|  |
| --- |
| Paul Scherrer Institut |
| Inventurskript zum Abgleich von installierter Hardware mit einer zentralen Datenbank |
| Individuelle Praktische Arbeit 2025 |

|  |
| --- |
| Wernle Yannick  17. März 2025 |

Versionierung

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Datum | Autor | Bemerkung |
| 1.0 | 04.03.2025 | Yannick Wernle | Dokumentstruktur, Teil 1, Informieren, Beginn Planen & Arbeitsjournal |
| 2.0 | 05.05.2025 | Yannick Wernle | Planen weitgehend abgeschlossen, Entscheidungsmatrix gemacht & Arbeitsjournal |

Tabelle 1: Versionierung

Verteiler

|  |  |  |
| --- | --- | --- |
| Person | Funktion | Kontkatdaten |
| Jan Hohenheim | Hauptexperte | [jan@hohenheim.ch](mailto:jan@hohenheim.ch) |
| Gajanthan Vasantharuban | Nebenexperte | [gajanthan.vasantharuban@gmail.com](mailto:gajanthan.vasantharuban@gmail.com) |
| Helge Brands | Fachvorgesetzter | [helge.brands@psi.ch](mailto:helge.brands@psi.ch) |
| Alfred Albisser | Berufsbildner | [alfred.albisser@psi.ch](mailto:alfred.albisser@psi.ch) |
| Yannick Wernle | Kandidat | [yannick.wernle@psi.ch](mailto:yannick.wernle@psi.ch) |

Tabelle 2: Verteiler

Dokumentinformationen

|  |  |
| --- | --- |
| Dokumentname | IPA\_Yannick\_iLO |
| Erstelldatum | 04.03.2025 |
| Abgabedatum | 24.03.2025 / 13:00 Uhr |
| Startblock | 2 (03.03 – 07.03.2025) |

Tabelle 3: Dokumentinformationen

Vorwort

Die Individuelle Praktische Arbeit (IPA) ist die letzte, aber auch eine der wichtigsten Prüfung in der Lehre als Informatiker. Innerhalb von 10 Tagen zeigt der Kandidat seine Kompetenzen und baut eine Applikation komplett von A-Z anhand einer individuellen Aufgabe auf, mitsamt Planung, Implementation und Dokumentation. Im besten Fall ist sie auch dem Lehrbetrieb nach der Durchführung noch von nutzen.

Diese IPA darf ich bei Helge Brands durchführen, bei dem ich seit letztem August sehr viel dazugelernt habe – nicht nur im Bereich Programmieren, sondern auch viel grundsätzliches Wissen über die Informatik. Mit seiner Hilfe konnte ich im Rahmen einer Test-IPA mich gut auf die IPA vorbereiten und wusste, was auf mich zukommt.

In diesem IPA-Bericht will ich nicht nur zeigen, dass ich die Anforderungen an das EFZ erfülle, sondern auch dass ich verstanden habe, womit ich überhaupt arbeite.

Die Arbeit entsteht im zwischen dem 04.03.2025 und dem 24.03.2025, unterbrochen von jeweils 1.5 Schultagen pro Woche.

Danksagung

Diese IPA ist das Ende der Lehre als Informatiker am Paul Scherrer Institut. An der Ausbildung ist aber nicht nur der Lernende beteiligt, sondern auch viele engagierte und kompetente Mitarbeiter, Fachvorgesetzte und Berufsbildner. Für dieses Engagement, das meistens neben der normalen Arbeit erbracht wurde, möchte ich mich herzlich bei allen Beteiligten Bedanken.

Besonders möchte ich mich bei meinem Fachvorgesetzten Helge Brands bedanken, der mich optimal auf diese IPA vorbereitet hat. Weiterhin möchte ich auch meinen ehemaligen Praxisbildnern Thomas Attinger, Hebatallah Mostafa, Markus Winter und auch meinem Berufsbildner Alfred Albisser bedanken. Trotz des etwas chaotischen Berufsbildnerwechsels mitten in der Lehre haben sie sich für die bestmögliche Ausbildung und Beteiligung an Aufgaben und Wissen eingesetzt. Vielen Dank für diese lehrreiche, spannende Lehre!

Inhalt

[1 Teil 1 – Umfeld und Ablauf 6](#_Toc191907039)

[1.1 Aufgabenstellung 6](#_Toc191907040)

[1.1.1 Titel 6](#_Toc191907041)

[1.1.2 Ausgangssituation 6](#_Toc191907042)

[1.1.3 Detaillierte Aufgabenstellung 6](#_Toc191907043)

[1.1.4 Mittel und Methoden 6](#_Toc191907044)

[1.1.5 Vorkenntnisse 6](#_Toc191907045)

[1.1.6 Vorarbeiten 6](#_Toc191907046)

[1.1.7 Verwendete Firmenstandards 6](#_Toc191907047)

[1.1.8 Neue Lerninhalte 6](#_Toc191907048)

[1.1.9 Relevante Arbeiten zur Vorbereitung 7](#_Toc191907049)

[1.2 Projektorganisation 8](#_Toc191907050)

[1.2.1 Organigramm 8](#_Toc191907051)

[1.2.2 Arbeitsumfeld 9](#_Toc191907052)

[1.2.3 Datensicherung und Versionsverwaltung 9](#_Toc191907053)

[1.2.4 Projektmanagementmethode 10](#_Toc191907054)

[1.3 Zeitplanung 10](#_Toc191907055)

[1.3.1 Arbeitstage 10](#_Toc191907056)

[1.4 Meilensteine 12](#_Toc191907057)

[1.5 Zeitplan 13](#_Toc191907058)

[2 Arbeitsjournal 14](#_Toc191907059)

[3 Teil 2 – Projektdokumentation 26](#_Toc191907060)

[3.1 Kurzfassung des IPA Berichts 26](#_Toc191907061)

[3.1.1 Ausgangslage 26](#_Toc191907062)

[3.1.2 Vorgehen 26](#_Toc191907063)

[3.1.3 Ergebnis 26](#_Toc191907064)

[3.2 Informieren 27](#_Toc191907065)

[3.2.1 Projektumfeld 27](#_Toc191907066)

[3.2.2 Systemgrenzen & Schnittstellen 27](#_Toc191907067)

[3.2.3 Use Cases, Personas, User Stories 27](#_Toc191907068)

[3.2.4 Anforderungsanalyse 28](#_Toc191907069)

[3.2.5 Programablaufplan 28](#_Toc191907070)

[3.3 Planen 29](#_Toc191907071)

[3.3.1 Testkonzept 29](#_Toc191907072)

[3.3.2 Testfälle 30](#_Toc191907073)

[3.3.3 Verwendete Technologien 31](#_Toc191907074)

[3.3.4 Weiteres 31](#_Toc191907075)

[3.4 Entscheiden 32](#_Toc191907076)

[3.5 Realisieren 33](#_Toc191907077)

[3.6 Kontrollieren 34](#_Toc191907078)

[3.6.1 Testprotokoll [NUMMER] 34](#_Toc191907079)

[3.7 Auswerten 35](#_Toc191907080)

[3.7.1 Technisches Fazit 35](#_Toc191907081)

[3.7.2 Zukunftsaussichten 35](#_Toc191907082)

[3.7.3 Selbstreflexion 35](#_Toc191907083)

[3.7.4 Persönliches Fazit 35](#_Toc191907084)

[4 Literaturverzeichnis 36](#_Toc191907085)

[5 Abbildungsverzeichnis 36](#_Toc191907086)

[6 Tabellenverzeichnis 36](#_Toc191907087)

[7 Codesnippetverzeichnis 36](#_Toc191907088)

[8 Glossar 36](#_Toc191907089)

[9 Abkürzungsverzeichnis 36](#_Toc191907090)

[10 Selbstständigkeitserklärung 36](#_Toc191907091)

Darstellung

Im Dokument wird für die Einheitlichkeit die folgenden Standards verwendet:

Lorem Ipsum Die Standardschrift des PSI (Calibri) ohne spezielle Formatierung wird für selbstverfasste Texte verwendet.

«Lorem Ipsum»Zitate werden mit «» gekennzeichnet.

**Lorem Ipsum** **Fett markierter Text hat besondere Wichtigkeit.**

Lorem Ipsum Dick unterstrichene Wörter kennzeichnen einen Eintrag im Glossar

Lorem Ipsum Unterstrichene Wörter kennzeichnen Links.

Lorem Ipsum Weisser Text auf schwarzem Grund kennzeichnet Code, resp. Codeausschnitte.



Abbildung 1 – Beispiel Abbildung

QClipboard \*clipboard = QApplication::clipboard();

Codesnippet 1 - Beispiel Code

|  |  |
| --- | --- |
| Definition | Wert |
| A | b |

Tabelle 4: Beispieltabelle

1. Teil 1 – Umfeld und Ablauf
   1. Aufgabenstellung
      1. Titel

Inventurskript zum Abgleich von installierter Hardware mit einer zentralen Datenbank (inventory.psi.ch)

* + 1. Ausgangssituation

Die in unseren Anlagen (SLS und SwissFEL) vorhandenen Serversysteme besitzen ein Wartungsinterface. Dieses Wartungsinterface (iLO von HPE) ist ein eigener Rechner im Rechner und stellt, externen Systemen, die Möglichkeit zur Verfügung auf Hardwareebene über Netzwerk Wartungsaufgaben (Installation, Fehlerdiagnosen und Inventur) durchzuführen. Um einen Überblick über die bestehenden Anlagen zu haben und Änderungen an den Serversysteme schnell erfassen zu können, benötigen wir ein Tool welches aus allen Servern die wichtigen Adressierungnummern (MAC Adressen) und die Seriennummern der Hauptkomponeten (Mainboard) ausliest und ein Reporting erstellt. Dieses Script muss in Powershell geschrieben werden, weil hier herstellerseitig eine Bibliothek zur Auslese der benötigten Parameter schon vorhanden ist.

Zur Ermittlung der gültigen Zielsysteme soll anhand von Suchstrings die für die Anlage gültigen Systeme aus dem Inventory (Web basierter Datenbankdienst) ermittelt und abgefragt werden.

Die abgefragten Serversysteme und ermittelten Daten sollen dann mit den erfassten Daten im Inventory basismässig verglichen werden.

* + 1. Detaillierte Aufgabenstellung

Die Hauptaufgaben ist das Erstellen eine Powershellskripts, dass mit Informationen aus der Inventorydatenbank (Webservice) oder der Implementierung einer Offlineversion Inventurdaten erfasst.

Die Verbindung zum Inventory soll dazu verwendet werden die, nach der Namenskonvention eingetragenen Serversysteme zu finden und ein Abgleich der vorhandenen Seriennummern und MAC Adressen zu machen. Der Report enthält dann die Unterschiede zum Inventory . Um das System für den Anwender transparent zu machen sollen die Zwischenschritte in Dateien zwischengespeichert werden, sodass man auch ohne den Webzugriff auf das Inventory arbeiten könnte. In einer frei wählbaren Konfigurationsdatei werden die Suchstrings spezifiziert nach denen gesucht werden kann.

Die Ergebnisse werden tabellarisch in einer Datei gespeichert um sie kontrollieren zu können. Der automatsche Update ist nicht wünschenswert, da hier zu viele Details sonst implementiert werden müssten.

**Anforderungen:**

a. Hilfe

• Anzeigen einer Hilfeseite mit Parametern bei

i. Anforderung(--help ; -h ; /?; PowerShell help command [Benutzerführungsstil Windows/Unix] )

ii. Bei fehlerhaften Parametern wird die Hilfeseite angezeigt und die Fehler zusätzlich beschrieben

iii. Bei Fehlern in den Konfigurationsdateien wird der Pfad und das Problem signalisiert und die Hilfe angezeigt.

iv. Die Hilfeseite wird bei den Anforderung angezeigt, als Hilfe für den Benutzer.

b. Initialisierung

• Die Bibliotheksversionen (während der Entwicklung und bei der Ausführung) sollen verglichen werden. Nur wenn die Versionen unterschiedlich sind, taucht ein Hinweis dazu auf.

• Es sollen Konfigurationsdateien in JSON initialisiert werden können. Es gibt 2 Arten davon:

i. Eine vorgegebene Datei mit einer Standardkonfiguration.

ii. Eine Leere Konfigurationsdatei, wo die Parameter vom Benutzer selbst eingefüllt werden müssen.

• Parameter ohne einer Konfigurationsdatei sollen nur über die Kommandozeile weitergegeben werden können. Welche Parameter das sind muss der Kandidat entscheiden.

c. Weitere Konfigurationsparameter des PowershellsKripts

• Logparameter

i. Ein und Ausschaltung der Logfunktion

ii. Level des Detailgrades (z.B: Alarmwerte 1-10, Warnwerte 11-20, Infowerte 21-30)

iii. Zieldatei in die die Loginformationen geschrieben werden müssen

• Verbindungsparameter

i. Suchstrings zur Datenbanksuche mit Feldangabe zur Remote-Management Host Abfrage

ii. Direkte Hostnamen des / der Zielsysteme (Standalone-Lösung zur Vorinventarisierung)

iii. Dateipfad zur Konfiguration

• Dateiparameter

i. Ziel der Report-Datei

ii. Suchpfade für Dateien

• Abfrageparameter

i. Pingtest: soll ausschaltbar sein

ii. Seriennummern: das Speichern (CSV-Dateien) von Seriennummern muss ausschaltbar sein

iii. Macadressen: das Speichern (CSV-Dateien) von Macadressen muss ausschaltbar sein

d. Ausführen einer Verbindungskontrolle und Problemlösung

• Einfacher Ping zur Sicherstellung der Erreichbarkeit

• Erstellung von Script-Messages mit Zeitstempeln

• Erstellung eines Logfiles zur Diagnose (TXT Format) des Laufzeitverhaltens

e. Auslese der Parameter

• Verbindungsaufbau mit Nutzerinformationen auf dem Bildschirm

• Weiteres schreiben des Logfiles zur Diagnose (TXT Format/log)

• Erstellung eines Reportfiles (JSON Format)

• Nutzinformationen (Hostname + ausgelesene Seriennummer + Anzahl an Netzwerkinterfaces) in einem Excel importfähigen Format (CSV Format)

f. Tests : Der Kandidat muss für das Testen des Tools Laufzeittests erstellen. Folgende Serversysteme sollen dabei getestet werden:

• Laufzeittest mit den Serversystemen DL380 Gen8/Gen9/Gen10/Gen11

• Laufzeittest mit DL20 Gen10 als nicht Standartsystem

g. Prozessoptimierung

• Anpassung der zur Inventarisierung notwendigen Schlüsselbegriffe, diese müssen vom Kandidaten bestimmt werden

• Optimierung der Datenstruktur in den Exportdateien zum Abgleich der Datenbankinformationen (Inventory). Diese ist vom Kandidaten durchzuführen.

**Anwendungsbeispiele:**

a. lokale Anwendung ohne Datenbankanbindung (Hardwareverantwortliche Person oder Administrator)

Das zu erstellende Skript wird mit den werkseitig (HPE) vordefinierten Parametern (Hostname, User und einem Kennwort z.B. mit Barcode/QR Code Scanner eingelesen[nicht Teil der IPA]) konfiguriert um Inventardaten aus einem einzelnen Rechnern automatisiert auszulesen

b. wöchentlicher automatische Ausführung (Administrator)

Das zu erstellende Skript liest seine Konfiguration ein und sucht in der Inventurdatenbank nach den installierten Servern (Namenkonvention). Das Resultat dieser Suche ( Servern in einem Beschleuniger) enthält unter anderem die Hostname des Managmentinterfaces welches zur Inventur abgefragt werden kann. Die gespeicherten Daten ermöglichen eine Rückverfolgung der eingesetzten Hardware in den abgefragten Servern.

**Personengruppe die diese Skript anwenden:**

Das zu erstellende Skript wird nur von den Administratoren der Kontrollgruppe und von dem Verantwortlichen für die Beschleunigerhardware gebraucht und benutzt. Es wird gebraucht um die manuelle Erfassung zu automatisieren und um Änderungen, die in unseren Anlagen durch Piketteinsätze durchgeführt werden besser Nachvollziehen zu können.

**weitere Datenverarbeitung [nicht Teil der IPA]**

a) das Powershellskript hat einen neuen Rechner ausgelesen:

Die erfassten Daten können per Importfunktion in die Datenbank übernommen werden.

b) es wurde eine Änderung am Wochenende im Pikettdienst vorgenommen:

man sucht anhand des Datums die benötigen Logfiles und überprüft die Hardwareänderung und ob die Informationen mit dem Lager übereinstimmen.

c) Änderungsnachverfolgung von grösseren Umbauarbeiten:

hier kann durch Abgleich von Serienummern und MAC Adressen die exakte Position im Server und auch im Netzwerk nachvollzogen werden.

* + 1. Mittel und Methoden

Software:

* Powershell
* HPEiLOCmdlets

Hardware:

* 4 Server HPE DL380 Gen8/9/10/11
* 1 Server HPE DL20 Gen10

Services:

* inventory - Datenbank /Webzugriff
* git Repository (intern oder extern)
  + 1. Vorkenntnisse
* Powershell
* Netzwerkdiagnose (Wireshark)
* iLO
* http/https Protokoll
* Dateiformate
  + 1. Vorarbeiten

In der Test-IPA wurden die verschiedenen Version von HPEiLOCmdlets für die vorhandene Servermodelle DL380 Gen8/9/10/11 getestet. Diese Vorarbeit war notwendig weil die älteren Versionen zum Teil sehr unterschiedlich auf die verschiedenen HPE Generationen reagiert haben.

Ausserdem hat der Lernende selbstständig die [Prüfungsdokumente](https://www.ict-bbag.ch/qv/) studiert und basierend auf der Wegleitung seine Vorlagen für die Dokumentation und den Zeitplan erarbeitet.

Welche Vorarbeiten wurden durchgeführt?

* + 1. Verwendete Firmenstandards

Im gesamten Dokument wird sich so gut als möglich an die vor kurzem eingeführten Firmenstandards bezgl. Stil und Formatierung verwendet (<https://intranet.psi.ch/de/psi/templates>).

* + 1. Neue Lerninhalte
* Suchanfrage an ein externes System
* Verwendung einer Webschnittstelle
* Abfangen zusätzliche Fehlerquellen
* Neben JSON auch CSV als Datenformat um in Excel zu importieren
  + 1. Arbeiten in den letzten 6 Monaten
* Weiterentwicklung (Markierfunktion) eines Widgets in unserem Displaymanager (Qt/C++)
* Entwicklung von Userpanels aus Expertenpanels zur Steuerung von Servomotoren (Benutzung des Displaymanagers)
* Test-IPA als Vorbereitung
  1. Projektorganisation
     1. Organigramm

Es sind neben dem Kandidat noch 4 weitere Personen direkt am Projekt beteiligt:



Abbildung 2: Organigramm

* + 1. Arbeitsumfeld

Der Kandidat arbeitet jeweils am Montag, Dienstag, Mittwochnachmittag und am Freitag. Am Mittwochmorgen und am Donnerstag ist er aufgrund von Berufsschule und Berufsmatura abwesend.

Er arbeitet während der ganzen IPA am PSI, genauer gesamt im Gebäude WBGB/009, wo er auch sonst arbeitet. Um etwaige Personen darauf hinzuweisen, dass er nicht verfügbar ist, hat er im Outlook einen Blocker für die gesamte IPA-Zeit eingetragen und ist im internen Teams als «Nicht Stören» gekennzeichnet.

Wie wird die Prüfungssituation anderen gegenüber signalisiert?

Welche Technologien benötigt der Kandidat?

Auf welche Maschinen verbindet sich der Kandidat?

Verwendete Software & Tools

* Microsoft Excel (Zeitplan & Entscheidungsmatrix)
* Microsoft Word (Dokumentation/IPA-Bericht)
* Microsoft Visual Studio Code (Entwicklungsumgebung)
  + [PowerShell Erweiterung](https://marketplace.visualstudio.com/items?itemName=ms-vscode.powershell) von Microsoft
* [Draw.io](https://draw.io/)
* [Github](https://github.com/yativilli/hpeilo_inventoryscript)
* [Inventory.psi.ch](https://inventory.psi.ch/)

Verwendete Server

Der Kandidat hat aufgrund seiner Aufgabenstellung den Auftrag, das Skript gegen die folgenden Server zu testen:

* [gfa-sioc-cs-dev](https://inventory.psi.ch/#action=Part&id=SRV0067) (ProLiant DL380 Gen8)
* [sf-sioc-cs-64\_defect](https://inventory.psi.ch/#action=Part&id=SRV0015) (ProLiant DL380 Gen9)
* [gfa-sioc-cs-de3](https://inventory.psi.ch/#action=Part&id=SRV0065) (ProLiant DL380 Gen10)
* [gfa-sioc-cs-de4](https://inventory.psi.ch/#action=Part&id=SRV0224) (ProLiant DL380 Gen11)
* [dl20test](https://inventory.psi.ch/#action=Part&id=SRV0143) (ProLiant DL20 Gen10)
  + 1. Datensicherung und Versionsverwaltung

Alle Daten und Dateien des Projekts werden auf Github abgelegt. Das Repository ist hier erreichbar: [yativilli/hpeilo\_inventoryscript](https://github.com/yativilli/hpeilo_inventoryscript). Es wurde nicht das PSI-interne Git verwendet, da dies bald von Gitlab auf Gitea migriert werden soll. Ausserdem beinhaltet das Skript keinerlei vetrauliche Daten, weswegen es unbedenklich ist, dies ausserhalb dieses gesicherten Rahmens zu verwenden.

Die Software wird mindestens einmal täglich durch Commits auf das Repository gesichert – Git an sich lässt anhand dieser Commits eine geniale Versionsverwaltung zu, weswegen die auch verwendet wird.

[Bild Git Repos]

Der IPA-Bericht wird einerseits lokal auf dem Netzlaufwerk des Lernenden gespeichert und aktiv bearbeitet, andererseits wird er auf dem Onedrive der Schule ebenfalls einmal täglich abgelegt.

Diese dreifache Speicherung garantiert, dass weitgehend unabhängig der Systeme des PSI arbeiten kann – solange Github oder mein Netzlaufwerk erreichbar sind, kann alles andere ausfallen oder ausgetauscht werden.

Als weiteres Element werden auch die Issues von Github verwendet – diese lassen sich optimal nutzen, die einzelnen Implementierschritte zu planen und im Zeitplan zu bleiben. Es wurde aber darauf verzichtet, für jedes Issue einen eigenen Branch zu erstellen, da dies unnötig viel Arbeit und Zeit fressen würde – da dies keine Gruppenarbeit ist, muss man auch keine Merge-Konflikte oder andere Konflikte erwarten.

[Bild Git Issues]

* + 1. Projektmanagementmethode

Um die praktische Arbeit reibungslos durchführen zu können, hat sich der Kandidat dazu entschieden, die Projektmanagementmethode IPERKA zu verwenden.

**IPERKA**

IPERKA ist ein Akronym, welches für «Informieren, Planen, Entscheiden, Realisieren, Kontrollieren, Auswerten» besteht. Die Projektmanagementmethode ist nach dem Wasserfallprinzip aufgebaut, was bedeutet, dass die Phasen aufeinander folgen und nicht iterativ sind.

Die Phase Informieren ist dafür gedacht, sich alle nötigen Informationen über das Projekt im Vorfeld bereits zu erfassen – bspw. welche Schnittstellen es gibt, wo die Grenzen liegen und welche Anforderungen es gibt.

Die Phase Planen ist wie am Namen erkennbar fürs Planen da – dazu gehören unter anderem Zeitplan oder Programmablaufplan.

Die Phase Entscheiden ist dafür gedacht, allfällige wichtige Entscheidungen, die getroffen werden müssen bevor das Projekt realisiert wird, strukturiert und objektiv gefällt werden.

Die Phase Realisieren ist dann die wirkliche Umsetzung des Projekts in die Wirklichkeit.

Darauf folgt die Phase Kontrollieren – dort wird überprüft, ob das Produkt allen Anforderungen entspricht und getestet, ob es denn auch funktionstüchtig ist.

Als abschliessende Phase ist die Auswertung gedacht – dort wird reflektiert, was am Projekt gut und schlecht gelaufen ist, wo es sowohl für die Beteiligten als auch das Produkt noch Verbesserungspotential gibt. Am Ende dieser Phase ist das Projekt abgeschlossen und abgabebereit.

**Alternativen**

Als Alternative zu IPERKA wird in der Informatikerwelt oft Scrum oder eine modifizierte Version davon verwendet – dies ist eine iterative Art des Projektmanagements.

**Entscheid**

Der Kandidat hat sich gegen solche iterative Methoden entschieden, da dies aufgrund der vielen nötigen Personen unnötig kompliziert werden würde – ausserdem ist aufgrund der kurzen Zeit von 10 Arbeitstagen die Umsetzung schlicht zu aufwändig für den Ertrag. Ausserdem ist der Kandidat gut mit IPERKA vertraut, da dies schon in diversen Projekten in der Berufsschule erfolgreich angewendet wurde.

Aufbau der Dokumentation

Die Dokumentation richtet sich nach den Vorgaben des Dokuments «Wegleitung und Weisung (FArbeit.pdf)» von der ICT-Berufsbildung Aargau. Sie ist hier abrufbar:

<https://www.ict-bbag.ch/wp-content/uploads/2024/11/IPA-Leitfaden_FArbeit_INF_2025-1.pdf>

Gemäss den darin enthaltenen Vorgaben ist der IPA-Bericht in zwei Teile geteilt. Der erste Teil bezieht sich auf die Projektstruktur, Umgebung und Aufgabenstellung. Der zweite Teil dokumentiert die eigentliche Arbeit und ist als Folge der Projektmanagement gemäss den Phasen von IPERKA gegliedert.

* 1. Zeitplanung

Welche Festen Daten stehen im Rahmen der IPA fest?

|  |  |
| --- | --- |
| Bezeichnung | Datum |
| Start der Arbeit | 03.03.2025 |
| Erster Besuch (via Zoom) | 05.03.2025 / 14:00 |
| Zweiter Besuch | ?? |
| Abgabe | 24.03.2025 / 13:00 |
| Präsentation und Fachgespräch | ???? |

Tabelle 5: Zeitplanung

* + 1. Arbeitstage



Abbildung 3: Arbeitstage

Die Arbeitstage am PSI bestehen aus 8,3 Stunden pro Tag, weswegen ich im Zeitplan und Arbeitsjournal, weswegen ich auch Im Arbeitsjournal immer damit gerechnet habe. Im Zeitplan habe ich das meist weggelassen, da es dort aufgrund der Struktur in 2h-Blöcken schnell zu Verwirrung kommt.

* 1. Meilensteine

Zum Projektmanagement gehören auch Meilensteine – diese resultieren einerseits aus den Phasenübergängen von IPERKA, andererseits aber auch aus den Daten der praktischen Arbeit selbst.

**Ende der Informierungsphase**

Alle notwendige Informationen zur Durchführung des Projektes sind erfasst, die Anforderungsanalyse ist fertigstellt.

**Ende der Planungsphase**

Alles, was im Vorherein planbar ist, ist geplant, um die Umsetzung später im Projekt zu vereinfachen. Das sind unter anderem der Zeitplan, der Programmablaufplan und die Testfälle.

**Ende der Entscheidungsphase**

Allfällige Entscheidungen, die während der beiden Phasen vorher aufgekommen sind, werden systematisch gefällt, sodass das Projekt durchgeführt werden kann.

**Ende der Realisierungsphase**

Das Projekt wird umgesetzt und ist im bestmöglichen Zustand. Idealerweise konnten alle Anforderungen implementiert werden.

**Ende der Kontrollphase**

Das Produkt der Realisation wird auf Herz und Nieren anhand der geplanten Tests und Anforderungen getestet, um etwaige Verbesserungen zu erfassen.

**Ende der Auswertungsphase**

Das gesamte Projekt und Produkt werden angeschaut und reflektiert, wo es Verbesserungsmöglichkeiten gibt und was in weiteren Durchgängen noch gemacht werden könnte.

**Ende der IPA & Abgabe**

Dies ist der Zeitpunkt, in dem die Arbeit in dem Stand, in welchem sie ist, abgegeben wird und anhand derer die Bewertung durchgeführt wird.

* 1. Zeitplan



1. Arbeitsjournal

Dienstag, 04.03.2025 – Tag 1

|  |  |  |  |
| --- | --- | --- | --- |
| Geplante Arbeiten: | Soll | Erledigte Arbeiten: | Ist |
| * Übernahme der Daten aus PkOrg in Dokumentation   + Arbeitsjournal   + Grundstruktur   + Git-Setup   + Teil 1 ausfülle * Informieren   + Umfeld   + Programmablaufplan   + Anforderungsanalyse * Planen   + Zeitplan   + Konfigurationsdatei | 2h  1,8 h  4,5 h | * Übernahme der Daten aus PkOrg in Dokumentation   + Grundstruktur   + Git-Setup   + Teil 1 ausfüllen * Informieren   + Umfeld   + Programmablaufplan   + Anforderungsanalyse * Planen   + Zeitplan   + Konfigurationsdateien   + Testkonzept,   + Parameterkonzept * Arbeitsjournal | 1,5 h  (0,5 h)  (0,25)  (0,75h)  2 h  (0.5h)  (0.5 h)  (1h)  4,5 h  (1h)  (0,75h)  (1,25h)  (1,5h)  0.3h |
| Überzeit | - | Überzeit - Ist |  |
| Arbeitszeit - Soll | 8,3h | Arbeitszeit - Ist | 8,3h |

|  |
| --- |
| Fazit & Reflexion: |
| Ich bin heute sehr gut vorwärtsgekommen – ich konnte sehr schnell die Grundstruktur meiner Dokumentation erstellen und die Daten von PkOrg übernehmen und habe dann auch den grossen Rest des ersten Teils bereits ausfüllen können, sodass ich bereits mit dem Informieren beginnen konnte. Da ich das Umfeld bereits aus der Test-IPA gut kenne, konnte ich auch die Informationsphase, samt Umfeld, PAP und Anforderungsanalyse gut und zackig bereits abschliessen.  Als weiteres habe ich den Zeitplan gemacht – dieser basiert grösstenteils auf der Anforderungsanalyse, da ich anhand derer genauere Vorhersagen machen konnte, welche Funktionen implementiert werden können. Ich habe die Phasen meist so klein wie möglich aufgeteilt, damit ich für mich schnell sehe, wenn ich aus dem Zeitplan gerate und so schnell wie möglich dann reagieren kann.  Ich bin im Laufe des Tages soweit gekommen, dass ich mit der Planung beginnen konnte – die Konfigurationsdateien, Parameter und das Testkonzept stehen bereits grösstenteils und ich kann morgen mit dem erweiterten Programmablaufplan weiterfahren.  Ich bin für den ersten Tag sehr zufrieden – ich konnte alles abschliessen, was ich mir vorgenommen habe. |

|  |
| --- |
| Recherchen: |
| Ist es möglich, Unit-Tests in Powershell zu programmieren? Ja -> (Pester, kein Datum) |

Tabelle 6: Arbeitsjournal Tag 1

Mittwoch, 05.03.2025 – Tag 2

|  |  |  |  |
| --- | --- | --- | --- |
| Geplante Arbeiten: | Soll | Erledigte Arbeiten: | Ist |
| * Planen   + Detaillierter Programmablaufplan * Erstes Gespräch mit HEX * Entscheiden   + Modulansatz oder Skripting * Arbeitsjournal | 2h  ??  2h  0,25h | * Planen   + Detaillierter Programmablaufplan * Erstes Gespräch mit HEX * Entscheiden   + Kurze Recherche  Modulansatz oder Skriptansatz   + Entscheidungsmatrix * Arbeitsjournal * Git eingerichtet | 1h  1h  2h  0,25h  0,25h |
| Überzeit | - | Überzeit - Ist | 0,25h |
| Arbeitszeit -Soll | 4,25h | Arbeitszeit -Ist | 4,5h |

|  |  |
| --- | --- |
| Fragen Kandidat Yannick Wernle: | Antworten Fachkraft Helge Brands |
| Ich habe mich dazu entschlossen, das Programm mit einer Modulstruktur umzusetzen, ist das in Ordnung? | Ja, das ist okay. |

|  |
| --- |
| Fazit & Reflexion: |
| Ich habe heute mit der Planung weitergemacht – heute stand vor allem ein Programmablaufplan auf dem Plan, den ich anhand des bisherigen gemacht habe. Ich habe noch einen zweiten gemacht, da ich mit dem ersten nur eine grobe Übersicht ermöglichen will, aber mit dem zweiten wollte ich eine grobe Blaupause haben, nach der ich das Programm erstellen kann.  Das erste Gespräch mit dem Hauptexperten war ebenfalls heute und ich denke es ist gut gelaufen – der zweite Termin ist am 18. März geplant. Der HEX hat mir ein paar Hilfreiche Tipps gegeben, die ich sonst vergessen hätte (z.B. dass das Glossar alphabetisch sortiert werden muss usw.).  Ich hatte leichte Schwierigkeiten ein Thema zu finden, zu dem ich eine Entscheidung durchführen kann, mir ist dann aber doch noch eines eingefallen, nämlich ob ich das Skript als Modul oder in Skriptform entwickeln soll – die Anwendung würde sich nur geringfügig ändern, aber die Implementation wäre anders – der Modulansatz hat vor allem in der Installation und Auslieferung eine gute Eigenschaften, die ein lineares Skript halt einfach nicht kann, ich habe aber mit Modulen bisher wenig bis gar nicht gearbeitet. Ich habe mich aufgrund der Entscheidungsmatrix aber am Ende trotzdem für den modularen Ansatz entschieden.  Ich hatte noch etwas Zeit über und habe schonmal für Donnerstag Git aktualisiert und mit dem Repository verbunden, welches ich gestern schon erstellt hatte.  Als nächstes steht am Donnerstag die Implementierung der Grundstruktur an und die Implementation der Parameter. Bevor das Losgeht muss ich aber noch die Issues in Github erstellen. |

|  |
| --- |
| Recherchen: |
| Recherche, ob eher Modulansatz oder Skriptansatz verwendet werden sollte: > (Microsoft Learn, 2021) (Oliver, 2011) ([deleted], 2022) |

Tabelle 7: Arbeitsjournal Tag 2

Freitag, 07.03.2025 – Tag 3

|  |  |  |  |
| --- | --- | --- | --- |
| Geplante Arbeiten: | Soll | Erledigte Arbeiten: | Ist |
| * Realisierung   + Grundstruktur   + Konfigurationsdateien * Issues in Github * Arbeitsjournal | 3h  4h  1h  0,3 h | * Realisierung   + Grundstruktur   + Konfigurationsdateien generieren * Issues in Github * Dokumentieren und Arbeitsjournal | 2h  4,5h  1h  0,8 h |
| Überzeit | - | Überzeit – Ist | 0,2h |
| Arbeitszeit | 8,3h | Arbeiszeit | 8,5h |

|  |  |
| --- | --- |
| Probleme: | Problemlösungen: |
| /?, --help usw. werden auch ohne Anführungszeichen als Auslöser für die Hilfe akzeptiert, -h aber nicht (wird als Parameter erkannt) | -h als [switch] definieren und die Hilfe dadurch manuell auslösen |

|  |  |
| --- | --- |
| Fragen Kandidat Yannick Wernle: | Antworten Fachkraft Helge Brands |
| Soll die Modulversionsüberprüfung eine spezifische Version oder Mindestversion voraussetzen? | Nein, es soll bei einer anderen Version nur der Hinweis kommen, dass eine andere Version verwendet wurde und es Probleme geben könnte. Am besten wäre auch ein Verweis, wie man |

|  |
| --- |
| Fazit & Reflexion: |
| Ich hatte für den Morgen eigentlich nur das Implementieren der Grundstruktur und der Issues geplant, ich bin aber bei der Grundstruktur bedeutend schneller vorwärtsgekommen, als ich es geplant hatte, weswegen ich genug Zeit hatte, am Morgen schon mit der Implementation mit den Konfigurationsdateien zu beginnen.  Ich hatte bei der Grundstruktur keine grösseren, erwähnenswerten Probleme gehabt. Das einzig erwähnenswerte ist, dass ich Probleme damit hatte, -h als Hilfsparameter auch zu akzeptieren, doch dann ist mir eingefallen, dass es als [switch]-Typ ganz einfach implementiert werden kann.  Bei der Generierung der Konfigurationsdateien bin ich zwar auch gut vorangekommen, leider jedoch nicht ganz so schnell wie ich erwartet hatte – das lag vor allem an vielen kleinen Fehlern und Problemen. Dadurch, dass ich heute morgen früher als geplant mit der Grundstruktur fertig war, hält sich der Zeitplan also noch die Waage. Ich hatte am Ende des Tages noch immer genug Zeit, um die Dokumentation weiterzuführen und den nächsten Tag zu planen. |

|  |
| --- |
| Recherchen: |
| Dokumentation, wie man ein Modul aufsetzt -> (Marquette, 2017)  Requires-Statement in Modulen -> (sdwheeler, 2024)  Special Characters in PowerShell -> (sdwheeler, 2024) |

Tabelle 8: Arbeitsjournal Tag 3

Montag, 10.03.2025 – Tag 4

|  |  |  |  |
| --- | --- | --- | --- |
| Geplante Arbeiten: | Soll | Erledigte Arbeiten: | Ist |
| * Realisieren   + Start per Parameter   + Logfunktion   + Pingtest   + Arbeitsjournal | 4h  2h  2h  0.3h | * Realisieren   + Start per Parameter   + Logfunktion   + Pingtest * Dokumentieren * Arbeitsjournal | 4,5h  2,3h  0.5h  1h  0.3h |
| Überzeit -Soll | 0h | Überzeit - Ist | 0.3h |
| Arbeitszeit – Soll | 8.3h | Arbeitszeit - Ist | 8,6h |

|  |  |
| --- | --- |
| Probleme: | Problemlösungen: |
| Beim Update-Config einer existierenden Konfigurationsdatei wurden Daten überschrieben, welche nicht überschrieben werden sollten und solche nicht überschrieben, welche überschrieben werden sollten. | Überprüfung vor dem Update, ob es über die Parameter Änderungen gibt und falls nicht wird der Wert aus der Konfiguration stattdessen ausgelesen. |

|  |
| --- |
| Fazit & Reflexion: |
| Ich bin heute auch wieder gut vorangekommen und konnte vor dem Mittag schon den Start per Parameter weitgehend abschliessen. Grössere, nennenswerte Probleme gab es keine. Der Start per Parameter konnte ich unter anderem deswegen schnell abschliessen, da ich mich dazu entschlossen hatte, im Hintergrund eine Konfigurationsdatei aus den Parametern zu generieren – dadurch kann ich bei der weiteren Implementierung sehr viel Zeit sparen, da ich über die Konfigurationsdatei in jeder Situation an die relevanten Daten komme.  Die Logfunktion an sich konnte ich schnell abschliessen, jedoch musste ich am Nachmittag einen Teil der Start-Per-Parameter-Funktion nochmals umschreiben – es gibt dort eine Update-Config-Funktion, welche anhand der Parameter die Konfiguration ausfüllt – ich hatte Probleme beim umschreiben davon, sodass sie auch bei der Verwendung von einer Konfiguration dazu verwendet werden könnten, einzelne Parameter zu ändern (z.B. nur Loglevel ändern, der ganze Rest soll aus der Konfigurationsdatei kommen). Ich war da etwas langsamer und deswegen hinter dem Zeitplan.  Ich konnte den Zeitplan aber wieder gut aufholen, da ich für den Pingtest nur ca. eine halbe Stunde benötigt hatte anstatt von 2. Ich hatte bei den ersten Implementierungsschritten extra teilweise für auch für die *einfacheren* Augaben jeweils 2 Stunden eingeplant, um eben genau solche Schwankungen ausgleichen zu können. Ich bin also wieder im Zeitplan und konnte die restliche Zeit fürs Dokumentieren nutzen. |

|  |
| --- |
| Recherchen: |
| Approved Verbs für PowerShell Funktionen -> (sdwheeler c.-h. T.-S., 2024)  Überprüfung ob [switch] verwendet wurde -> (JaapBrasser, 2019) |

Tabelle 9: Arbeitsjournal Tag 4

Dienstag, 11.03.2025 – Tag 5

|  |  |  |  |
| --- | --- | --- | --- |
| Geplante Arbeiten: | Soll | Erledigte Arbeiten: | Ist |
| * Implementation   + Abfrage Inventory * Arbeitsjournal | 8h  0.3h | * Implementation   + Abfrage Inventory   + Abfrage von ILO * Arbeitsjournal * Dokumentation | 3 h  4,5h  0.3 h  0.5 h |
| **Überzeit** | **0h** |  | **0h** |
| Arbeitszeit | 8.3h |  | 8.3h |

|  |
| --- |
| Fazit & Reflexion: |
| Ich bin heute weitaus schneller und besser vorangekommen als eingeplant – ich habe anstatt der 8h für die Implementierung der Inventory-Abfrage nur ca. 3h gebraucht, da sie weitaus weniger komplex war als eigentlich gedacht. Ich konnte daher schon am späten morgen und am Nachmittag mit der Abfrage ohne Inventory beginnen, also der eigentlichen ILO-Abfrage. Wenn ich in diesem Tempo weitermache, dann habe ich gegen Ende der IPA vermutlich noch einige Zeit über – diese könnte ich gut einerseits für die Dokumentation verwenden und ich könnte, falls ich Zeit habe, Unittests doch noch implementieren.  Am Nachmittag bin ich gut vorwärtsgekommen, ich konnte aber die Abfrage von ILO noch nicht abschliessend, da es für die verschiedenen Strukturen in den ILO-Versionen teils sehr (zeit-)aufwändig ist, diese auf den gleichen Stand zu bringen. Bei den Teilen, die ich heute implementiert habe, sind jeweils ILO 4 & 6 abgedeckt – ich muss aber noch testen, inwiefern sich ILO 5 und die anderen Server verhalten. Da ich aber ja mehr Zeit habe wegen meines Tempo heute morgen, sollte dies hoffentlich keine grösseren Einflüsse auf den Zeitplan haben.  Viel mehr gibt es nicht zu erwähnen, da nicht viel erwähnenswertes passiert ist, sondern einfach die Arbeit zwar sehr zeitaufwändig, aber gleichzeitig simpel war. |

|  |
| --- |
| Recherchen: |
| Select-Object in verschachtelten Arrays -> (codeConcussion, 2013)  Regular Expressions in Powershell -> (sdwheeler, ArieHein, sethvs, Blake-Madden, 2025) |

|  |
| --- |
| Hilfestellungen: |
| Regex-Visualisierung für PowerShell -> (gskinner, kein Datum) |

Tabelle 10: Arbeitsjournal Tag 5

Mittwoch, 12.03.2025 – Tag 6

|  |  |  |  |
| --- | --- | --- | --- |
| Geplante Arbeiten: | Soll | Erledigte Arbeiten: | Ist |
| * Implementation   + Abfrage von ILO * Arbeitsjournal | 4h  0.3h | * Implementation   + Abfrage von ILO   + Einbau MAC-Adresse- und Seriennummern-Switch   + Scriptmessages implementiert * Arbeitsjournal * Dokumentieren ILO-Abfrage | 2,5h  0.5h  0.5h  0.3h  0.5h |
| **Überzeit** | **0h** |  | **0h** |
| Arbeitszeit | 4.3h |  | 4.3h |

|  |
| --- |
| Fazit & Reflexion: |
| Heute war ich vor allem damit beschäftigt, die Abfrage von ILO weiter auszuarbeiten, wobei ich die in Inventory abgelegten MAC 1 bis MAC 4 und die IPv4 sowie IPv6-Konfigurationen dazu gehören. Im Laufe des Nachmittags konnte ich einen grossen Teil der ILO-Abfrage bereits abschliessen, bis auf die Abfrage der in den Servern eingebauten Storage – für die vollständige Abfrage derer müssen die Server laufen, was ich im vollen Büro vermeiden wollte und ich am Freitag in Ruhe umsetzen kann.  Im Laufe des Nachmittages ist mir noch aufgefallen, dass ich in der Planung der Konfigurationsdatei das Ausschalten der MAC-Adressen und Seriennummern ausgelassen habe. Diese musste ich daher noch im Nachhinein einfügen.  Da im Skript bestimmte Vorgänge mit Zeitstempeln in die Konsole geschrieben werden sollen, habe ich dafür die bereits implementierte Logfunktion modifiziert. Ansonsten ist nichts nennenswertes passiert, was ich erwähnen könnte.  Ich bin weiterhin vor dem Zeitplan und werde voraussichtlich am Freitag die ILO-Abfrage beenden können und die Speicherung in die CSV- und JSON-Dateien beginnen können. |

|  |
| --- |
| Recherchen: |
| Entfernen von 2 Array-Elementen vom Anfang -> (DarkLite1, 2014) |

Tabelle 11: Arbeitsjorunal Tag 6

Freitag, 14.03.2025 – Tag 7

|  |  |  |  |
| --- | --- | --- | --- |
| Geplante Arbeiten: | Soll | Erledigte Arbeiten: | Ist |
| * Implementieren   + ILO-Abfrage fertigstellen   + Speicherung in JSON-Datei   + Speicherung in CSV-Datei * Arbeitsjournal | 4h  1h  3h  0.3h | * Implementieren   + ILO-Abfrage fertigstellen   + Speicherung in JSON-Datei   + Speicherung in CSV-Datei * Arbeitsjournal | 2,5h  0,5h  5h  0.3 |
| **Überzeit** | **0h** |  | **0h** |
| Arbeitszeit | 8.3h |  | 8.3h |

|  |  |
| --- | --- |
| Probleme: | Problemlösungen: |
| Kompatibilität einiger Funktionen --> Die Abfrage ist je nach ILO-Version nicht möglich. | Abfangen der ILO-Version und bei einer Inkompatibilität wird entweder eine Funktion verwendet, die die gleichen Werte auf anderem Weg holt, oder wenn nicht möglich die Nachricht, dass es nicht Kompatibel ist, gespeichert wird. |
| Funktion zur Abfrage der Storage exisitert für ILO 6, sie gibt aber (während der Server läuft) einen Fehler zurück, da es einen Fehler mit dem Typ gibt. | Liegt vermutlich an einer HPE-seitigen Inkompatibilität mit meiner PS-Version (7.5). Die Abfrage wird trotzdem durchgeführt – sie sollte in zukünftigen Versionen gefixt werden. |

|  |  |
| --- | --- |
| Fragen Kandidat Yannick Wernle: | Antworten Fachkraft Helge Brands |
| In der Aufgabe steht, dass die Speichern der Seriennummern und MACAdressen in der CSV-Datei abschaltbar sein muss. Wofür ist das gedacht? | In der Zukunft soll irgendwann Inventory die generierte JSON-Datei einlesen können – die CSV-Dateien sind nur für die manuelle Seite gedacht – später soll das Skript mal als Dienst laufen und dann braucht es die CSV-Dateien nicht mehr. |

|  |
| --- |
| Fazit & Reflexion: |
| Heute morgen war ich vor allem damit beschäftigt, die ILO-Abfrage noch fertigzustellen – für die Abfrage der Storage mussten die Server laufen, weswegen ich aufgrund der Startzeiten der Server teils lange brauchte. Ich bin ausserdem auf ein Problem der Kompatibilität bei der Storageabfrage auf ILO6 gestossen, dass oben beschrieben ist. Die Lösung ist zwar weniger schön, aber da dieser Fehler nicht von meiner Seite gelöst werden kann, sind mir die Hände gebunden. Ich habe hierzu auch Chatgpt gefragt, wie der Fehler sich beheben liesse und er kommt auf eine Inkompatibilität zwischen ILO 6 und der Modulversion.  Den Rest des Tages habe ich damit verbracht, die JSON- und CSV-Dateien abzuspeichern: Die generierung der JSON-Datei ging sehr schnell, da ich die bereinigte Abfrage von ILO mit einem einfachen Befehl konvertieren konnte und ich das bereits bei der Konfiguration benutze.  Deutlich länger dauerte es, die CSV-Dateien zu erstellen: CSV-Dateien sind mehr oder weniger Tabellen, in denen sich die verschachtelte Natur der Abfrage kaum gescheit abbilden lässt. Deswegen brauchte ich etwas länger, unter anderem auch deswegen, weil ich mich dazu entschlossen hatte, für die genauen MAC-Addressen und Seriennummern je eigene Dateien zu erstellen – hierfür musste ich wieder im Hintergrund filtern, was relativ zeitaufwändig war. Ich bin aber trotzdem noch im Zeitplan und werde voraussichtlich mit der eigentlichen Aufgabe bald fertig, es geht am Montag und die nächste Woche vor allem darum, die Hilfestellungen einzufügen und Fehler auszumerzen. |

|  |
| --- |
| Recherchen: |
| Fehler: […] violates the constraint of type ‘T’ --> (pl5, 2024)  Überprüfung ob ein Pfad in einem Directory endet --> (Scripto, 2014) |

|  |
| --- |
| KI-Prompts: |
| Wie liesse sich dieser Fehler von HPEILOCmdlets (vers. 4.4.0.0), der Server hat ILO6 beheben? |

Tabelle 12: Arbeitsjournal Tag 7

Montag, 17.03.2025 – Tag 8

|  |  |  |  |
| --- | --- | --- | --- |
| Geplante Arbeiten: | Soll | Erledigte Arbeiten: | Ist |
| * Realisieren Abschluss Dateien * Dokumentierung Dateien * Hilfestellungen | 3,5h  1h  3,5h | * Informieren: * Dokumentation: * Zeitplan erstellen * Arbeiten für nächsten Tag geplant |  |
| **Überzeit** | **0h** |  | **0h** |
| Arbeitszeit |  |  |  |

|  |  |
| --- | --- |
| Probleme: | Problemlösungen: |
|  |  |

|  |  |
| --- | --- |
| Fragen Kandidat Yannick Wernle: | Antworten Fachkraft Helge Brands |
| (Kurze Diskusion mit Helge) Sollen Fehler in Konfigurationsdateien und an sonstigen Orten so weit als möglich korrigiert werden (im Bezug auf Fehler der Konfiguration)? | Die Fehler sollen vor allem bei der Generierung der Konfiguration abgefangen werden. Im restlichen Teil des Programmes soll der User darauf hingewiesen werden (mit Hilfestellungen), aber fixen muss er sie selbst. |

|  |
| --- |
| Fazit & Reflexion: |
| Heute stand vor allem der Abschluss der Ausgabedateien im Vordergrund, speziell die CSV-Dateien – ich konnte am Freitag schon einen grossen Teil abschliessen, für heute blieb aber die letzte CSV-Datei übrig, welche die Seriennummern ausgibt. Probleme gab es vor allem bei der Konvertierung, wenn ein Feld leer war – ich musste diese daher mühsam alle abfangen.  Ich musste ausserdem eine Funktion erstellen, welche die Keys unter allen abgefragten Servern standardisiert – wenn dem nicht so ist, wird nämlich einfach nur die Felder angezeigt, deren Key bei allen vorhanden ist – das ist nicht schön, weil so einfach Daten verloren gehen. Ausserdem habe ich bei den Seriennummern die Tabelle erweitert, sodass nicht nur die Seriennummern angezeigt werden, sondern in einer separaten Tabelle unten auch Details dazu, damit man weiss, was zu was gehört.  Helge und ich haben kurz vor dem Mittag noch darüber geredet, wie der Stand ist und haben uns dann entschlossen, die IPA testweise am Mittwochnachmittag auf den SwissFEL-Maschinen zu testen.  Am Nachmittag habe ich mich vor allem mit den Hilfestellungen befasst und auch damit begonnen, die Testfälle vom Anfang des Projektes auszubauen: Da ich nun ein konkretes Programm habe, kann ich die bereits vorhandenen korrigieren und noch weitere, vorher nicht berücksichtigte, Fälle hinzufügen, nach denen ich testen kann.   * Kurz Zusammenfassung, was habe ich gemacht. * Was gelang mir gut, was weniger? Worin liegen die Gründe? * Sind alle Ziele erreicht worden? Weshalb nicht? * Warum bin ich mit meiner Leistung zufrieden / unzufrieden? * Was habe ich gelernt? * Was kann ich besser / anders machen? |

|  |
| --- |
| Recherchen: |
|  |

Tabelle 13: Arbeitsjournal Tag 8

|  |
| --- |
| Hilfestellungen: |
|  |

|  |
| --- |
| KI-Prompts: |
|  |

[WOCHENTAG], [DATUM] – Tag [TAGESNUMMER]

|  |  |  |  |
| --- | --- | --- | --- |
| Geplante Arbeiten: | Soll | Erledigte Arbeiten: | Ist |
| * Informieren: * Dokumentation: * Arbeitsjournal * Dokumentstruktur * Organigramm * Aufgabenstellung * Zeitplan erstellen |  | * Informieren: * Dokumentation: * Zeitplan erstellen * Arbeiten für nächsten Tag geplant |  |
| **Überzeit** | **0h** |  | **0h** |
| Arbeitszeit |  |  |  |

|  |  |
| --- | --- |
| Probleme: | Problemlösungen: |
|  |  |

|  |  |
| --- | --- |
| Fragen Kandidat Yannick Wernle: | Antworten Fachkraft Helge Brands |
|  |  |

|  |
| --- |
| Fazit & Reflexion: |
| * Kurz Zusammenfassung, was habe ich gemacht. * Was gelang mir gut, was weniger? Worin liegen die Gründe? * Sind alle Ziele erreicht worden? Weshalb nicht? * Warum bin ich mit meiner Leistung zufrieden / unzufrieden? * Was habe ich gelernt? * Was kann ich besser / anders machen? |

|  |
| --- |
| Recherchen: |
|  |

|  |
| --- |
| Hilfestellungen: |
|  |

|  |
| --- |
| KI-Prompts: |
|  |

[WOCHENTAG], [DATUM] – Tag [TAGESNUMMER]

|  |  |  |  |
| --- | --- | --- | --- |
| Geplante Arbeiten: | Soll | Erledigte Arbeiten: | Ist |
| * Informieren: * Dokumentation: * Arbeitsjournal * Dokumentstruktur * Organigramm * Aufgabenstellung * Zeitplan erstellen |  | * Informieren: * Dokumentation: * Zeitplan erstellen * Arbeiten für nächsten Tag geplant |  |
| **Überzeit** | **0h** |  | **0h** |
| Arbeitszeit |  |  |  |

|  |  |
| --- | --- |
| Probleme: | Problemlösungen: |
|  |  |

|  |  |
| --- | --- |
| Fragen Kandidat Yannick Wernle: | Antworten Fachkraft Helge Brands |
|  |  |

|  |
| --- |
| Fazit & Reflexion: |
| * Kurz Zusammenfassung, was habe ich gemacht. * Was gelang mir gut, was weniger? Worin liegen die Gründe? * Sind alle Ziele erreicht worden? Weshalb nicht? * Warum bin ich mit meiner Leistung zufrieden / unzufrieden? * Was habe ich gelernt? * Was kann ich besser / anders machen? |

|  |
| --- |
| Recherchen: |
|  |

|  |
| --- |
| Hilfestellungen: |
|  |

|  |
| --- |
| KI-Prompts: |
|  |

[WOCHENTAG], [DATUM] – Tag [TAGESNUMMER]

|  |  |  |  |
| --- | --- | --- | --- |
| Geplante Arbeiten: | Soll | Erledigte Arbeiten: | Ist |
| * Informieren: * Dokumentation: * Arbeitsjournal * Dokumentstruktur * Organigramm * Aufgabenstellung * Zeitplan erstellen |  | * Informieren: * Dokumentation: * Zeitplan erstellen * Arbeiten für nächsten Tag geplant |  |
| **Überzeit** | **0h** |  | **0h** |
| Arbeitszeit |  |  |  |

|  |  |
| --- | --- |
| Probleme: | Problemlösungen: |
|  |  |

|  |  |
| --- | --- |
| Fragen Kandidat Yannick Wernle: | Antworten Fachkraft Helge Brands |
|  |  |

|  |
| --- |
| Fazit & Reflexion: |
| * Kurz Zusammenfassung, was habe ich gemacht. * Was gelang mir gut, was weniger? Worin liegen die Gründe? * Sind alle Ziele erreicht worden? Weshalb nicht? * Warum bin ich mit meiner Leistung zufrieden / unzufrieden? * Was habe ich gelernt? * Was kann ich besser / anders machen? |

|  |
| --- |
| Recherchen: |
|  |

|  |
| --- |
| Hilfestellungen: |
|  |

|  |
| --- |
| KI-Prompts: |
|  |

[WOCHENTAG], [DATUM] – Tag [TAGESNUMMER]

|  |  |  |  |
| --- | --- | --- | --- |
| Geplante Arbeiten: | Soll | Erledigte Arbeiten: | Ist |
| * Informieren: * Dokumentation: * Arbeitsjournal * Dokumentstruktur * Organigramm * Aufgabenstellung * Zeitplan erstellen |  | * Informieren: * Dokumentation: * Zeitplan erstellen * Arbeiten für nächsten Tag geplant |  |
| **Überzeit** | **0h** |  | **0h** |
| Arbeitszeit |  |  |  |

|  |  |
| --- | --- |
| Probleme: | Problemlösungen: |
|  |  |

|  |  |
| --- | --- |
| Fragen Kandidat Yannick Wernle: | Antworten Fachkraft Helge Brands |
|  |  |

|  |
| --- |
| Fazit & Reflexion: |
| * Kurz Zusammenfassung, was habe ich gemacht. * Was gelang mir gut, was weniger? Worin liegen die Gründe? * Sind alle Ziele erreicht worden? Weshalb nicht? * Warum bin ich mit meiner Leistung zufrieden / unzufrieden? * Was habe ich gelernt? * Was kann ich besser / anders machen? |

|  |
| --- |
| Recherchen: |
|  |

|  |
| --- |
| Hilfestellungen: |
|  |

|  |
| --- |
| KI-Prompts: |
|  |

1. Teil 2 – Projektdokumentation
   1. Kurzfassung des IPA Berichts
      1. Ausgangslage

Was ist die Ausgangssituation?

Was ist das Problem?

Was ist das Ziel?

* + 1. Vorgehen

Wie wurde das Problem gelöst?

Welche Herausforderungen gab es?

* + 1. Ergebnis

Was ist das Ergebnis?

Gibt es Verbesserungspotential beim Programm?

Erfüllt das Produkt die Anforderungen?

* 1. Informieren
     1. Projektumfeld



Abbildung 4: Projektumfeld

Das Umfeld des Projekts ist in der Abbildung dargestellt – grundsätzlich läuft es so ab, dass sich die Person, die das Skript ausführen will, per Remotedesktopverbindung auf einen Server einloggt. Diese Server befinden sich in einem gesonderten, gesicherten Netz ohne Internetzugang. Da das Skript lokal ausgeführt wird, befindet es sich ebenfalls in diesem Netz – um das zu ermöglichen muss die HPEiLOCmdlets-Bibliothek lokal vorhanden sein.

Das PowerShell-Skript muss aber gleichzeitig vom Inventory, welches sich im Officenetz (mit Internetzugang) befindet, Daten abfragen – namentlich die Hostnamen fürs iLO. Inventory stellt diese per API bereit.

Als Zielgruppe im weitesten Sinn kann man meinen Fachvorgesetzten nennen – er braucht das Skript, um Inventurdaten zu erfassen. Die einzige andere Person, die noch zur Benützung in Frage käme ist Kurt Bitterli, welcher die Gruppe leitet, die Inventory grossflächig zur Inventarisierung nutzt. Im Vorfeld der IPA wurde aber entschieden, Kurt nicht in die IPA zu integrieren, da er einerseits genug anderes zu tun hat, und andererseits seine Ansprüche & Vorschläge den Zeitrahmen der IPA sprengen würden.

* + 1. Systemgrenzen & Schnittstellen

Die Grenzen des Systems bildet vor allem die Abfrage vom Inventory und gleichzeitig von den Servern – es ist als normaler User am PSI nicht möglich, sich gleichzeitig mit zwei verschiedenen Netzen zu verbinden. Das bedeutet, die beiden Teile (Abfrage Inventory und Server) müssen separat voneinander funktionsfähig sein.

Ausserdem ist das Einzige, was als Resultat des Skripts übrig bleiben soll, Dateien mit den ausgelesenen Daten – sie werden weder per Mail verschickt, noch sonst irgendwie bereitgestellt. Sprich, es soll kein Update im Inventory durch das Skript ausgelöst werden.

Schnittstellen gibt es vom *normalen* PowerShell zur *HPEiLOCmdlets*-Bibliothek, welche die HP-Server per iLO-Technologie abfragt. Weiter gibt es natürlich auch die Schnittstelle von PowerShell zur WebAPI von Inventory – diese kommunizieren über HTTPS miteinander.

Eine sehr wichtige Grenze des Systems ist die Art an Systemen, welche Abgefragt werden können – iLO ist eine Technologie, welche nur von HPE-Servern bereitgestellt werden. Das bedeutet, dass bspw. Netzwerkkarten oder andere im Server eingebaute Teile, die nicht von HPE verwendet oder verkauft werden, nicht abfragbar sind mit einem solchen Skript. Ausserdem werden selbstgebaute Teile in der Ausgabe von iLO nicht mit dem gleichen Informationsumfang beschrieben, wie ihre eigenen (Nur *PCIe-Card* statt konkreter Bezeichnung).

* + 1. Anforderungsanalyse

In der Anforderungsanalyse werden alle Anforderungen an das Produkt dargestellt und einzeln erfasst. Sie basieren auf der detaillierten Aufgabenstellung und sollen so konkret sein, dass sie sich jeweils mit nur einem Feature befassen.

Es wird zwischen Funktionalen Anforderungen (ANF-XX), qualitativen Anforderungen (ANF-Q-XX) und Rahmenbedingungen (ANF-R-XX) unterschieden. Für die funktionalen Anforderungen gibt es aufgrund der Anzahl eine eigene Tabelle.

Funktionale Anforderungen

|  |  |
| --- | --- |
| Anforderung | Beschreibung |
| ANF-01 | Es wird auf Anforderung des Users eine Hilfeseite angezeigt |
| ANF-02 | Es wird bei Fehlerhaften Parametern dem User eine Hilfeseite angezeigt und auf den Fehler hingewiesen. |
| ANF-03 | Es wird bei Fehlern in den Konfigurationsdateien eine Hilfeseite angezeigt und auf den Pfad und das Problem hingewiesen. |
| ANF-04 | Es wird bei der Initialisierung des Programmes die Version der Bibliothek verglichen und auf Unterschiede hingewiesen. |
| ANF-05 | Das Programm soll per Konfigurationsdatei konfiguriert werden können. |
| ANF-06 | Es soll eine leere Konfigurationsdatei generiert werden können. |
| ANF-07 | Es soll eine Konfigurationsdatei generiert werden können, welche Beispieldaten beinhaltet. |
| ANF-08 | Das Programm soll ebenfalls nur per Kommandozeile gestartet werden können. |
| ANF-09 | Es soll eine Logfunktion geben, deren Aktivierung, Level und Zielort konfiguriert werden können. |
| ANF-10 | Es soll Parameter geben, welche mittels Suchstring Inventory abfragen und diese zur iLO-Abfrage verwenden. |
| ANF-11 | Es soll Parameter geben, welche anstatt von Inventory eine Liste an Hostnamen zur iLO-Abfrage verwenden |
| ANF-12 | Das Skript soll so aufgebaut werden, dass auch ohne Inventory das Skript zur Abfrage verwendet werden kann. |
| ANF-13 | Es soll einen Parameter geben, welcher den Pfad zur Konfiguration enthält. |
| ANF-14 | Es soll einen Parameter geben, welcher den Pfad zum Report enthält. |
| ANF-15 | Es soll einen Parameter geben, welcher einen Pfad enthält, wo nach den benötigten Dateien gesucht wird. |
| ANF-16 | Es soll ein Verbindungstest/Pingtest durchgeführt werden, um zu testen, ob der gewünschte Server erreichbar ist. |
| ANF-17 | Es soll einen Parameter geben, welcher den Pingtest ausschalten lässt. |
| ANF-18 | Es sollen vom Server die MAC-Adressen und Seriennummern abgefragt werden können. |
| ANF-19 | Das Ergebnis der Abfrage soll in einem JSON-File gespeichert werden. |
| ANF-20 | Das Ergebnis der Abfrage soll in einem CSV-File gespeichert werden, welches den Hostnamen, die Seriennummer und die Anzahl Netzwerkinterfaces beinhält. |
| ANF-21 | Das Speichern von MAC-Adressen und Seriennummern soll jeweils per Parameter ausschaltbar sein. |
| ANF-22 | Die Ausgaben in der Konsole sollen Zeitstempel beinhalten. |
| ANF-23 | Die Konsole soll den Verbindungsaufbau und Stand der Abfrage auf dem Bildschirm zeigen. |
| ANF-24 | Es sollen Fehler und das Verhalten während der Laufzeit in einem Logfile gespeichert werden. |
| ANF-25 | Das Skript mit den Servern DL380 Gen8/Gen9/Gen10/Gen11& DL20 Gen10 getestet werden. |

Nicht Funktionale Anforderungen

|  |  |
| --- | --- |
| Anforderung | Beschreibung |
| ANF-Q-01 | Die Datenstruktur des Reports soll ähnlich zum Inventory sein. |
| ANF-R-01 | Das Skript wird mittels PowerShell entwickelt & verwendet die Bibliothek *HPEiLOCmdlets*. |
| ANF-R-02 | Das Skript soll per Abfragen Daten von den spezifizierten Servern auslesen und abspeichern. |

* + 1. Programablaufplan

Aufgrund der sehr kleinen Zielgruppe des Skriptes und als Resultat davon, dass sich keine verschiedenen Rollen aus der Aufgabenstellung ableiten lassen, ist es nicht zielführend hier mit Personas, Use Cases oder User Stories zu arbeiten, da diese mehrere Rollen voraussetzen – sie trotzdem zu machen wäre weder effizient noch notwendig, da die Wünsche der einzigen Rolle schon in der Anforderungsanalyse persönlich abgebildet sind.

Weitaus zielführender ist hingegen das Erstellen eines Programmablaufplans. Durch ihn können die relevanten Abläufe und Schnittstellen besser dokumentiert werden und sind auch für das Verständnis des gesamten Skripts als Aussenstehender wichtig.



Abbildung 5: Programmablaufplan (simpel)

Dieser PAP ist lediglich als grober Ablauf des Programms zu verstehen – genaue Abfragen und Optionen, wie sie z.B. bei der Aktion *Generiere Konfigurationsdatei* sind bewusst weggelassen worden, sodass dieser Plan eine Übersicht aller geplanter und vorgegebenen Funktionen darstellt. In der nächsten Phase, der Planung, gibt es einen weiteren, weitaus detaillierteren Ablaufplan, anhand dessen das Skript im Endeffekt implementiert wird.

* 1. Planen
     1. Testkonzept

**Testsystem**

Die Tests, welche anhand der Aufgabenstellung durchgeführt werden müssen, werden in Umgebungen getestet, welche der tatsächlichen Anwendung auch so vorkommen könnten. Dabei gibt es zwei unterschiedliche Fälle: Die Server hängen im Officenetzwerk oder sie hängen im Maschinennetz.

Getestet werden soll, ob das Skript wie geplant funktioniert und reagiert, vor allem im Bezug auf die Erreichbarkeit des Inventory, sowie die komplette Funktionstüchtigkeit. Es ist in PowerShell zwar möglich, Unittests mittels Pester zu implementieren, jedoch ist das nicht Bestandteil der Aufgabenstellung – gefordert sind nur Laufzeittests.

**Voraussetzungen**

Der Laptop des Kandidaten muss angeschaltet sein und sich in einem Netzwerk befinden, in dem sich mind. Einer der 5 Server befindet. Ausserdem muss die richtige PowerShell-Version (mind. v.7) installiert sein. Ausserdem muss der iLO-Server, der abgefragt werden soll, am Strom hängen – sonst kommen keine Daten zurück. Sie müssen, wenn möglich, am laufen sein, damit die volle Informationsbreite getestet werden kann (wenn der Server nicht läuft und nur das iLO geben manche der Befehle nicht alle Daten zurück.

**Testumgebung**

Jegliche Tests wurden auf dem persönlichen PSI-Laptop des Kandidaten durchgeführt. Dieser hat die folgenden Spezifikationen:

|  |  |
| --- | --- |
| PSI Laptop | PC14734 / HP EliteBook 840 G7 Notebook PC |
| Betriebssystem | Windows 11 Enterprise (23H2) – Build 22631.4317 |
| Powershell | Version 7.5.0 |
| VS-Code | Version 1.97.0 |
| VS-Code PowerShell Extension | 2025.0.0 |

**Testmittel und Methoden**

Es werden **Laufzeittests** durchgeführt – das heisst, sie werden getestet, während das Programm effektiv so läuft wie es nach der Auslieferung auch laufen würde.

**User Tests**

Die Testfälle werden in Form von User Tests getestet, sprich sie werden von Hand und von einer echten Person getestet. Es werden dafür Testfälle geschrieben, welche die genaue Vorgehensweise beim jeweiligen Test angeben. Zu jeder Testfallspezifikation gibt es einen Eintrag im Testprotokoll. Im Testbericht ist das Ergebnis und falls vorhanden, notwändige Verbesserungen festgehalten.

**Nicht getestete Aspekte**

Als Folge der Durchführung als User Tests wird nur das Zusammenspiel der Funktionen als Endprodukt getestet, nicht aber alle Funktionen, die es im Skript gibt. Diese liessen sich in Zukunft mit Unittests abdecken.

* + 1. Testfälle

|  |  |  |  |
| --- | --- | --- | --- |
| Testfall: | TF-01 | Anforderung: | ANF-01 |
| Testart | Funktionaler Test | | |
| Beschrieb | Benutzer kann Hilfe anzeigen lassen | | |
| Voraussetzungen | Skript ist gestartet | | |
| Testschritte | 1. Get-Help Get-DataFromILO | | |
| Erwartetes Ergebnis | Die Hilfeseite wird angezeigt. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Testfall: | TF-02 | Anforderung: | ANF-02 |
| Testart | Funktionaler Test | | |
| Beschrieb | Benutzer kann Hilfe anzeigen lassen | | |
| Voraussetzungen | Skript ist gestartet | | |
| Testschritte | 1. Get-DataFromILO -Path 0 | | |
| Erwartetes Ergebnis | Die Hilfeseite über Parameter wird angezeigt und auf den Fehler hingewiesen, dass der Pfad nicht richtig ist. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Testfall: | TF-03 | Anforderung: | ANF-03, ANF-05 |
| Testart | Funktionaler Test | | |
| Beschrieb | Benutzer kann Hilfe anzeigen lassen | | |
| Voraussetzungen | Skript ist gestartet & Konfiguration vorhanden, Loglevel ist keine Zahl | | |
| Testschritte | 1. Get-DataFromILO -Path C:\PathToConfig | | |
| Erwartetes Ergebnis | Die Hilfeseite über Parameter wird angezeigt und auf den Fehler hingewiesen, dass der Loglevel keine Zahl sein kann. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Testfall: | TF-04 | Anforderung: | ANF-06 |
| Testart | Funktionaler Test | | |
| Beschrieb | Benutzer kann eine Leere Konfiguration generieren | | |
| Voraussetzungen | Skript ist gestartet, keine Konfigurationsdatei hinterlegt. | | |
| Testschritte | 1. Get-DataFromILO 2. Aus Optionen “Generate empty config» wählen | | |
| Erwartetes Ergebnis | Leere Konfigurationsdatei «config.json» wird erstellt. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Testfall: | TF-05 | Anforderung: | ANF-07 |
| Testart | Funktionaler Test | | |
| Beschrieb | Benutzer kann eine Konfiguration generieren, welche Beispieldaten beinhaltet | | |
| Voraussetzungen | Skript ist gestartet, keine Konfigurationsdatei hinterlegt. | | |
| Testschritte | 1. Get-DataFromILO 2. Aus Optionen “Generate dummy-config» wählen | | |
| Erwartetes Ergebnis | Konfigurationsdatei «config.json» wird erstellt und enthält Beispieldaten. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Testfall: | TF-06 | Anforderung: | ANF-07 |
| Testart | Funktionaler Test | | |
| Beschrieb | Benutzer kann komplettes Skript ausführen mit Verbindung zum Inventory. | | |
| Voraussetzungen | Skript ist gestartet, eine richtige Konfigurationsdatei ist hinterlegt, Inventory ist erreichbar. | | |
| Testschritte | 1. Option in Konfig «Suchstring» und «Field\_Inventory» sind auf «sf-sioc-cs» und «Hostname\_Mngt» gesetzt. 2. Get-DataFromILO -Path | | |
| Erwartetes Ergebnis | Es werden die Servernamen aus Inventory abgefragt, in ein .json-File zwischengespeichert. Danach wird anhand der Hostnamen ein Pingtest durchgeführt --> als Resultat werden ein report.json und report.csv generiert. Das Resultat enthählt MAC-Adressen und Seriennummern. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Testfall: | TF-07 | Anforderung: | ANF-07 |
| Testart | Funktionaler Test | | |
| Beschrieb | Benutzer kann komplettes Skript ausführen ohne Verbindung zum Inventory. | | |
| Voraussetzungen | Skript ist gestartet, eine richtige Konfigurationsdatei ist hinterlegt, in der Konfiguration ist ein Feld «QueryServerPath» mit einem Pfad zu einem Array an Servernamen. Inventory ist nicht erreichbar. | | |
| Testschritte | 1. Option in Konfig «Suchstring» und «Field\_Inventory» sind auf «sf-sioc-cs» und «Hostname\_Mngt» gesetzt. 2. Get-DataFromILO -Path | | |
| Erwartetes Ergebnis | Es werden die Servernamen aus Inventory abgefragt – es kommt keine Antowrt, Es wird stattdessen der «QueryServerPath» nach Servern abgefragt. Danach wird anhand der Hostnamen ein Pingtest durchgeführt --> als Resultat werden ein report.json und report.csv generiert. Das Resultat enthählt MAC-Adressen und Seriennummern. | | |
| Testfall: | TF-08 | Anforderung: | ANF-07 |
| Testart | Funktionaler Test | | |
| Beschrieb | Benutzer kann komplettes Skript ausführen ohne Verbindung zum Inventory, nur mit der Komandozeile. | | |
| Voraussetzungen | Skript ist gestartet, keine Konfigurationsdatei ist hinterlegt. | | |
| Testschritte | 1. Get-DataFromILO -Servers @{«rmgfa-sioc-cs-dev», «rmgfa-sioc-cs-de4”} | | |
| Erwartetes Ergebnis | Es werden die Servernamen aus Inventory abgefragt – es kommt keine Antowrt, Es wird stattdessen der «QueryServerPath» nach Servern abgefragt. Danach wird anhand der Hostnamen ein Pingtest durchgeführt --> als Resultat werden ein report.json und report.csv generiert. Das Resultat enthählt MAC-Adressen und Seriennummern. | | |

* + 1. Verwendete Technologien

Bei der Implementation des Skripts werden die folgenden Technologien, Bibliotheken und Programme verwendet:

* PowerShell v.7.5.0
* HPEiLOCmdlets v.4.4.0.0
* Visual Studio Code v.1.97.0
* VS-Code PowerShell Extension (2025.0.0)
* Git-Extensions

In der Dokumentation und den Darstellungen darin kamen die folgenden Programme zur Anwendung:

* Microsoft Onedrive.com (Backup Dokumentation)
* Draw.io (Darstellungen und Diagramme)
* Microsoft Word (Dokumentation)
* Microsoft Excel (Zeitplan und Entscheidungsmatrix)
* PAP-Designer
  + 1. Konfigurationsdatei(en)

Für die Konfigurationsdatei macht es Sinn, sie in verschiedene kleinere Konfigurationsdateien aufzuteilen – dies ist vor allem beim Login wichtig, denn wenn man eine Konfigurationsdatei verschickt, will man nicht unbedingt, dass das eigene Password mitsamt Benutzername verschickt wird – es besteht immer das Risiko, dass es abgefangen wird und ein gestohlenes Passwort kann viel Schaden anrichten.

Als Resultat davon soll das Login in einer separaten Konfiguration gespeichert werden – so kann man die *normale* Konfiguration ohne bedenken weitersenden.

Ein weiterer Teil, welcher in ein separates File gehört, ist die Liste an Servern, die abgefragt werden soll – ist das Inventory erreichbar, so wird diese Liste ja automatisch aufgefüllt, aber wenn es nicht erreichbar ist nicht. So kann man die in der Aufgabenstellung geforderte Zwischenspeicherung einfach machen und es trotzdem sinnvoll und schnell änderbar machen.

Als Dateiformat der Konfiguration bietet sich JSON sehr gut an – es ist weit verbreitet und unterstützt. Ausserdem ist es gut lesbar und einfach änderbar.

**Haupt-Konfigration (*config.json*)**

|  |  |
| --- | --- |
| Bezeichnung | Wert |
| searchForFilesAt | C:\Path\To\Somewhere |
| configPath | C:\Path\To\Somewhere |
| loginConfigPath | C:\Path\To\Somewhere |
| reportPath | C:\Path\To\Somwhere |
| logPath | C:\Path\To\Somwhere |
| serverPath | C:\Path\To\Somwhere |
| logLevel | 0 |
| loggingActivated | True |
| searchStringInventory | sf-sioc-cs |
| doNotSearchInventory | false |
| remoteMgmntField | Hostname Mgnt |
| deactivateCertificateValidationILO | false |

Die Bezeichnungen sind weitestgehend selbsterklärend – sie sind jeweils der Pfad wo entweder eine Datei abgelegt ist oder wo Dateien abgelegt werden sollen.

Besonders zu erwähnen sind die Felder *searchStringInventory* und *RemoteMgmntField*. Ersteres gibt an, mit welchem String gesucht werden soll und letzteres ist das Feld in Inventory, welches schlussendlich zurückgegeben werden soll.

Als weiteres wichtiges Feld ist *serverPath* – dort werden die server von Inventory zwischengespeichert, resp. dort muss man die Hostnamen der Server ablegen, um sie via ILO abzufragen.

Falls das der Benützer wünscht, kann er mittels *deactivateCertificateValidationILO* die Validation der Zertifikate deaktivieren, sodass er kein gültiges Zertifikat braucht, um mit iLO zu kommunizieren.

**Login-Konfiguration (*login.json*)**

|  |  |
| --- | --- |
| Bezeichnung | Wert |
| Username | SomeUsername |
| Password | SomePassword |

**Server-Konfiguration (*servers.json*)**

Die Serverkonfiguration unterscheidet sich von den anderen beiden Konfigurationen darin, das es eigentlich nur ein Array an Hostnamen ist und keine weiteren Eigenschaften enthält.

[

«server-hostname-1»,

«server-hostname-2»,

«server-hostname-3»,

«server-hostname-4»,

«server-hostname-5»,

]

Codesnippet 1: Inhalt der Server-Konfiguration

* + 1. Parameter

PowerShell ermöglicht viele verschiedene Arten, Parameter zu verwenden. Die Parameter sind essentiell eigentlich die der Konfigurationsdateien.

Im Prinzip geht es darum herauszufinden, welche Parameter zwingend nötig sind, um das Skript per Kommandozeile und ohne Konfiguration starten zu können: Dies sind eigentlich nur der Benutzername und Passwort für ILO. Die anderen Parameter sind alles solche, die für das Skript genutzt werden können, aber nicht immer notwendig sind.

Es gibt insgesamt vier Fälle, um das Skript per Kommadozeile zu starten:

via Verbindung auf Inventory (d.h. Angabe von *searchStringInventory*, *Username****,*** *Password*)

via direkter Angabe einer Liste an Servern (d.h. Angabe eines Arrays zur Suche, *Username*, *Password*)

via Angabe einer Liste an Servern als Datei (d.h. *serverPath*, *Username*, *Password*)

via Angabe einer Konfiguration (d.h. *configPath*, *Username*, *Password*)

Aus diesen Überlegungen lässt sich diese Parameterkonfiguration ableiten:

[CmdletBinding(PositionalBinding = $False)]

param(

    [Parameter(Mandatory = $true,

        ParameterSetName = "Config")]

    [string]

    $ConfigPath,

    [Parameter()]

    [string]

    $LoginConfigPath,

    [Parameter()]

    [string]

    $ReportPath,

    [Parameter()]

    [string]

    $LogPath,

    [Parameter(Mandatory = $true,

        ParameterSetName = "ServerPath")]

    [string]

    $ServerPath,

    [Parameter(Mandatory = $true,

        ParameterSetName = "ServerArray")]

    [array]

    $server,

    [Parameter()]

    [int]

    $LogLevel,

    [Parameter()]

    [switch]

    $LoggingActivated,

    [Parameter(Mandatory = $true,

        ParameterSetName = "Inventory")]

    [string]

    $SearchStringInventory,

    [Parameter()]

    [switch]

    $DoNotSearchInventory,

    [Parameter()]

    [string]

    $RemoteMgmntField,

    [Parameter()]

    [switch]

    $DeactivateCertificateValidationILO,

    [Parameter(Mandatory = $true,

        ParameterSetName = "Config")]

    [Parameter(Mandatory = $true,

        ParameterSetName = "ServerPath")]

    [string]

    $Username,

    [Parameter(Mandatory = $true,

        ParameterSetName = "Config")]

    [Parameter(Mandatory = $true,

        ParameterSetName = "ServerPath")]

    [Parameter(Mandatory = $true,

        ParameterSetName = "ServerArray")]

    [Parameter(Mandatory = $true,

        ParameterSetName = "Inventory")]

    [string]

    $Password

)

Codesnippet 2: Geplante Parameter

Dabei wird anhand der oben beschriebenen vier Fälle je ein Parameterset erstellt – das bedeutet, es kann nur jeweils ein einziges davon gleichzeitig verwendet werden. Man kann also nicht z.B. ein ServerArray angeben & ein Server-Path, das wird damit verhindert.

Die Datentypen sind ebenfalls so gut als möglich den Datentypen angepasst, welche später vermutlich verwendet werden.

Die Nutzung des **Positionalbinding**-Argument ganz am Anfang der Parameterliste führt dazu, dass die Parameter nicht positionell gesetzt werden können, sondern nur durch genaue Angabe des Namens – dadurch wird der Nutzer dazu gebracht, darüber nachzudenken, was er da überhaupt eingibt und macht das es ausserdem etwas einfacher zu dokumentieren.

* + 1. Programmablaufplan detailliert

Aus dem simplen Programmablaufplan lässt sich zwar ableiten, wie die grobe Funktionalität des Skriptes am Ende aussehen soll, aber als Template für die Implementation ist es nicht geeignet – es ist zu wenig detailliert und zu verallgemeinert.

Um es als grobes strukturelles Template verwenden zu können braucht es mehr Details. Anhand des bisherigen Templates und der Anforderungsanalyse ergibt sich das folgende PAP:

Ein Bild, das Text, Screenshot, Karte Menü enthält.

AI-generated content may be incorrect.

Abbildung 6: Programmablaufplan Skript detailliert

* 1. Entscheiden

Aus der Aufgabenstellung gibt es eigentlich nur eine vorgeschriebene Entscheidung: Welche Parameter sind nötig zum Starten des Skripts. Mittels der Parametersets in PowerShell lassen sich mehrere verschiedene Optionen gleichzeitig implementieren – so lassen sich verschiedene Bedürfnisse gleichzeitig abdecken, ohne dass man auf einen einzeln beschränkt wird.

Es gibt jedoch noch eine andere Entscheidung in Bezug auf die Programmierstruktur – PowerShell lässt die Implementierung von Code sowohl als Modul als auch als einzelnes Skript (oder mehrere Dateien) zu, was natürlich je nach Option einige Dinge stark vereinfacht oder komplizierter macht. Die Aufgabenstellung stellt keine besonderen Anforderungen an die Struktur des Skriptes und der Fachvorgesetzte war damit einverstanden, dass ich das auch so implementieren kann. Anhand der Entscheidungsmatrix soll nun entschieden werden, welche der beiden Optionen die bessere ist.

Entscheidungsmatrix Programmstruktur

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | **Lösungskonzepte** | | | |
| **Kriterien** | | Skript | | Modul | |
| **Anforderungen** | **Gewichtung** | **Max = 5** |  | **Max = 5** |  |
|  |  | **Nutzwert** |  | **Nutzwert** |
| Lesbarkeit in grossen Projekten | **3** | 3 | **9** | 4 | **12** |
| Einfachheit d. Entwicklung | **2** | 5 | **10** | 3 | **6** |
| Unabhängigkeit von Internet | **2** | 4 | **8** | 3 | **6** |
| Einfachheit d. Deployments | **1** | 3 | **3** | 5 | **5** |
| Anpassbarkeit | **1** | 3 | **3** | 5 | **5** |
| Wiederverwendbarkeit Code | **2** | 2 | **4** | 5 | **10** |
| **TOTAL** |  |  | **37** |  | **44** |

Tabelle 14: Entscheidungsmatrix Programmierstruktur

Fazit

Wie man anhand der Entscheidungsmatrix unschwer erkennen kann, ist die Entscheidung relativ knapp ausgefallen und auf dem Modul gelandet. Das ist vor allem der Wiederverwendbarkeit des Codes sowie der Lesbarkeit geschuldet.

Als Skript ist übrigens gemeint, zwar mehrere Dateien zu verwenden, aber diese nicht durch ein Modul zentral zu bündeln. Es ist zwar einfacher in der Entwicklung, aber dafür geht die Modularität und die Wiederverwendbarkeit verloren – da dieses Skript später noch weiterentwickelt werden soll, ist diese aber stark von Vorteil. Ansonsten ist das Modul in den meisten Bereichen nur leicht besser als das Skript. Ein Nachteil des Moduls ist, dass der Kandidat noch nie selbst ein komplettes Modul entwickelt hat, sondern nur zum Testen.

Das Modul bietet ausserdem als entscheidenden Vorteil auch ein Vereinfachtes Deployment mit sich – es kann mit Import-Module von einem zentralen Repository installiert werden, ausserdem können Funktionen gezielt ein- oder ausgeblendet werden. Ein Nachteil ist zugleich aber, das für die Ausführung des Programmes nun nicht über eine einzelne Zeile gestartet werden kann, sondern nun mit 2 Zeilen. Dies ist aber nicht so dramatisch und es kann darüber hinweggesehen werden.

* 1. Realisieren
     1. Grundstruktur

Die Grundstruktur des Programms ist nach den Standards von PowerShell-Modulen aufgebaut:



Abbildung 7: Ordnerstruktur

Der Ordner **ILO-Inventorizer** wird dabei als Wrapper für das Modul verwendet, mit der gleichnamigen Moduldatei (**.psm1**) darin. Ausserdem sind dort weitere Scripts angelegt, die spezifische Funktionen beinhalten, um so den Überblick besser zu behalten.

Die **ILO-Inventorizer.help.txt**-Datei ist dazu da, die Modulhilfe abzufangen (**Get-Help ILO-Inventorizer**) und leitet in ihrem enthaltenen Text in auf die eigentlichen Funktionen und Hauptfunktionen weiter.

Zur Vereinfachten entwicklung gibt es das **launch.json**, welches die Konfiguration fürs Debuggen im VS Code enthält – diese ruft eigentlich nur das **script.ps1** auf, welches jeweils das Modul neu lädt und die Start-Funktion aufruft. Dies ist dazu gedacht, das Skript einfacher debuggen zu können:

Remove-Module ILO-Inventorizer;

Import-Module .\ILO-Inventorizer\ILO-Inventorizer.psm1;

Get-HWInfoFromILO

Codesnippet 3: Inhalt des Startscripts

Es soll am Ende des Skriptes nur eine kleine Anzahl an Funktionen nach Aussen erreichbar sein – wie oben erkennbar ist eine davon die **GetHWInfoFromILO** eine davon. Diese soll den ganzen Dialog und die eigentliche Abfrage triggern.

Da für die Ausführung des Programms eine externe Bibliothek nötig ist, wird sie ebenfalls hier überprüft – falls sie eine andere als erwartet ist, dann wird darauf hingewiesen und davor gewarnt.

Grundstruktur für die Hilfe ist ebenfalls bereits implementiert: Sie wird durch zwei Parameter gelöst, wobei der eine davon help ist und die Parameter **--h**, **--help** und **/?** Abfängt. Da PowerShell zur Kennzeichnung von Parametern **‘-‘** verwendet, muss **-h** als Parameter implementiert werden und kann nicht wie die anderen direkt als String abgefangen werden. Das ist sehr einfach durch den Parametertyp [switch] gelöst – es wird im Hintergrund einfach etwas getrickst, um trotzdem die Hilfe-Funktion aufzurufen.

    ## Check if Help must be displayed

    if ($h -eq $true) { $help = "-h"; }

    if ($help.Length -gt 0) { Show-Help $help; }

Die aufgerufene Show-Help Funktion macht im Grunde auch nichts anderes, als Get-Help Get-HWInfoFromILO -Full aufzurufen. Das bedeutet auch, dass die native Art, Hilfe anzufordern in PowerShell selbstverständlich auch läuft.

* + 1. Generierung der Konfigurationsdatei

Die Generierung der Konfigurationsdateien lässt sich grob auf zwei verschiedene Aspekte aufteilen. Das ist einerseits die Eingabe auf Seiten des Users, der sich entscheiden muss, was für eine Konfiguration er denn überhaupt haben will, was man als *Frontend* bezeichnen könnte.

Die Umsetzung des *Frontends* ist im Endeffekt nichts anderes, als verschachtelte Switch-Case-Statements, welche die 4 verschiedenen Möglichkeiten der Generierung abdecken und generell diesem Muster an Nutzereingaben (Read-Host) und switch-Statements folgen:

$pathToSaveAt = Read-Host -Prompt "Where do you want to save the config at?";

$withInventory = Read-Host -Prompt "Do you want to:`nRead From Inventory [y/N]?"

switch ($withInventory) {

    "y" {

        New-Config -Path $pathToSaveAt -NotEmpty;

        break;

        }

    "N" {

        New-Config -Path $pathToSaveAt -WithoutInventory -NotEmpty;

        break;

        }

}

Codesnippet 4: "Frontend" Generierung der Konfigurationsdateien

Im Hintergrund wird anhand der Eingaben dann ein PowerShell-Objekt erstellt und mit den entsprechenden Werten (Dummy-Werte oder leere Werte) gefüllt und anschliessend mittels ConvertTo-Json & Out-File zuerst in ein JSON-Format konvertiert und anschliessend in einer **config.json**-Datei abgespeichert.

Der Pfad zur Konfigurationsdatei wird schlussendlich in einer Umgebungsvariable namens $ENV:HPEILOCONFIG gespeichert. Über sie läuft die eigentliche Abfrage der Konfigurationsparameter ab.

Wenn man sich nicht sicher ist, welche Datei jetzt gerade als Konfigurationsdatei festgelegt ist, gibt es zwei weitere Funktionen: Mit Get-ConfigPath lässt sich die momentan ausgewählte Konfigurationsdatei anzeigen oder mit Set-ConfigPath unkompliziert ändern.

* + 1. Start via Parameter

Der Start via Parameter unterscheidet sich relativ wenig von der Nutzung mit einer Konfigurationsdatei – es wird im Hintergrund nämlich aus den Parametern eine **temporäre** Konfigurationsdatei erstellt, welche gleich strukturiert ist wie eine normale. Der Unterschied liegt vor allem darin, dass diese bei jedem Start mit Parametern wieder überschrieben wird. Diese Implementation wurde so implementiert, um im *Hintergrund* überall die gleiche Struktur zu wahren und die späteren Abfragen von Inventory und den ILO-Systemen zu vereinfachen.

Da PowerShell mit den ParameterSets eine gute Möglichkeit besteht, verschiedene Arten, wie das Skript gestartet werden kann, lässt sich das Komplizierte über die Parameter lösen – es muss definiert werden, welche Parameter unabdingbar sind und welche für alle Arten zugänglich sein müssen. Das ist teils zwar umständlich (wie bspw. beim untenstehenden Benutzernamen), aber verkürzt den restlichen Teil erheblich.

        [Parameter(Mandatory = $true,

            ParameterSetName = "ServerPath")]

        [Parameter(Mandatory = $true,

            ParameterSetName = "ServerArray")]

        [Parameter(Mandatory = $true,

            ParameterSetName = "Inventory")]

        [Parameter(

            ParameterSetName = "Config")]

        [Parameter(

            ParameterSetName = "None")]

        [string]

        $Username,

Codesnippet 5: ParameterSet am Benutzer

Wie oben schon zu sehen sind, gibt es im Grunde fünf verschiedene Arten. Diese sind deckungsgleich mit der Planung. Einzig das ParameterSet *None* ist neu – dieses ist nötig, dass das Skript auch ohne Parameter gestartet werden kann.

Aus diesen Parametersets ergibt sich auch die Syntax:

Get-HWInfoFromILO -configPath <string> [Other Parameters]

Get-HWInfofromILO -ServerPath <string> -Username <string> -Password <string> [Other Parameters]

Get-HWInfofromILO -server <array> -Username <string> -Password <string> [Other Parameters]

Get-HWInfofromILO -SearchStringInventory <string> -Username <string> -Password <string> [Other Parameters]

Codesnippet 6: Syntax Get-HWInfofromILO

Das Handling der Parametersets erfolgt durch ein einfaches Switch-Case. Erwähnenswert ist auch die Funktion, welche die Änderungen speichert, die zusätzlich mitgegeben werden (z.B. mit *configPath* gestartet, aber mit der Änderung, dass das logLevel = 0 ist. Im Hintergrund überprüft die Update-Funktion dann vorhandene Änderungen und speichert diese ab.

Update-Config -configPath $configPath -LoginConfigPath $LoginConfigPath -ReportPath $ReportPath -LogPath $LogPath

Codesnippet 7: Beispiel Update-Config

* + 1. Logfunktion

Loglevels

Im Laufe der Entwicklung wurden die folgenden Loglevels definiert, die von *Keinen Logs* bis zu fast Zeilenweise alles beinhalten.

|  |  |
| --- | --- |
| **Level** | **Bezeichnung** |
| 0 | Keine Logs |
| 1 | Keine Logs, ausser bei Fehlern |
| 2 | Logs nur bei Start, Ende des Skripts und bei Fehlern |
| 3 | Logs nur bei den wichtigsten Schritten   * Auslesen der Konfiguration * Pingtest * Abfrage von Inventory * Abfrage von ILO * Speichern in die Dateien * Ende des Skripts * Fehler |
| 4 | Genauere Logs bei den wichtigsten Schritten   * Auslesen der Konfiguration * Pingtest   + Ergebnis der einzelnen Server * Abfrage von Inventory   + Ergebnis der einzelnen Server * Abfrage von ILO   + Abschluss der einzelnen Server   + Abschluss der Abfrage von ILO * Speicherfunktion   + In JSON   + In CSV * Ende des Skripts * Fehler |
| 5 | Genaue Logs der wichtigsten Schritte & Logs bei Beginn und Ende einer aufgerufenen Funktion |
| 6 | Genaue Logs der wichtigsten Schritte & genaue Logs der Funktionen. |

Implementierung

Die Logfunktion ist zentral geregelt – ausserdem ist sie so implementiert, dass sie keine anderen nichtstandard-Funktionen aufruft – so lässt sie sich über alle ohne Probleme einsetzen, ohne das es zu Rekursiven Funktionen kommt.

Die Funktion erstellt jeweils pro Tag ein einzelnes Logfile, in welchem alle Vorkommnisse des Tages erfasst sind. Die Speicherung erfolgt jeweils mit Zeitstempel und die einzelnen Ausführungen werden durch eine Linie getrennt.

2025.03.10 14:30:49 --------------------------------------

ILO-Inventorizer has been started.

2025.03.10 14:30:49 Configure new Configuration

2025.03.10 14:30:49 Import Configuration

2025.03.10 14:30:49 Start Updating Configuraton File

Codesnippet 8: Ausschnitt Logdatei

* + 1. Pingtest

Der Pingtest besteht aus zwei Teilen: Einem nslookup und einem Test-Connection – die Teile geben unterschiedliche Informationen zurück, die für den Nutzer beim Problemlösen sehr nützlich sein können:

* Nslookup stellt eine Anfrage an den DNS-Server, wodurch sich sagen lässt, ob ein abgefragter Server oder Computer überhaupt existiert
* Test-Connection wird dann dazu verwendet um zu prüfen, ob man den anderen Server auch erreichen kann.

Das heisst für den User, dass wenn schon beim nslookup keine Antwort kommt, dann ist der Server im Netzwerk nicht erreichbar oder nicht vorhanden. Gibt es hingegen beim Test-Connection keine Antwort (trotz Auftauchen im lookup), bedeutet das lediglich dass man keinen Zugriff auf das Teilnetz hat.

Der Pingtest wird per Loop mit allen eingegebenen Systemen durchgeführt, wobei die Systeme abgefangen werden, welche nicht erreichbar sind und somit auch eine ILO-Abfrage keinen Sinn machen würde.

        [Array]$reachable;

        foreach($srv in $serverJSON){

            if(Invoke-PingTest $srv){

                $reachable += $srv;

            }

        }

Codesnippet 9: Aufruf Pingtest

* + 1. Abfrage von Inventory

Die Abfrage von Inventory wird dazu benötigt, anhand eines Suchstrings aus der Datenbank die ILO-Hostnamen zu finden. Hierzu wird zuerst überprüft, ob gemäss Konfiguration Inventory überhaupt ausgelesen werden soll.

Wenn dies Fall ist, wird zuerst mittels des bereits implementierten ein Pingtest an Inventory.psi.ch durchgeführt, um die Erreichbarkeit zu verifizieren. Anschliessend wird die eigentliche Abfrage durchgeführt, in der neben dem Hostnamen für ILO auch die Bezeichnung im Inventory abgefragt wird, die später für die CSV-Dateien wichtig sind.

Als letztes werden schliesslich der Hostname für die ILO-Abfrage zur Nachverfolgung im server.json gespeichert, von dem sie später wieder ausgelesen werden.

Die Abfrage von Inventory verwendet die bereitgestellte JSON-API unter [IV4 - JSON API Introspection](https://inventory.psi.ch/#action=Introspection), genauer gesagt den Endpoint «FindObjects», der für die generelle Suche verwendet werden kann. Dieser ist am besten geeignet, da er frei konfigurierbar ist was die durchsuchten Felder und Operatoren angeht.

$uri = "https://inventory.psi.ch/DataAccess.asmx/FindObjects";

$headers = @{"Content-Type" = "application/json; charset= utf-8" };

$body = @{

"search" = @{

      "query"   = @(@{

            "Field"    = "ANY"

                  "Operator" = "Contains"

                  "Value"    = $searchStringInventory

                  })

            "columns" = @(

                  "Label",

                  "Hostname",

          $remoteMgmntField,

            "Serial",

                  "Part Type",

                  "Facility",

                  "MAC 1",

                  "MAC 2",

                  "MAC 3",

                  "MAC 4",

                  "Mgnt MAC",

                  "HW Status",

                  "OS"

                  )

           }

         } | ConvertTo-Json -Depth 4

$resp = Invoke-RestMethod -Uri $uri -Method Post -Headers $headers -Body $body -HttpVersion 3.0

Codesnippet 10: Abfrage Inventory

Es wird neben den Hostnamen weitere Informationen abgespeichert, wie die MAC-Adressen oder die Seriennummer, die nicht direkt für die ILO-Abfrage nützlich sind, aber für den Nutzenden zur Konfiguration der Computer sehr nützlich sein kann.

Um für die Abspeicherung der CSV-Dateien einfacher an das *Label*-Feld zu kommen, braucht es eine Funktion, welche die Antwort von Inventory vom unschönen verschachtelten Array zu einem schönen Array aus Objekten konvertiert:

            $servers = (($resp).d.Rows);

            $serversClean = @();

            foreach ($srv in $servers) {

                $serversClean += [ordered]@{

                    Label         = $srv[0]

                    Hostname      = $srv[1]

}

}

Codesnippet 11: Konvertierung der Inventory-Antwort

Dieses Array wird dann zu JSON konvertiert und in einer für später und zur Nachverfolgung zwischengespeichert. Aus dieser Datei stammen bspw. das Label, welches später bei der CSV-Speicherung wieder wichtig wird.

* + 1. Abfrage von ILO

Die Abfrage von ILO ist simpel aufgebaut – sobald mit Connect-HPEILO die Verbindung aufgebaut ist, lässt sich durch Piping der jeweilige Server abfragen. Probleme gibt es vor allem zwischen den Versionen von ILO – bei den verwendeten DL380-Servern der 8. Generation beispielsweise gibt es einige Funktionen, die nicht genutzt werden können (z.B. PCI-Devices). Ausserdem gibt es selbst unter den gleichen Funktionen sind diese teilweise anders strukturiert.

Aus diesem Grund und daraus, dass nur die wichtigsten Dinge abgespeichert werden sollen, ist ein grosser Teil dieses Codes einfach nur dazu nötig, um über alle abgedeckten ILO-Versionen (4-6) die Eigenschaften zu vereinheitlichen und ansatzweise zu standardisieren.

Der Teil, der die Informationen des Memories ausliest sieht beispielsweise so aus:

Log 6 "`tQuerying Memory" -IgnoreLogActive

$memory = $iLOVersion -eq 4 ? ($conn | Get-HPEiLOMemoryInfo).MemoryComponent : ($conn | Get-HPEiLOMemoryInfo).MemoryDetails.MemoryData;

$memoryDetails = @();

foreach ($me in $memory) {

$memoryDetails += [ordered]@{

Location = $iLOVersion -eq 4 ? $me.MemoryLocation.ToString() : $me.DeviceLocator.ToString();

SizeMB   = $iLOVersion -eq 4 ? $me.MemorySizeMB.ToString() : $me.CapacityMiB.ToString();

}

}

Codesnippet 12: ILO-Abfrage & Standardisierung von Memory

Die Eigenschaften, welche ausgelesen und standardisiert werden beschränken sich, wenn möglich auf die Seriennummern, MACAdressen und Standortinformationen.

**Script-Messages mit Zeitstempel**

Eine der Anforderungen ans Programm ist unter anderem, dass die Abfrage des Servers mittels Messages gemacht wird, welche zusätzlich zum Text auch den Zeitstempel beinhalten – das lässt sich durch eine kleine Anpassung der Logfunktion sehr einfach bewerkstelligen:

Die Logfunktion nutzt bereits Zeitstempel und kann über die Konfiguration ausgeschaltet werden, sowie in die Konsole geloggt werden – beide Anforderungen sind also bereits vorhanden. Deswegen ist nur eine kleine Modifikation notwendig, nämlich ein neuer Parameter für die Logfunktion: [switch]$IgnoreLogActive.

Ist dieser Parameter gesetzt wird im Hintergrund nicht nur die Konfiguration ignoriert, sondern auch das Logging in eine Logdatei ausgeschalten. Dadurch wird es immer angezeigt, auch wenn in der Konfiguration das gesamte Logging ausgeschalten ist – die Logdatei wird aber trotzdem nicht zugemüllt.

# Log only if activated

if ($logActive -or $IgnoreLogActive) {

                if ($Level -le $logLevel -or $IgnoreLogActive) {

# Code

}

# If LogActive is ignored, log only to console.

if (-not $IgnoreLogActive) {

# Save Log to File

}

}

Codesnippet 13: Modifizierte Logfunktion

**Kompatibilitätsunterschiede**

Da die abgefragten Systeme eine Spanne an verschiedenen ILO-Versionen von Version 4 bis Version 6 abdecken, kommt es zwangsläufig zu Funktionen und Features, welche in der einen existiert und in anderen nicht.

Falls es nur an der Struktur ändert, wird wie oben bereits beschrieben die Struktur angepasst, falls eine Funktion gar nicht verfügbar ist, so wird lediglich die Inkompatibilitätsnachricht abgespeichert.

Dies trifft beispielsweise auf die Devices zu, die vor der ILO4 nicht abgefragt werden konnten, weshalb dort die erwähnte Statusinfo angezeigt wird.

$devices = ($conn | Get-HPEiLODeviceInventory);

$deviceDetails = @();

if ($iLOVersion -eq 4) { $deviceDetails = $devices.StatusInfo.Message; }else {

foreach ($dev in $devices.Devices) {

}

Codesnippet 14: DeviceInventory Inkompatibilität

Die Speicherung sieht dann im Objekt so aus:

"Devices": "Feature not supported on iLO3, iLO4 and iLO5 (FW Ver: 1.10,1.11,1.15,1.17).",

Codesnippet 15: Ausgabe bei Inkompatibilität

* + 1. Dateispeicherung

Die Report-Dateien werden am Ende in den Pfad hineingespeichert, welcher unter dem Pfad reportPath in der Konfigurationsdatei abgelegt ist.

JSON

Die Dateispeicherung in JSON ist die einfachere und simplere von beiden – sie umfasst die gefilterte Ausgabe vom ILO. Sie unterscheidet sich in der Implementation nur wenig von der Generierung der Konfigurationsdateien, welche auf dem gleichen Weg generiert werden: Es wird zuerst ein PowerShell-Objekt erstellt, welches dann in JSON konvertiert und schlussendlich in einer Datei gespeichert wird.

    $report | ConvertTo-Json -Depth 15 | Out-File -FilePath $name -Force;

Codesnippet 16: Dateispeicherung JSON

Um zu garantieren, dass die Ausgaben sich in einem Ordner befinden und nicht einer Datei befinden, wurde eine Funktion implementiert, welche die Datei aus dem Pfad entfernt und den Ordner erstellt, falls dieser nicht existiert:

Function Register-Directory {

    param(

        [Parameter(Mandatory = $true)]

        [string]

        $Path,

        [Parameter()]

        [switch]

        $ignoreError

    )

    try{

        if ((-not (Test-Path -Path $Path)) -and $ignoreError) {

            New-Item -Path $Path -Force -ItemType Directory;

        }else{

            throw [System.IO.DirectoryNotFoundException] "The directory at '$Path' does not exist."

        }

        $isDirectory = (Get-Item ($Path)) -is [System.IO.DirectoryInfo];

        if (-not $isDirectory) {

            $Path = $Path | Split-Path -Parent -Resolve;

        }

        return $Path.ToString();

    }catch{

        Write-Error $\_

    }

}

Codesnippet 17: Generierung des Pfades falls er nicht existiert.

Nach einem Gespräch mit dem Fachvorgesetzten wurde die Funktion so modifiziert, dass sie lediglich bei der Generation der Konfiguration den Pfad erstellt – in allen anderen Fällen muss der Anwender garantieren, dass der Pfad existiert.

CSV

Im Vergleich zur Datenspeicherung in JSON ist diejenige mit CSV komplexer, da sie stärker standardisiert ist und sich nicht wie ein Objekt, sondern wie eine Tabelle verhält – dadurch darf es unter anderem keine Verschachtelungen geben.

Die Ausgabe im CSV ist primär für den Nutzenden gedacht und nicht geeignet dafür, sie durch Automatisierungsprozesse weiter zu verwenden. Sie sind auch aus diesem Grund im Vergleich zur JSON-Datei so kurz als möglich gehalten. Aufgrund dessen ist die CSV-Ausgabe in drei verschiedene Dateien strukturiert:

* **Generell**: Diese Datei stellt +- die Standardansicht in Inventory dar – d.h. sie ist sehr kurz  
  

Abbildung 8: Resultat der generellen CSV-Datei

* **MAC:** Diese Datei stellt alle MAC-Adressen dar, egal ob NIC, Netzwerkadapter oder Devices.  
  ****

Abbildung 9: Resultat der MAC-Adressen CSV-Datei

* **Seriennummern**: Diese Datei stellt alle MAC-Adressen dar, welche gefunden werden konnten (Prozessor, Storage, Memory usw.) Ausserdem enthält sie zusätzlich dazu noch eine kurze Liste mit genaueren Details der Namen und Orte der Seriennummern, um so die einzelnen Teile besser unterscheiden zu können.  
  

Abbildung 10: Resultat der Seriennummern CSV-Datei

Überall wo ein ‘-‘ zu sehen ist, befindet wurde kein passender Wert gefunden.

Das in allen drei Dateien vorhandene Feld *Label* ist der Identifier innerhalb des Inventory – es ist daher nur vorhanden, wenn auch eine Inventory-Abfrage durchgeführt wurde und wird aus der Datei ausgelesen, in der das Resultat der Inventory-Abfrage abgespeichert wird.

**CSV-Standardisierung**

Es gibt bei der Konvertierung von PowerShellvariablen in CSV ein kleines Problem was ein Array aus Objekten angeht: Es werden nur jeweils die Keys übernommen, welche in allen Instanzen des Objektes vorkommt. Da aber je nach abgefragtem Server nicht überall gleich viele Teile eingebaut; Ein Server hat bspw. 4 RAM-Riegel, ein anderer vielleicht aber 8 oder 16, aber weil ein anderer nur 2 hat, wird in der Datei bei ALLEN nur zwei angezeigt.

Um dieses Problem zu umgehen, kann man mittels einer Funktion über alle Instanzen standardisieren, sodass alle Instanzen die gewünschten Eigenschaften besitzen – dort wo sie vorher nicht vorhanden war, wird sie mit einem ‘-‘ hinzugefügt.

Function Get-StandardizedCSV {

    param(

        [Parameter(Mandatory = $true)]

        $Report

    )

    $unique = $Report | ForEach-Object { $\_.Keys } | Select-Object -Unique

    foreach ($srvObj in $Report) {

        foreach ($uniqueMember in $unique) {

            if (($srvObj.Keys -contains $uniqueMember) -eq $false) {

                $srvObj | Add-Member -Name $uniqueMember -Value "-" -MemberType NoteProperty;

            }

        }

    }

    return $Report

}

Codesnippet 18: Standardisierung von CSV-Dateien

* + 1. Hilfestellungen

Die Hilfestellungen erfolgen pro Funktion jeweils in der Form von *comment-based help* – das ist der native Weg von PowerShell, mit dem üblicherweise dokumentiert wird. Damit lassen sich aber nur die Funktionen dokumentieren, nicht aber das Modul an sich. Aus diesem Grund wurde ein *ILO-Inventorizer.help.txt* erstellt, welches von PowerShell aufgerufen wird, wenn man den *Get-Help*-Befehl auf das Modul anwendet.

Um die Nutzung zu vereinfachen zeigt die Hilfe des Modules eine kleine Tabelle mit allen *public* Funktionen und jeweils kurz zusammengefasst, was sie so machen. Ausserdem zeigt sie, wie man über die verschiedenen Hilfsparameter zur Hilfe kommt.

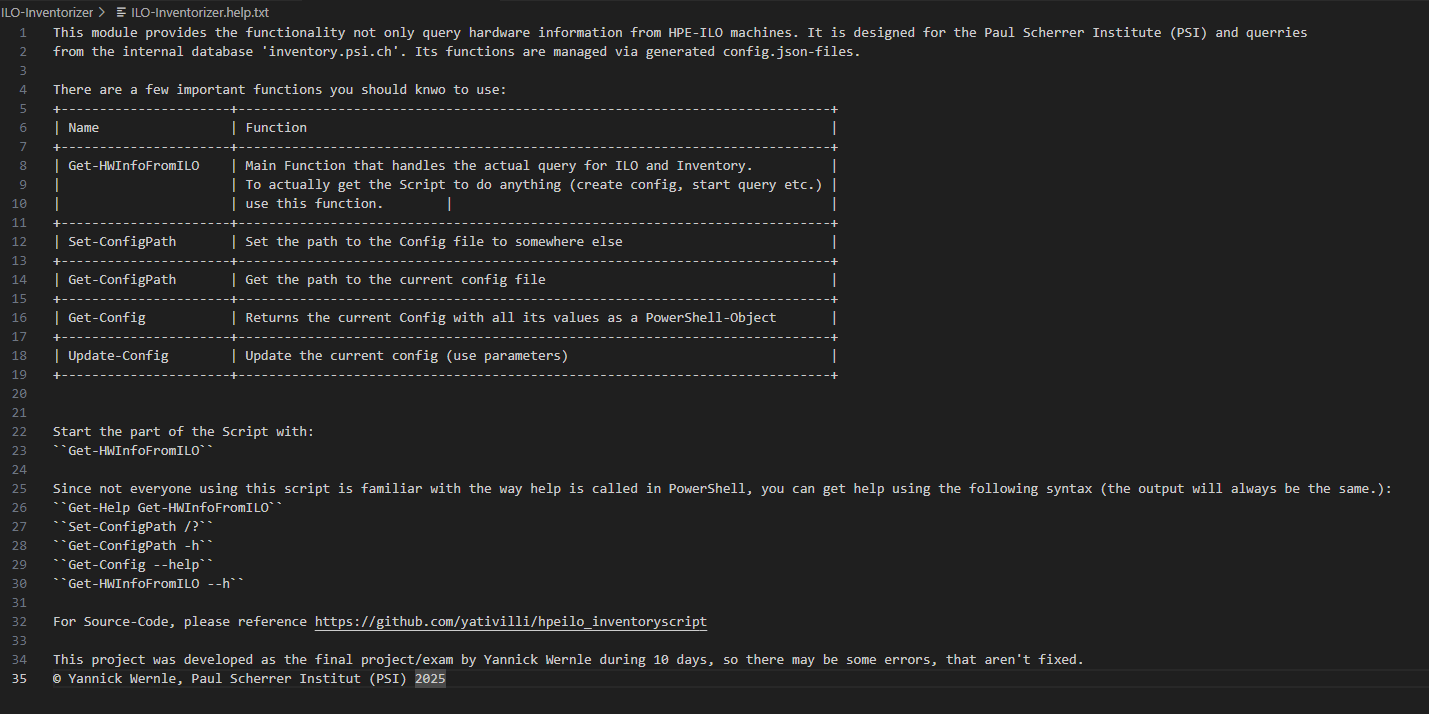


Abbildung 11: Modulhilfes

Die Comment-Based Help ist nur bei den 5 *public* Funktionen zu finden, aus dem Grund, dass die anderen Funktionen gar nicht durch den späteren User verwendet werden sollen und verwendet werden können. Das heisst, um sie zu verwenden, muss man im Code etwas ändern – wenn man so weit ist, sollten die regulären Kommentare beim Programmieren als Hilfestellungen reichen.

[UNTERTEILUNG IN KAPITEL FÜR TEILE (mysql, frontend, backend)]

Was wurde umgesetzt?

Wie wurde es umgesetzt?

Wie ist es entstanden und welche Schritte wurden umgesetzt?

Bei Umsetzung beantwortung der Fragen:

* Was,
* Warum,
* Wie
* Mit Wem
* Welche Begründung

[BILDER/CODE]

* 1. Kontrollieren
     1. Testprotokoll [NUMMER]

Testfälle

|  |  |  |  |
| --- | --- | --- | --- |
| Testfall-Nr | Testfallspez.nr. | Resultat | Bemerkung |
| TF-01 |  | OK | - |
| TF-02 |  | OK | - |

Testbericht

Was lief gut?

Was lief schlecht?

Wo gibt es Probleme?

In welchem Zustand befindet sich die Applikation gemäss Test?

* 1. Auswerten
     1. Technisches Fazit

Wurden die Anforderungen erfüllt?

Wurde genügend Dokumentiert?

Wurden Code-Best-Practices eingehalten?

Was läuft noch nicht?

* + 1. Zukunftsaussichten

Wo gibt es Verbesserungspotential?

Was sind mögliche Features, die erweitert werden könnten?

* + 1. Selbstreflexion
* Was habe ich gerne gemacht und weshalb?
* Was gelang mir gut, was weniger? Worin liegen die Gründe?
* Welche Schwierigkeiten / Probleme sind aufgetreten und was waren die Ursachen?
* Welche Erkenntnisse ziehe ich daraus?
* Sind alle Ziele erreicht worden? Weshalb nicht?
* Wurden die Ressourcen, das Material, die Personen, die Zeit optimal eingesetzt?
* Sofern ich den Zeitplan nicht einhalten konnte, weshalb war dies nicht möglich /
* warum habe ich mich verschätzt?
* Warum bin ich mit meiner Leistung zufrieden / unzufrieden?
* Was mache ich beim nächsten Auftrag anders / besser?
* Ist gewährleistet, dass gewonnene Erkenntnisse in neuen Aufgaben berücksichtigt oder aufgedeckte Mängel berücksichtigt werden?
* Sind Verbesserungsmassnahmen vorzunehmen?
* Was von meinen früheren Vorsätzen konnte ich umsetzen?
* Was habe ich gelernt?
  + 1. Persönliches Fazit

Wie ist die IPA verlaufen?

Wie ist das Resultat der IPA?

Was habe ich gelernt?

Wo gab es Probleme?

# Literaturverzeichnis

Pester. (kein Datum). *Pester.dev*. Von Quick Start: https://pester.dev/docs/quick-start abgerufen

1. Abbildungsverzeichnis

[Abbildung 1 – Beispiel Abbildung 5](#_Toc191906088)

**Nicht selbst erstellt:**

Abb. XY. Name, Vorname. (Jahr). Titel. Abgerufen am [Datum] von <http://www.xxxxx.com>

1. Tabellenverzeichnis

**Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.**

1. Codesnippetverzeichnis

[Codesnippet 1 - Beispiel Code 5](#_Toc191906041)

1. Glossar (Alphabetisch sortiert)

|  |  |
| --- | --- |
| Fachbegriff | Erläuterung |
| IPERKA | IPERKA ist eine Projektmanagementmethode, welche als Wasserfallmodell, bestehend aus den Phasen Informieren, Planen, Entscheiden, Realisieren, Kontrollieren, Auswerten. |
| Inventory.psi.ch | Interne Datenbank des PSI zur Erfassung und Wartung von Elektronischer und informationstechnischen Gegenständen, Teilen und Servern. |
| HPEiLOCmdlets |  |
| HTTPS |  |
| Officenetz |  |
| Maschinennetz |  |
| Pester |  |
| ParameterSet |  |

1. Abkürzungsverzeichnis

|  |  |
| --- | --- |
| Abkürzung | Erläuterung |
| PSI | Abkürzung des Lehrbetriebs «Paul Scherrer Institut» |
| PAP | Abkürzung für «Programmablaufplan» |
|  |  |
| NIC |  |

1. Selbstständigkeitserklärung

Hiermit erkläre ich, Yannick Wernle, dass ich die vorliegende IPA-Arbeit selbstständig und ihne fremde Hilfe verfasst habe. Sämtliche verwendeten Quellen und Hilfsmittel sind im Bericht vollständig und korrekt angegeben. Ich bestätige zudem, dass alle KI-Anfragen, die während der Erstellung dieser Arbeit gemacht wurden, im Bericht dokumentiert sind.

Ort, Datum: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Unterschrift: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Anhang
   1. Protokoll Erster Expertenbesuch

|  |  |  |
| --- | --- | --- |
| Datum | Teilnehmer | Dauer |
| 05.03.2025 | HEX, VF, KAND | Ca. 45 min |

**Traktanden**:

* Kurze Vorstellung,
* Besprechung Aufgabenstellung
* Zeitplan anschauen & Stand der Dokumentation überprüfen,

**Bemerkungen HEX & VF:**

* Bei Glossar auf alphabetische Reihenfolge achten,
* Bei der Projektorganisation begründen, wieso IPERKA und nicht einfach «weil ich es schon verwende»
* Bei der Entscheidungsphase nicht etwas nehmen wo man schon im Vorherein weiss, was man nimmt.
* Quellenangaben und KI sind erlaubt und es reicht einfach ein Link, resp. zu erwähnen, wie man es verwendet hat, aber eine konkrete Angabe des Prompts ist nicht nötig.
* (VF) Beim PAP sollte die Versionsüberprüfung am Anfang stattfinden.

**Fragen:**

* Was abgeben? --> Falls im Git schon alles vorhanden ist, dann reicht dass, ansonsten der Rest dazu (Dokumentation, Zeitplan usw.)