // Vis ny tid

        })

        .catch(err => {

          document.getElementById('resultat').textContent = '❌ Fejl ved kørsel af test.';

          console.error(err);

        });

    }

    function loadLog() {

      fetch('/log')

        .then(res => res.text())

        .then(data => {

          document.getElementById('logEditor').value = data;

        })

        .catch(err => {

          alert('❌ Kunne ikke indlæse logfil.');

          console.error(err);

        });

    }

    function saveLog() {

      const content = document.getElementById('logEditor').value;

      fetch('/save-log', {

        method: 'POST',

        headers: { 'Content-Type': 'text/plain' },

        body: content

      })

      .then(res => res.text())

      .then(msg => {

        alert(msg);

      })

      .catch(err => {

        alert('❌ Kunne ikke gemme logfil.');

        console.error(err);

      });

    }

    function opdaterTidspunkt() {

      fetch('/last-run')

        .then(res => res.text())

        .then(data => {

          const felt = document.getElementById('sidstKoert');

          const tid = new Date(data);

          if (!isNaN(tid)) {

            felt.textContent = `Senest kørt: ${tid.toLocaleString()}`;

          } else {

            felt.textContent = 'Senest kørt: (ukendt)';

          }

        })

        .catch(() => {

          document.getElementById('sidstKoert').textContent = 'Senest kørt: (fejl ved hentning)';

        });

    }

    // Hent tidspunkt ved indlæsning

    window.onload = opdaterTidspunkt;

  </script>

</body>

</html>

# Bilag 1.8 – server.js

const express = require('express');

const path = require('path');

const fs = require('fs');

const { exec } = require('child\_process');

const bodyParser = require('body-parser');

const app = express();

const port = 3000;

const logFilePath = path.resolve(\_\_dirname, '../logs/overvaagning\_log.csv');

const lastRunPath = path.resolve(\_\_dirname, 'last\_run.txt'); // 🔹 ny fil til tidspunkt

app.use(express.static(path.join(\_\_dirname, '../public')));

app.use(bodyParser.text());

// GET log file

app.get('/log', (req, res) => {

  res.sendFile(logFilePath);

});

// POST: Save edited log file

app.post('/save-log', (req, res) => {

  fs.writeFile(logFilePath, req.body, 'utf8', (err) => {

    if (err) {

      console.error('❌ Error saving log file:', err);

      return res.status(500).send('❌ Could not save log file.');

    }

    res.send('✅ Log file saved successfully!');

  });

});

// GET: Run PowerShell script

app.get('/koertest', (req, res) => {

  const scriptPath = path.resolve(\_\_dirname, '../script/overvaagning.ps1');

  exec(`powershell -ExecutionPolicy Bypass -File "${scriptPath}"`, (error, stdout, stderr) => {

    if (error) {

      console.error(`❌ PowerShell error: ${error.message}`);

      return res.status(500).send(`❌ PowerShell error: ${error.message}`);

    }

    if (stderr) {

      console.error(`⚠️ PowerShell stderr: ${stderr}`);

      return res.status(500).send(`⚠️ PowerShell stderr: ${stderr}`);

    }

    // 🔹 Gem tidspunkt for seneste kørsel

    const now = new Date().toISOString();

    fs.writeFileSync(lastRunPath, now, 'utf8');

    res.send(`✅ PowerShell output:\n\n${stdout}`);

  });

});

// 🔹 NYT: Send tidspunkt for sidste kørsel

app.get('/last-run', (req, res) => {

  if (fs.existsSync(lastRunPath)) {

    const timestamp = fs.readFileSync(lastRunPath, 'utf8');

    res.send(timestamp);

  } else {

    res.send('Ingen kørsel registreret endnu.');

  }

});

// Start server

app.listen(port, () => {

  console.log(`✅ Webapp running at http://localhost:${port}`);

});

1. https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/test-connection?view=powershell-7.5 [↑](#footnote-ref-1)
2. https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/invoke-webrequest?view=powershell-7.5 [↑](#footnote-ref-2)
3. https://learn.microsoft.com/en-us/powershell/scripting/developer/cmdlet/cmdlet-overview?view=powershell-7.5 [↑](#footnote-ref-3)
4. https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/send-mailmessage?view=powershell-7.5 [↑](#footnote-ref-4)
5. https://www.geeksforgeeks.org/express-js/ [↑](#footnote-ref-5)
6. https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf#zoom=100 [↑](#footnote-ref-6)
7. <https://www.educba.com/powershell-invoke-webrequest/> [↑](#footnote-ref-7)
8. <https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/invoke-webrequest?view=powershell-7.5> [↑](#footnote-ref-8)