Prüfungsteil A

Prüfling (private Anschrift):	Ausbildungsbetrieb:
Zadjad Rezai [redacted]	Drägerwerk AG & Co. KGaA Moislinger Allee 53-55 23558 Lübeck

Bestätigung über durchgeführte Projektarbeit

diese Bestätigung ist mit der Projektdokumentation einzureichen

Ausbildungsberuf (bitte unbedingt angeben):

Fachinformatiker (Anwendungsentwicklung)

Projektbezeichnung:

Entwicklung eines Test-Tools für SDC-Netzwerke - Tool zur Unterstützung der ServicetechnikerInnen

Projektbeginn: 02.03.2020 Projektfertigstellung: 20.03.2020 Zeitaufwand in Std.: 70

Bestätigung der Ausbildungsfirma:

Wir bestätigen, dass der/die Auszubildende das oben bezeichnete Projekt einschließlich der Dokumentation im Zeitraum

vom: 02.03.2020 bis: 20.03.2020 selbständig ausgeführt hat.

Projektverantwortliche(r) in der Firma:

[redacted]

Vorname Name Telefon Unterschrift

Ausbildungsverantwortliche(r) in der Firma:

[redacted]

Vorname Name Telefon Unterschrift

Eidesstattliche Erklärung:

Ich versichere, dass ich das Projekt und die dazugehörige Dokumentation selbständig erstellt habe.

Ort und Datum: Lübeck, 04.05.2020 Unterschrift des Prüflings: Zadjad Rezai Digital unterschrieben von Zadjad Rezai Datum: 2020.05.04 08:29:37 +02'00'

Abschlussprüfung Sommer 2020

Fachinformatiker für Anwendungsentwicklung

Dokumentation zur betrieblichen Projektarbeit

Entwicklung eines Test-Tools für SDC-Netzwerke

Tool zur Unterstützung der ServicetechnikerInnen

Abgabetermin: Lübeck, den 04.05.2020

Prüfungsbewerber:
Zadjad Rezai
[redacted]

Ausbildungsbetrieb:
Drägerwerk AG & Co. KGaA
Moislinger Allee 53-55
23558 Lübeck

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

GUI	Graphical User Interface
JSON	JavaScript Object Notation
MVC	Model View Controller
SDC	IEEE 11073 Service-oriented Device Connectivity
BUITS	BU IT & Systems
WSGI	Web Server Gateway Interface
HTTP	Hypertext Transfer Protocol

Glossar

D | E | F | P | R | W

D

DRY

Don't repeat yourself (DRY, englisch für „wiederhole dich nicht") ist ein Prinzip - besonders bekannt in der Softwareentwicklung -, das verlangt, Redundanz zu vermeiden oder zumindest zu reduzieren.

E

Entwurfsmuster

Entwurfsmuster (englisch design patterns) sind bewährte Lösungsschablonen für wiederkehrende Entwurfsprobleme sowohl in der Architektur als auch in der Softwarearchitektur und -entwicklung. Sie stellen damit eine wiederverwendbare Vorlage zur Problemlösung dar, die in einem bestimmten Zusammenhang einsetzbar ist.[1]

F

Flask

Flask ist ein leichtgewichtiges WSGI Webanwendungs-Framework. Es wurde entwickelt, um einen schnellen und einfachen Einstieg in Webanwendungs-Frameworks zu ermöglichen, und bietet die Möglichkeit zur Skalierung auf komplexe Anwendungen.[2]

P

Python

Python ist eine interpretierte, interaktive, objektorientierte Programmiersprache. Sie umfasst Module, Exceptions, dynamische Typisierung sowie dynamische Datentypen.

R

REST API

REST steht für REpresentational State Transfer, API für Application Programming Interface. Gemeint ist damit ein Programmierschnittstelle, die sich an den Paradigmen und Verhalten des World Wide Web (WWW) orientiert und einen Ansatz für die Kommunikation zwischen Client und Server in Netzwerken beschreibt.[3]

[1] Vgl. Wikipedia [2020a]
[2] Vgl. The Pallets Project
[3] Siehe https://www.cloudcomputing-insider.de/was-ist-eine-rest-api-a-611116/, letzter Zugriff 06. März 2020

W

WSGI

Web Server Gateway Interface (WSGI) ist die Webserver-Gateway-Schnittstelle. Es ist eine Spezifikation, die beschreibt, wie ein Web-Server mit Web-Anwendungen kommuniziert und wie Web-Anwendungen miteinander verkettet werden können, um eine Anfrage zu bearbeiten.

WSGI ist ein Python-Standard, der in PEP 3333 ausführlich beschrieben wird.[4]

[4]Siehe https://wsgi.readthedocs.io/en/latest/what.html, letzter Zugriff 03. März 2020

1 Einleitung

Die folgende Projektdokumentation schildert den Ablauf des IHK-Abschlussprojektes, welches der Autor im Rahmen seiner Ausbildung zum Fachinformatiker Fachrichtung Anwendungsentwicklung durchgeführt hat. Ausbildungsbetrieb ist die Drägerwerk AG & Co. KGaA. Die Drägerwerk AG & Co. KGaA (auch bekannt als **Dräger**) stellt Produkte der Medizin- und Sicherheitstechnik her. Unser 1889 in Lübeck gegründetes Familienunternehmen besteht in fünfter Generation und hat sich zu einem globalen börsennotierten Konzern entwickelt. Dräger beschäftigt weltweit mehr als 14.000 Mitarbeiter und ist in über 190 Ländern der Erde vertreten.

1.1 Projektbeschreibung

Im Rahmen der Einführung des Kommunikationsprotokolls für Medizingeräte IEEE 11073 Service-oriented Device Connectivity (SDC), *dessen Einsatzbereich in (potentiell) kritischen Bereichen der Patientenversorgung liegt*[5], müssen unsere Kunden ihre Netzwerkstruktur anpassen, um den neuen Anforderungen gerecht zu werden.

SDC zielt darauf hin eine herstellerunabhängige Kommunikationsschnittstelle für medizinische Geräte zu bieten und die Drägerwerk AG & Co. KGaA gehört zu den federführenden Unternehmen im Entscheidungsgremium für das SDC-Protokoll. Das Ziel ist es, einen vereinfachten *Release-Cycle* [6] sowie simplere Integrationsmöglichkeiten für alle Hersteller in der Branche zu bieten[7].

Eine Fehlkonfiguration des Netzwerkes könnte fatale Folgen haben, da SDC in besonders kritischen Einsatzbereichen wie z. B. Operationssälen genutzt wird und daher einwandfrei funktionieren muss.

Das Ergebnis dieses Projektes soll zukünftig Weltweit in allen Dräger Kundenstandorten genutzt werden, weshalb Softwarequalität und Dokumentation besonders wichtig sind.

1.2 Projektziel

Ziel des Projektes ist es, eine Software zu entwickeln, welche den Servicetechniker/innen erlaubt Netzwerke auf fehlerhafte Konfiguration mit Bezug auf das SDC Protokoll zu untersuchen.

Diese Software soll in einer Form entwickelt werden, dass sie vollkommen portabel ist und kein hohes software-technisches Verständnis verlangt.

Das Ziel ist es entweder zwei unabhängige Softwarelösungen zu bieten:

- Eine Anwendung mit graphischer Oberfläche
- Eine REST API,

[5] Vgl. IEEE 11073 SDC Wikipedia [2020b]
[6] dt. Veröffentlichungszyklus
[7] Vgl. OR.NET Kasparick u. a.

oder eine Softwarelösung, die ausschließlich über die Kommandozeile genutzt werden kann.

1.3 Projektumfeld

Auftraggeber des Projektes ist die Abteilung BU IT & Systems (BUITS).

Zu den Aufgaben der Abteilung des BUITS gehören u. a. die Vorbereitung und Entwicklung des SDC-Protokolls innerhalb der Dräger-Landschaft. Aktuell werden die SDC-fähigen Geräte ausschließlich an ausgewählte Kunden in der sogenannten *Early-Adopter-Region*[8] geliefert, damit sich die regionalen Support-Teams auf das SDC-Protokoll vorbereiten und erste Erkenntnisse sammeln können.

Aus diesen ersten Abnahmen wurde ersichtlich, dass eine Software notwendig ist, um das interne Netzwerk der Kunden schnell und zuverlässig auf Fehlkonfigurationen überprüfen zu können.

Bei einigen Kunden wurde die Installation um einige Wochen verzögert, da es Probleme mit der Einrichtung ihres Netzwerkes gibt. Eine Fehlerquelle eindeutig zu benennen nimmt enorm viel Zeit in Anspruch, da ein manueller Test des Netzwerkes lange dauert und Expertise im Bereich der Netzwerktechnik sowie SDC benötigt.

1.4 Projektbegründung

Für die problemlose Installation und den anschließenden Betrieb von Geräten, die über SDC kommunizieren, muss das Krankenhausnetzwerk in geeigneter Weise konfiguriert sein. Diese Anforderungen werden der Kunden-IT rechtzeitig vor der Installation mitgeteilt.

Fehlende Genauigkeit in der Kommunikation der Anforderungen, fehlerhafte Konfiguration durch menschliches oder technisches Versagen, führen bei der Einführung von SDC-fähigen Geräten zu Problemen, welche Zeit und damit Geld kosten.

Oft ist es schwierig auf Anhieb zu erkennen wo genau das Problem liegt; z. B. welche Ports nicht nach Vorgabe konfiguriert sind, welche Dienste nicht funktionieren, etc.

Bei Beginn der Arbeiten vor Ort ist es für die Servicemitarbeiter/innen aktuell nicht möglich, auf einfache Weise zu prüfen, ob die Konfiguration korrekt erfolgt ist. Bei auftretenden Problemen während der Installation ist es dadurch schwierig zu unterscheiden, ob diese in den Geräten liegen (z. B. falsches oder fehlendes Zertifikat), oder ob es am Netzwerk liegt.

Die aus diesem Projekt entstehende Software wird der Drägerwerk AG & Co. KGaA sowie den jeweiligen Kunden enorme Kosteneinsparung bringen, da die Fehlersuche verkürzt und vereinfacht wird. Genaue Zahlen können nicht benannt werden, da es von Kunde zu Kunde unterschiedlich ist, jedoch liegt die Ersparnis pro Installation im Bereich von Stunden bis Wochen - je nach Komplexität des vorliegenden Kundennetzwerkes.

[8] Stand März 2020: Große Teile des europäischen Raumes

1.5 Projektschnittstellen

Die technischen Schnittstellen dieses Projektes, sind weder Geräte noch Dienste, sondern das Netzwerk selbst. Die Software arbeitet auf der dritten und vierten Schicht des OSI-Schichtenmodells[9].

Genehmigt und finanziert wird dieses Projekt durch die Drägerwerk AG & Co. KGaA, da Benutzer der Anwendung Servicetechniker/innen der Drägerwerk AG & Co. KGaA sind.

Selbstverständlich wird das Produkt dem direkten Kunden - BUITS - vorgestellt, welche dann das Wissen innerhalb der Dräger-Strukturen an alle Anwender vermittelt.

Die konkreten Endanwender/innen sind weltweit alle Servicetechniker/innen, welche für den Aufbau und den Test der SDC-fähigen Drägerwerk AG & Co. KGaA Flotte zuständig sind.

1.6 Projektabgrenzung

Da der Projektumfang beschränkt ist, soll das Bereitstellen der Software nicht Bestandteil des Abschlussprojektes sein.

2 Projektplanung

2.1 Projektphasen

Für die Umsetzung des Projektes standen dem Autor 70 Stunden zur Verfügung. Diese wurden vor Projektbeginn auf verschiedene Phasen verteilt, die während der Softwareentwicklung durchlaufen werden. Eine grobe Zeitplanung sowie die Hauptphasen lassen sich der Tabelle 1 entnehmen.

Projektphase	Geplante Zeit
Analysephase	8 h
Entwurfsphase	16 h
Implementierungsphase	30 h
Abnahmetest der Fachabteilung	1 h
Einführungsphase	1 h
Erstellen der Dokumentation	9 h
Pufferzeit	5 h
Gesamt	**70 h**

Tabelle 1: Zeitplanung

Eine detaillierte Zeitplanung finden Sie im Anhang A.1: Detaillierte Zeitplanung auf Seite i.

[9] OSI-Schichtenmodell Wikipedia [2020b]

2.2 Abweichungen vom Projektantrag

Es gab einige Abweichungen zur im Antrag angegebenen Zeitplanung. Ein Soll-ist-Vergleich befindet sich in Sektion 9.1.

Da sich die Softwarelösung in einem Bereich befindet, der neu für den Autor war, gab es einige kleinere Probleme, für die man sich einlesen musste, weshalb vor allem die Implementierungsphase länger Zeit benötigt hat als erwartet.

Insgesamt wurde dadurch jedoch nur der geplante Puffer aufgebraucht.

2.3 Ressourcenplanung

In der Übersicht, welche sich im Anhang A.2: Verwendete Resourcen auf Seite ii befindet, sind alle Ressourcen aufgelistet, die für das Projekt eingesetzt wurden. Damit sind sowohl Hard- und Softwareressourcen als auch das Personal gemeint.

Bei der Auswahl der verwendeten Software wurde darauf geachtet, dass diese kostenfrei (z. B. als Open Source) zur Verfügung steht oder bereits Lizenzen für diese im Besitz der Drägerwerk AG & Co. KGaA sind. Dadurch sollen anfallende Projektkosten möglichst gering gehalten werden.

2.4 Entwicklungsprozess

Bevor mit der Realisierung des Projektes begonnen werden konnte, musste sich der Autor für einen geeigneten Entwicklungsprozess entscheiden. Dieser definiert die Vorgehensweise, nach der die Umsetzung erfolgen soll. Für das Abschlussprojekt wurde von dem Autor ein agiler Entwicklungsprozess gewählt. Hierbei soll in Anlehnung an das Vorgehensmodell *Scrum* gearbeitet werden. Der iterative Prozessablauf sowie die ständige Konsultation der Interessenvertreter sind einige der Merkmale, die den agilen Entwicklungsprozess charakterisieren.[10]

Die relativ kurzen Iterationszyklen ermöglichen eine flexible Umsetzung der Anforderungen, so dass die Ergebnisse in relativ kurzer Zeit der Abteilung präsentiert werden können. Aufgrund dieser Tatsache wurde bei der Projektplanung in Abschnitt 2.1 für die Entwurfsphase vergleichsweise wenig Zeit eingeplant, da sich Teile dieser Phase erst im Laufe der Entwicklung der Software ergeben.

Die ständige wechselseitige Kommunikation mit den Abteilungen und das von ihnen erhaltene Feedback gewährleisten auch eine bessere Reaktionsfähigkeit auf spätere Änderungswünsche und fördern die Erzielung besserer Ergebnisse. Gleichzeitig kann die Fachabteilung bereits mit dem zu entwickelnden System vertraut gemacht werden. Dies hat wiederum den Vorteil, dass bei der Projektabnahme Zeit gespart werden kann. Aus diesem Grund wurde bei der Projektplanung auch für diese Phase relativ wenig Zeit berücksichtigt.

[10] Vgl. Schwaber und Sutherland

Da die Größe des Projekts und die Anzahl der Projektmitarbeiter unter der empfohlenen Menge von drei Personen[11] liegt, ist es nicht das Ziel sich an alle Empfehlungen des *Scrum-Frameworks* zu halten. U. a. werden keine täglichen Meetings[12] organisiert.

Der agile Entwicklungsprozess wird durch die Praxis der testgetriebenen Entwicklung erweitert, sofern es die Zeit erlaubt. Hier werden die Softwaretests bereits vor der Implementierung des eigentlichen Quellcodes geschrieben. Da bei diesem Vorgehen die Struktur und der Einsatz der einzelnen Module im Vorfeld berücksichtigt werden müssen, haben die Tests nicht nur eine verifizierende Funktion für den Entwickler, sondern können auch als Unterstützung bei der Auswahl der Architektur dienen.

Bei der testgetriebenen Entwicklung werden die folgenden drei Phasen iterativ (Red-Green-Refactor) durchlaufen:[13]

- **Test schreiben, welcher fehlschlägt:**
  Zunächst wird ein Test definiert, um die korrekte Funktionalität einer neuen Funktion zu überprüfen. Dieser Test wird jedoch fehlschlagen, da diese Funktionalität noch nicht implementiert wurde.
- Implementierung der Funktionalität, so dass der Test nicht mehr fehlschlägt
- **Refactorisierung**
  Der Quellcode kann nun umstrukturiert und verbessert werden, so dass er verständlicher und übersichtlicher wird. Die bereits definierten Tests stellen sicher, dass das Verhalten des Programms nicht versehentlich verändert wird.

3 Analysephase

3.1 Ist-Analyse

Momentan gibt es keine völlig vergleichbare Software auf dem Markt oder intern bei Dräger.

Die BUITS Abteilung wünscht sich eine Software auf OSI-Schicht 3 bis 4[14], welche ausschließlich das Netzwerk auf die von Ihnen festgesetzten Parameter überprüft.

Die Software soll außerdem sehr einfach weiterzugeben sein, keine Administratorrechte oder Nebensoftware benötigen und simple zu nutzen sein.

Folgendes soll möglich sein:

- Überprüfe, ob der Testserver erreichbar ist

[11] Vgl. Schwaber und Sutherland, [unter *Development Team Size*]
[12] Vgl. Schwaber und Sutherland, [unter *Daily Scrum*]
[13] Siehe https://zrezai-dev.de/methodik/warum-tdd/, letzter Zugriff 13.03.2020
[14] Siehe Wikipedia [2020c]

- Überprüfe, ob bestimmte Ports nutzbar sind (beidseitig)[15]
- Überprüfe, ob Multicast auf bestimmten Ports und Netzwerkinterfaces funktioniert (beidseitig)
- Überprüfe, ob die Systemzeit mit der Zeit von verschiedenen NTP-Servern übereinstimmt (beidseitig)

Die Wünsche der Abteilung können z. B. durch eine Anwendung mit Graphical User Interface (GUI), welche mit einer zweiten Anwendung auf einem Testserver kommuniziert, erfüllt werden.

3.2 Wirtschaftlichkeitsanalyse

Da die Drägerwerk AG & Co. KGaA ständig mit der technologischen Entwicklungen mitgehen muss, gibt es für das Unternehmen kaum einen Weg an einer Software vorbei, welche den Kunden und den eigenen Mitarbeiter/innen den Umstieg und Einstieg in diese neue Technologie - hier SDC - erleichtet. Durch die Automatisierung der Netzwerküberprüfung werden viele Arbeitsschritte gespart, welches selbstverständlich auch für die Wirtschaftlichkeit einer solchen Software spricht.

Daher ist die Frage, ob diese Software wirtschaftlich sinnvoll ist, mit einem klaren Ja zu beantworten.

3.2.1 „Make or Buy“-Entscheidung

Da SDC noch relativ neu auf dem Markt ist und aktuell in seiner Einführungsphase ist, gibt es nicht viel Auswahl bei externen Lieferanten. Dräger ist aktuell selbst noch in der Lernphase, weshalb es nach aktuellem Stand nicht vorteilhaft sein kann sich auf ein externes Produkt festzulegen, welches nicht ohne Umwege oder lange Wartezeiten angepasst werden kann.

Das einzige Produkt, welches den Anforderungen nahekommt ist **Nagios Xi**[16]. Nagios Xi bietet jedoch viel mehr als tatsächlich benötigt und produziert damit unnötigen Overhead bei den Kosten und beim Ressourcenverbrauch. Obwohl Nagios Xi mehr bietet als benötigt, gibt es Dinge, die aktuell nicht geboten werden wie zum Beispiel eine Zweiwege-Multicast-Überprüfung [17].

Zu den Herausforderungen des Projekts zählt nicht nur das Abdecken der Anforderungen, sondern auch das Ermöglichen von Verbesserungen und Erweiterungen.

Aufgrund dessen haben wir uns gegen einen Kauf von externer Software entschieden.

[15] Auf Clients und Server
[16] Siehe https://www.nagios.com/products/
[17] Eine Multicast-Nachricht wird von Client abgesendet und von Server abgehört. Danach sendet der Server die selbe Nachricht über die selbe Multicast-Gruppe heraus und der Client hört diese ab. Beide bestätigen sich gegenseitig den Erhalt der Nachricht.

3.2.2 Projektkosten

Die Projektkosten, die während der Entwicklung des Projektes anfallen, sollen im Folgenden kalkuliert werden. Dafür müssen neben den Personalkosten, die durch die Realisierung des Projektes verursacht werden, auch noch die Aufwendungen für die Ressourcen (Hard- und Software, Büroarbeitsplatz etc.) berücksichtigt werden. Da die genauen Personalkosten nicht herausgegeben werden dürfen, wird die Kalkulation mit angenommenen Stundensätzen gerechnet.

Berechnung (verkürzt) Die Kosten für die Durchführung des Projekts setzen sich sowohl aus Personal-, als auch aus Ressourcenkosten zusammen. Laut Tarifvertrag[18] verdient ein Auszubildender im zweiten Lehrjahr pro Monat 1078 € Brutto.

$$7\ \text{h/Tag} \cdot 220\ \text{Tage/Jahr} = 1540\ \text{h/Jahr} \tag{1}$$

$$1078\,€/\text{Monat} \cdot 13,3\ \text{Monate/Jahr} = 14337,40\,€/\text{Jahr} \tag{2}$$

$$\frac{14337,40\,€/\text{Jahr}}{1540\ \text{h/Jahr}} = 9,31\,€/\text{h} \tag{3}$$

Es ergibt sich also ein Stundenlohn von 9,31 €. Die Durchführungszeit des Projekts beträgt 70 Stunden inkl. 1 Stunde Abnahmetest. Für die Nutzung von Ressourcen[19] wird ein pauschaler Stundensatz von 15 € angenommen. Für die anderen Mitarbeiter wird pauschal ein Stundenlohn von 25 € angenommen. Eine Aufstellung der Kosten befindet sich in Tabelle 2 und sie betragen insgesamt 2837,39 €. Da die Anwendung weltweit eingesetzt werden soll, gehe ich pauschal von ca. 25 Personen aus, die die Software regelmäßig nutzen.

Vorgang	Zeit	Kosten pro Stunde	Kosten
Entwicklungskosten	69 h	9,31 € + 15 € = 24,31 €	1677,39 €
Fachgespräch	3 h	25 € + 15 € = 40 €	120 €
Abnahmetest	1 h	25 € + 15 € = 40 €	40 €
Anwenderschulung	25 h	25 € + 15 € = 40 €	1000 €
			2837,39 €

Tabelle 2: Kostenaufstellung

3.2.3 Amortisationsdauer

Durch die völlige Automatisierung der Netzwerküberprüfung entfällt natürlich ein beachtlicher Wert an Arbeitszeit, der ansonsten mit dem Durchlesen und Durchsuchen des Netzwerkdatenfluss' (z. B. über Wireshark) verbracht worden wäre.

[18] IG Metal Ausbildungvergütung IG-Metall [2018]
[19] Räumlichkeiten, Arbeitsplatzrechner etc.

Obwohl man den Datenfluss anschauen kann, kann man nicht immer erkennen woher das Problem tatsächlich kommt - man kann nur erkennen, dass bestimmte Pakete nicht versendet oder empfangen werden. Die tatsächliche Fehlersuche kann wie die Suche nach der Nadel im Heuhaufen sein.

Konkrete monetäre Vorteile des Projekts sind

- Einsparung von Lizenzkosten[20]
- Arbeitszeitersparnis

Berechnung (verkürzt) Bei einer Zeiteinsparung von 10 Minuten am Tag für jeden der 25 Anwender und 220 Arbeitstagen im Jahr ergibt sich eine gesamte Zeiteinsparung von

$$25 \cdot 220 \text{ Tage/Jahr} \cdot 10 \text{ min/Tag} = 55000 \text{ min/Jahr} \approx 917 \text{ h/Jahr} \tag{4}$$

Dadurch ergibt sich eine jährliche Einsparung von

$$917\text{h} \cdot (25 + 15)\,€/\text{h} = 36680\,€ \tag{5}$$

Die Amortisationszeit beträgt also $\frac{2837{,}39\,€}{36680\,€/\text{Jahr}} \approx 0{,}077$ Jahre ≈ 4 Wochen.

Diese Rechnung ist natürlich weit entfernt von der Realität, weil es unklar ist wie viele Minuten/Stunden/Tage oder Wochen pro Einsatz eingespart werden.

Um die Berechnung simple zu halten, habe ich eine Zeiteinsparung von 10 Minuten und nur 25 regelmäßig Anwender ausgewählt, obwohl Dräger eine weitaus größere Menge an Servicetechniker/innen hat.

Die Zeiteinsparung pro Einsatz liegt höchstwahrscheinlich im Bereich von Stunden bis Tagen, jedoch sind Einsätze grundsätzlich unregelmäßig in ihrer Natur.

3.3 Nutzwertanalyse

Da die Ergebnisse der Wirtschaftlichkeitsanalyse die Realisierung des Projektes bereits ausreichend rechtfertigen, soll an dieser Stelle auf eine detaillierte Analyse nicht-monetärer Vorteile verzichtet werden. Nicht-monetäre Vorteile der neuen Anwendung wären aber z.B. die höhere Sicherheit beim Einrichten und Überprüfen von Netzwerken, die nun gewährleistet wird.

[20] Da keine externen Produkte gekauft werden müssen

3.4 Anwendungsfälle

Um eine grobe Übersicht über die Anwendungsfälle zu erhalten, die von dem umzusetzenden Programm abgedeckt werden sollen, wurde im Laufe der Analysephase ein UseCase Diagramm erstellt. Dieses Diagramm befindet sich im Anhang A.5: UseCase Diagramm auf Seite vii und erlaubt einen groben Überblick über die Funktionsweise sowie die Zusammenarbeit der Hauptkomponenten.

3.5 Lastenheft/Fachkonzept

Ein Auszug aus dem Lastenheft findet sich im Anhang A.3: Lastenheft (Auszug) auf Seite iii. Obwohl dies leider nicht in das Konzept dees Scrum-Vorgehensmodells passt.

4 Entwurfsphase

4.1 Zielplattform

Das Abschlussprojekt soll, wie bereits in Abschnitt 1.2 erwähnt worde, in zwei Einheiten gegliedert werden. Der erste Teil besteht aus einer simplen Desktopanwendung.

Der zweite Teil besteht aus einer REST API, welche man über die Desktopanwendung erreichen kann.

Die REST API sowie die GUI sollen in der Programmiersprache Python geschrieben werden, da die Sprache Python bereits im Unternehmen etabliert ist und sich in Projekten der BUITS als geeignet erwiesen hat.

Beide Anwendungen, sowohl die REST API als auch die GUI-Anwendung müssen auf jeder beliebigen Windows-Plattform mit Netzwerkzugriff funktionieren, ohne Administratorrechte.

4.2 Architekturdesign

Die Architektur wurde so gewählt, dass zwei Anwendungen - von einander unabhängig - erstellt werden. Das Abschlussprojekt wird damit in einen Teil, die REST API, und einen anderen Teil die GUI aufgeteilt.

Die Auswahl des Web-Frameworks, mit der die REST API realisiert werden soll, wurde anhand einer Nutzwertanalyse durchgeführt. In Tabelle 3 sind die einzelnen Kriterien mit ihren jeweiligen Gewichtungen und den Bewertungen für die verschiedenen Web-Frameworks aufgelistet. Sowohl für die Gewichtung[21] als auch für die Bewertung[22] werden Werte zwischen 1 und 5 verwendet.

[21] Gewichtung: 1 = unnötig, 2 = verzichtbar, 3 = wünschenswert, 4 = erforderlich, 5 = unbedingt erforderlich
[22] Bewertung: 1 = mangelhaft, 2 = ausreichend, 3 = befriedigend, 4 = gut, 5 = sehr gut

Um die Gewichtung in die Nutzwertanalyse einfließen zu lassen, wird diese mit der Bewertung multipliziert. Der Quotient aus der gewichteten Gesamtbewertung und der Summe der Gewichtungen ergibt den Nutzwert der jeweiligen Frameworks. Der geringste Wert, der hier erreicht werden könnte, wäre 1, der höchste wäre 5.

Anhand der Entscheidungsmatrix in Tabelle 3 wurde für die Implementierung der REST API das Python-Framework Flask ausgewählt.

Eigenschaft	Gewichtung	Pyramid	Django	Flask	Eigenentwicklung
Dokumentation	5	4	5	5	0
Reenginierung	3	4	3	5	3
Generierung	3	5	5	5	2
Testfälle	4	3	2	4	3
Standardaufgaben	3	3	3	4	0
Gesamt:	**18**	**68**	**66**	**83**	**27**
Nutzwert:		**3,78**	**3,67**	**4,61**	**1,5**

Tabelle 3: Entscheidungsmatrix

Für die Wahl eines Architekturdesigns hat der Autor jeweils verschiedene Entwurfsmuster gewählt, um eine schwer wartbare, fehlerbehaftete Programmierung von unklaren Verantwortungen zu vermeiden. Innerhalb kleiner Projekte ist es nicht nötig auf Entwurfsmuster (engl. *design patterns*) zurück zugreifen, jedoch wird es nötig und wichtig diese zu nutzen, sobald man ein größeres Projekt erarbeiten möchte und u. a. Erweiterbarkeit besonders wichtig sind.

Da dieses Abschlussprojekt eine besondere Größe hat und aus mehreren großen Komponenten besteht, wird es unabdingbar auf verschiedene Entwurfsmuster zurückzugreifen.

Die REST API soll auf Basis des Model View Controller (MVC)-Architekturmusters umgesetzt werden. Demnach lässt sich jede Komponente einer Software einem der drei Bestandteile – Model, View oder Controller – dieses Musters zuordnen.[23] Jeder dieser drei Teile hat einen speziellen Aufgabenbereich, der von denen der anderen weitestgehend unabhängig ist. Das Model setzt sich hierbei aus den Daten und der entsprechenden Verarbeitungslogik zusammen, der View ist für die Präsentation bzw. Anzeige der Daten zuständig und über den Controller erfolgt die Steuerung der Anwendung.

Der Controller stellt das Bindeglied zwischen Model und View dar. Die lose Kopplung der einzelnen Komponenten erhöht die Wiederverwendbarkeit und Austauschbarkeit. Man könnte beispielsweise die Benutzeroberfläche austauschen, ohne das Model anpassen zu müssen. Außerdem können die einzelnen Komponenten durch die strikte Trennung einfacher getestet, gewartet und flexibel erweitert werden. Aufgrund dieser Vorteile soll dieses Architekturmuster für die Realisierung des Projektes verwendet werden.

Da keine Anbindung an eine Datenbank benötigt wird, gibt es in diesem Abschlussprojekt das Model - wie man es vermutet - nicht. Eine Datenbank wird hier nicht benötigt, weil keine Daten gespeichert

[23] Vgl. The Real Python

oder abgerufen werden müssen und alle Berechnungen mit den aktuell vorliegenden Werten und Gegebenheiten getätigt werden müssen. Als Model werden die verschiedenen Ausgabeschemata der REST API definiert.

Außerdem werden Funktionen und Klassen, welche von der GUI und der REST API gebraucht werden, so aufgeteilt, dass sie nicht erneut geschrieben werden müssen. Damit wird versucht dem DRY-Prinzip zu folgen.

Da Tkinter[24] das einzige GUI-Modul aus der Standardbibliothek Pythons ist, wurde es gewählt, um weitere Abhängigkeiten zu Fremdmodulen zu unterbinden. Die Weiterentwicklung des Tkinter-Moduls wäre damit gesicherter als ein anderes Open-Source-Projekt, welches nicht zur Standardbibliothek gehört.

Tkinter arbeitet direkt auf Code-Level und benötigt daher keinen Markup für die Benutzeroberfläche, wie man ihn aus Frameworks wie Qt[25] kennt.

Alle visuellen Aspekte wie beispielsweise die Größe oder Positionierung der Elemente, aus denen sich die Oberfläche zusammensetzt, werden direkt im Code definiert. Somit nutzt man eine einheitliche Sprache und muss von dieser nie abweichen.

Da es bei normaler Nutzung viele Codezeilen benötigt eine Oberfläche in Tkinter zu definieren, hat der Autor verschiedene Entwurfsmuster, wie Factory[26] und Builder[27] genutzt. So werden einzelne Komponenten der Oberfläche leichter angelegt, erweitert und verbessert ohne die darunterliegende Logik oder die komplexen Methoden oder Objektaufrufe anpassen zu müssen.

Um dem Autor eine einfache und weitestgehend automatische Interaktion mit den Nutzern der REST API zu ermöglichen, soll Marshalling[28] genutzt werden. Dabei handelt es sich um eine Methode Daten so umzuwandeln, dass man diese leicht an andere Prozesse weitergeben kann. Das Marshalling ist damit die einzige Schnittstelle der REST API zu anderen Prozessen - sollte das Ausgabeformat jemals angepasst werden müssen, muss man dies nur an einer Stelle tun.

Die Ausgabe der REST API soll vorerst die JavaScript Object Notation (JSON) sein.

Somit muss sich der Autor nicht mehr um diese Konvertierung kümmern und kann sich stattdessen stärker auf die eigentliche Verarbeitungslogik der Anwendung konzentrieren.

4.3 Entwurf der Benutzeroberfläche

Um die neue Anwendung möglichst einfach bedienen zu können, soll eine klar strukturierte, einfache Benutzeroberfläche entwickelt werden. Mit Hilfe des Mockups wurde hierfür zunächst ein Prototyp der

[24] Vgl. The Python Standard Library
[25] Vgl. The Qt Company
[26] Siehe https://refactoring.guru/design-patterns/factory-method, letzter Zugriff 12.03.2020
[27] Siehe https://refactoring.guru/design-patterns/builder, letzter Zugriff 12.03.2020
[28] Siehe https://de.wikipedia.org/wiki/Marshalling, letzter Zugriff 12.03.2020

Oberfläche angefertigt. Damit die Benutzeroberfläche am Ende den Anforderungen und Vorstellungen des Fachbereichs entspricht, wurde dieser bei der Entwurfsphase intensiv mit einbezogen.

In der GUI sollen sich auf Wunsch des Fachbereichs Eingabeelemente befinden, die erlauben das Netzwerkinterface, einen Testserver[29] sowie einen NTP-Server auszuwählen. Außerdem sollen die einzelnen Überprüfungsschritte übersichtlich angezeigt werden, sodass die Nutzer sofort sehen, welche Schritte fehlgeschlagen oder nicht fehlgeschlagen sind.

Das Mockup der Hauptansicht der GUI befindet sich im Anhang A.7: Oberflächenentwürfe auf Seite x.

Im weiteren Verlauf des Projekts haben sich - nach agiler Manier - neue Wünsche ergeben, die im Endprodukt natürlich auch in Betracht gezogen wurden. U. a. die Möglichkeit zu sehen auf welcher Seite[30] welches Ergebnis erzielt wurde.

4.4 REST API

Im Anhang 19: Klassendiagramm der REST API auf Seite xxix wird ein Klassendiagramm dargestellt, welches lediglich die Objekte darstellt, die die REST API zusammenstellen.

4.5 Geschäftslogik

Ein Klassendiagramm, welches die Klassen der Anwendung und deren Beziehungen untereinander darstellt kann im Anhang A.14: Klassendiagramm auf Seite xxvii eingesehen werden.

Die wichtigsten Komponenten, welche einen besonders großen Teil der Geschäftslogik abdecken, befinden sich innerhalb des Pakets *src.common*.

Da die REST API selbst nur den Controller darstellt und die GUI den View, haben diese keine eigenen netzwerküberprüfenden Funktionen oder Klassen.

Die GUI hat lediglich eine Klasse, den *NetwerkChecker*, welcher die Verbindung zur REST API ermöglicht.

Sie sind beide angewiesen auf Komponenten innerhalb des *src.common*-Pakets, welches alle Funktionen beinhalten.

Besonders wichtiger Teil der Geschäftslogik ist das Producer-Consumer (dt. Erzeuger-Verbraucher) Paar, welches die Kommunikation mittels Multicast gewährleistet. Beim Producer-Consumer geht es darum die Prozesssynchronisation so zu regeln, dass beide Komponenten zur selben Zeit arbeiten können - daher ein Threadingproblem.

[29] Die REST API
[30] Testserver oder Client

Die Klasse *MulticastConsumer* ermöglicht ein Objekt zu instanziieren, welches kontinuierlich, bis zu einer bestimmten Zeitgrenze, auf eine Nachricht, in einer Multicastgruppe, wartet, die sie konsumieren kann.

Die Klasse *MulticastProducer* ermöglicht ein Objekt zu instanziieren, welche eine Nachricht in eine vorgegebene Multicastgruppe versenden kann und kontinuierlich, bis zu einer bestimmten Zeitgrenze, auf eine Antwort[31] wartet.

4.6 Maßnahmen zur Qualitätssicherung

Um zur Qualitätssicherung beizutragen, wurden verschiedene Testfälle geschrieben. Diese Tests können innerhalb jeder Entwicklungsplattform (z. B. Jetbrains PyCharm) oder über die Kommandozeile aufgerufen werden.

4.7 Pflichtenheft/Datenverarbeitungskonzept

Am Ende der Entwurfsphase wurde ein Pflichtenheft erstellt. Dieses baut auf dem Lastenheft (vgl. dazu Anhang A.3: Lastenheft (Auszug) auf Seite iii) auf und beschreibt wie und womit der Autor die Anforderungen des Fachbereiches umsetzen möchte. Es dient somit als Leitfaden für die Realisierung des Projektes. Ein Auszug aus dem Pflichtenheft befindet sich im Anhang A.4: Pflichtenheft (Auszug) auf Seite iv.

5 Implementierungsphase

Anhand der in der Planungsphase getroffenen Entscheidungen konnte in der Durchführungsphase mit der konkreten Umsetzung der Anforderungen begonnen werden.

5.1 Ablaufplan

Vor Beginn der Implementierung wurde ein Sequenzdiagramm erstellt. Durch dieses Diagramm werden die Überprüfungsphasen:

1. Kontaktieren des Testservers
2. Zeitvergleich mittels NTP-Servers
3. Überprüfen der Ports
4. Überprüfen der Multicastfunktionalität

[31] Ein Acknowledgement (ACK), welches andeuetet, dass die Nachricht vom Partner erhalten wurde

visualisiert. Es wurde die Darstellung mittels Sequenzdiagramm gewählt, da sich diese gut eignen, um komplexe Abläufe strukturiert abzubilden. Außerdem erleichtern diese das Erkennen von Multithreading-Probleme, wie Deadlocks oder Race Conditions. Ein vollständiges Sequenzdiagramm kanndem Anhang A.6: Sequenzdiagramm auf Seite viii entnommen werden.

Diese Phasen werden iterativ abgearbeitet.

5.2 Implementierung der REST API

Zuerst wurde die REST API geschrieben. Dafür wurde das Python-Framework Flask genutzt. Alle JSON Rückmeldungen sowie alle Routen wurden definiert, damit diese von anderen Prozessen übernommen werden können.

Da die GUI mit diesen JSON Rückmeldungen und den Routen arbeitet, müssen diese möglichst genau definiert werden, sodass Änderungen im Nachhinein nicht regelmäßig auftreten müssen.

Ein Beispiel der Implementierung finden Sie im Anhang A.12: CheckHello-Klasse inkl. Namespace Definition auf Seite xxiv. Darin wird die JSON Rückmeldung sowie die Route definiert.

Das Beispiel stellt einen Teil des ersten Schrittes des Sequenzdiagramms dar - in diesem Schritt erhält man ein simples *hello* vom Testserver. Damit wird die Funktionalität und Erreichbarkeit des Testservers überprüft.

Die REST API wurde auf drei Namensräume aufgeteilt:

1. NTP
   Bietet Methoden, um Antworten von NTP-Servern durch den Testserver zu erhalten.
2. Response
   Bietet Methoden, um Antworten vom Testserver zu erhalten.
3. Network
   Bietet Methoden zur Überprüfung von Netzwerkparametern wie offene Ports, Multicast, etc.

Im Anhang A.10: REST API Dokumentation auf Seite xix sind Teile der Dokumentation der REST API zu sehen.

5.3 Implementierung der Benutzeroberfläche

Nach der Implementierung der REST API wurde mit der Entwicklung der GUI begonnen.

Die GUI ruft mittelt Hypertext Transfer Protocol (HTTP)-Abfragen die verschiedenen Funktionen der REST API auf und übermittelt dessen Resultate durch die Symbole, zusehen im Anhang 22: Mögliche Ergebnisse auf Seite xxxi. Außerdem werden Logs geschrieben, die die Server-Antworten dokumentieren.

Screenshots der Anwendung befinden sich im Anhang A.8: Screenshots der Anwendung auf Seite xi.

5.4 Implementierung der Geschäftslogik

Die Geschäftslogik, wie schon in der Sektion 4.5 erklärt, befindet sich zum größten Teil in *src.common*, da sich die wichtigen Bestandteile und Funktionen auf beiden Seiten der Anwendung befinden müssen.

Der Client, welcher die GUI nutzt und der Testserver auf dem die REST API läuft, müssen die Möglichkeit bieten alle Überprüfungen eigenständig auszuführen.

Wichtige Teile der Singleton-Klasse *NetworkInterface*, welche Informationen der Netzwerkschnittstellen extrahieren kann, finden Sie im Anhang A.13: NetworkInterface-Klasse auf Seite xxv.

6 Abnahmephase

Die Software wurde am Freitag, den 13. März 2020, abgenommen. Da die Software in ihrer Gesamtheit alle Schritte loggt, konnten diese Logs und die Umfangreichen Unit Tests bei der Abnahme vorgezeigt werden.

Außerdem wurde die Software in einer konkreten Testumgebung gemeinsam mit der Fachabteilung überprüft.

Ein Auszug einiger Unit Tests befindet sich im Anhang A.11: Testfall und sein Aufruf in der Entwicklungsumgebung auf Seite xxii. Dort ist auch der Aufruf des Tests innerhalb der Entwicklungsumgebung zu sehen.

7 Einführungsphase

Um die Software bereitzustellen, wurden Windows-Ausführbare Dateien[32] mittels des Python-Moduls Pyinstaller[33] erstellt.

Innerhalb des Projekts wurde die Bereitstellung vorerst manuell getätigt, jedoch wird diese Bereitstellung in naher Zukunft automatisiert.

Aufgrund der Coronavirus-Krise, musste die Arbeit teilweise verschoben werden, weshalb es momentan nicht bekannt ist wann die Automatisierung der Bereitstellung mittels Tools wie Jenkins[34] durchgeführt werden kann.

[32] .exe Dateien
[33] Siehe https://www.pyinstaller.org/, letzter Zugriff am 11.03.2020
[34] Siehe https://jenkins.io/, letzter Zugriff 11.03.2020

Wie im Projektantrag beschrieben ist die Bereitstellung und Nutzerschulung nicht Teil des Abschlussprojektes.

8 Dokumentation

Die Zielgruppe dieser Softwarelösung ist eine globale, IT-affine Gruppe aus spezialisierten Servicetechniker/innen, welche das Einrichten von Netzwerken und Infrastrukturen zu ihren täglichen Aufgaben zählen.

Deshalb wurde die Benutzerdokumentation, die Code-Dokumentation und die Dokumentation der REST API auch mit IT-Fachbegriffen und komplett in englischer Sprache geschrieben.

Ein Ausschnitt aus der erstellten Benutzerdokumentation befindet sich im Anhang A.15: Benutzerdokumentation auf Seite xxx. Die Entwicklerdokumentation wurde mittels pdoc[35] automatisch generiert. Ein beispielhafter Auszug aus der Dokumentation einer Klasse befindet sich im Anhang A.9: Entwicklerdokumentation auf Seite xv.

Auch die REST API ist dokumentiert und ein Beispiel der Dokumentation ist im Anhang A.10: REST API Dokumentation auf Seite xix.

9 Fazit

9.1 Soll-/Ist-Vergleich

Das Projektziel wurde innerhalb der gewünschten Zeit vom 02. März 2020 bis zum 20. März 2020 erfüllt und der Auftraggeber war zufrieden mit der Softwarelösung.

Wie in Tabelle 4 zu erkennen ist, konnte die Zeitplanung bis auf wenige Ausnahmen eingehalten werden.

Phase	Geplant	Tatsächlich	Differenz
Entwurfsphase	16 h	12 h	-4 h
Analysephase	8 h	8 h	
Implementierungsphase	30 h	36 h	+6 h
Abnahmetest der Fachabteilung	1 h	1 h	
Einführungsphase	1 h	1 h	
Erstellen der Dokumentation	9 h	12 h	+3 h
Pufferzeit	5 h	0 h	-5 h
Gesamt	70 h	70 h	

Tabelle 4: Soll-/Ist-Vergleich

[35] Vgl. pdoc3.github.io/pdoc

9.2 Lessons Learned

Im Zuge des Projektes konnte der Autor wertvolle Erfahrungen bzgl. der Planung und Durchführung von Projekten sammeln. Dabei wurde besonders deutlich, von welch großer Bedeutung stetige Kommunikation untereinander und Rücksprachen mit den Fachbereichen für eine erfolgreiche Projektumsetzung sind. Außerdem konnten neue Erkenntnisse in Bezug auf das Einbinden und Nutzen von Frameworks gewonnen werden. Das Nutzen von Entwurfsmustern erwies sich beispielsweise als sehr hilfreich, da es u. a. die Wartbarkeit und Erweiterbarkeit erhöht und der Autor sich somit auf die wesentlichen Komponenten der Anwendung konzentrieren konnte. Abschließend kann man sagen, dass die Realisierung des Projektes nicht nur einen Mehrwert für die Drägerwerk AG & Co. KGaA bietet, sondern auch für den Autor eine große Bereicherung war.

9.3 Ausblick

Obwohl alle im Lastenheft definierten Anforderungen realisiert werden konnten, können in Zukunft dennoch neue Anforderungen definiert bzw. Erweiterungsvorschläge entwickelt werden.

Der nächste Schritt ist, dass die Bereitstellung automatisiert wird, bei jedem Update neu gebaut und verteilt wird.

Die Softwarelösung kann in Zukunft weitere, SDC-spezifische Überprüfungen implementieren wie beispielsweise:

- Finden alle SDC-fähigen Geräte im Netzwerk
- Überprüfung der Zertifikate auf den jeweiligen Geräten
- Überprüfung der Zeit auf den jeweiligen Geräten
- und weitere..

Diese Funktionen kann als eigene REST API angeboten werden, welche die GUI als Schnittstelle zum Nutzer hat.

Literaturverzeichnis

IG-Metall 2018

IG-Metall: *Metall- und Elektroindustrie ERA – Ausbildungsvergütungen (in Euro) ab 01.04.2018*. https://www.igmetall.de/download/docs_MuE_ERA_Ausbildung_Juni2018_9bdc083c9c0ed885c63bd1b076830a817eaec814.pdf. Version: 2018, Abruf: 05.03.2020

Kasparick u. a.

Kasparick, Martin ; Schmitz, Malte ; Andersen, Björn ; Rockstroh, Max ; Franke, Stefan ; Schlichting, Stefan ; Golatowski, Frank ; Timmermann, Dirk ; Dössel, Olaf (Hrsg.): *OR.NET: a service-oriented architecture for safe and dynamic medical device interoperability*. https://doi.org/10.1515/bmt-2017-0020, Abruf: 03.03.2020

pdoc3.github.io/pdoc

pdoc3.github.io/pdoc: *phpDocumentor-Website*. https://pdoc3.github.io/pdoc/, Abruf: 06.03.2020

Schwaber und Sutherland

Schwaber, Ken ; Sutherland, Jeff: *The Scrum Guide*. https://www.scrumguides.org/scrum-guide.html, Abruf: 11.03.2020

The Pallets Project

The Pallets Project: *Flask - Open-Source PHP Web Framework*. https://palletsprojects.com/p/flask/, Abruf: 06.03.2020

The Python Standard Library

The Python Standard Library: *Graphical User Interfaces with Tk*. https://docs.python.org/3.7/library/tk.html, Abruf: 09.03.2020

The Qt Company

The Qt Company: *Qt | Cross-platform software development for embedded & desktop*. https://www.qt.io/, Abruf: 10.03.2020

The Real Python

The Real Python: *Model-View-Controller (MVC) Explained – With Legos*. https://realpython.com/the-model-view-controller-mvc-paradigm-summarized-with-legos/, Abruf: 10.03.2020

Wikipedia 2020a

Wikipedia: *Entwurfsmuster — Wikipedia, Die freie Enzyklopädie*. https://de.wikipedia.org/w/index.php?title=Entwurfsmuster&oldid=195980935. Version: 2020, Abruf: 11.03.2020

Wikipedia 2020b

Wikipedia: *IEEE 11073 Service-oriented Device Connectivity (SDC) — Wikipedia, Die freie Enzyklopädie*. https://de.wikipedia.org/w/index.php?title=IEEE_11073_Service-oriented_Device_Connectivity_(SDC)&oldid=195539894. Version: 2020, Abruf: 03.03.2020

Wikipedia 2020c

Wikipedia: *OSI-Modell — Wikipedia, Die freie Enzyklopädie*. https://de.wikipedia.org/w/index.php?title=OSI-Modell&oldid=197434019. Version: 2020, Abruf: 05.03.2020

Eidesstattliche Erklärung

Ich, Zadjad Rezai, versichere hiermit, dass ich meine **Dokumentation zur betrieblichen Projektarbeit** mit dem Thema

> *Entwicklung eines Test-Tools für SDC-Netzwerke – Tool zur Unterstützung der ServicetechnikerInnen*

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, wobei ich alle wörtlichen und sinngemäßen Zitate als solche gekennzeichnet habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Lübeck, den 04.05.2020

Zadjad Rezai Digital unterschrieben von Zadjad Rezai
Datum: 2020.05.04 08:51:21 +02'00'

Zadjad Rezai

A Anhang

A.1 Detaillierte Zeitplanung

Analysephase			**8 h**
1. Analyse des Ist-Zustands		2 h	
1.1. Fachgespräch mit den Kunden	1 h		
1.2. Prozessanalyse	1 h		
2. „Make or buy“-Entscheidung und Wirtschaftlichkeitsanalyse		1 h	
3. Erstellen eines „Use-Case“-Diagramms		2 h	
4. Erstellen des Lastenhefts		3 h	
Entwurfsphase			**16 h**
1. Prozessentwurf		2 h	
2. GUI-Entwurf		3 h	
3. Sequenzdiagramm erstellen		2 h	
4. Erstellen eines UML-UseCase Diagramms der Anwendung		4 h	
5. Erstellen des Pflichtenhefts		5 h	
Implementierungsphase			**30 h**
1. Schreiben der Tests (TDD)		7 h	
2. Programmierung der Komponenten für die Funktionen		8 h	
2.1. Ping/Response-Überprüfer	1 h		
2.2. DHCP-Überprüfer	2 h		
2.3. Multicast-Überprüfer	2 h		
2.4. Port-Überprüfer	2 h		
2.5. NTP-Überprüfer	1 h		
3. Umsetzung der Serverseitigen-Anwendung		7 h	
4. Umsetzung der Clientseitigen-Anwendung		5 h	
5. Client- und Serverseitige Anwendung Verbindung		3 h	
Abnahmetest der Fachabteilung			**1 h**
1. Abnahmetest der Fachabteilung		1 h	
Einführungsphase			**1 h**
1. Kurze Einführung/Abnehmerschulung		1 h	
Erstellen der Dokumentation			**9 h**
1. Erstellen der Benutzerdokumentation		2 h	
2. Erstellen der Projektdokumentation		6 h	
3. Programmdokumentation		1 h	
Pufferzeit			**5 h**
1. Puffer		5 h	
Gesamt			**70 h**

A.2 Verwendete Resourcen

A.2.1 Hardware

- Büroarbeitsplatz mit einem Laptop
- Zwei Monitore

A.2.2 Software

- git - Verteilte Versionsverwaltung
- JetBrains PyCharm Professional Edition - Entwicklungsumgebung Python
- Python 3.7.4 - Interpreter für Python
- pytest - Framework zur Durchführung von Unit-Tests
- Flask - Framework zum Erstellen der REST API
- pdoc - Python-Modul zum Erstellen der Code-Dokumentation
- Pyinstaller - Python-Modul zum Erstellen von Windows-Ausführbaren Dateien
- TexLive - Distribution des Testsatzsystems TEX
- TeXstudio - Entwicklungsumgebung LATEX
- Windows 10 Enterprise - Betriebssystem
- PlantUML - Erstellen von UML Diagrammen

A.2.3 Personal

- Entwickler - Umsetzung des Projekts
- Mitarbeiter des BUITS Teams - Festlegung der Anforderungen und Abnahme des Projekts

A.3 Lastenheft (Auszug)

Es folgt ein Auszug aus dem Lastenheft mit Fokus auf die Anforderungen:

Die Anwendung muss folgende Anforderungen erfüllen:

1. Funktionen
   1.1. NTP-Server können abgefragt werden
   1.2. Es kann überprüft werden, ob bestimmte Ports offen sind
   1.3. Es kann überprüft werden, ob Multicast innerhalb des Netzes funktioniert
2. Sonstige Anforderungen
   2.1. Die Anwendung muss ohne das Installieren einer zusätzlichen Software nutzbar sein
   2.2. Die Anwendung soll jederzeit einsetzbar sein.
   2.3. Die Anwendung muss so flexibel sein, dass sie bei Änderungen im Entwicklungsprozess einfach angepasst werden kann.

A.4 Pflichtenheft (Auszug)

Zielbestimmung

Die entstehende Software stellt eine Erleichterung des manuellen Prozesses der Fehlerüberprüfung innerhalb von Netzwerken unserer Kunden dar. Außerdem bietet die Software eine graphische Oberfläche an.

1. Musskriterien
   - Eine GUI wird geboten
   - Die Anwendung ist leicht übertragbar, ggf. über USB
   - Keine Administratorrechte oder externe Programme sollen benötigt werden
   - Folgende Funktionen werden geliefert
     - NTP-Server können abgefragt werden
     - Es kann überprüft werden, ob bestimmte Ports offen sind
     - Es kann überprüft werden, ob Multicast innerhalb des Netzes funktioniert
   - Konfiguration
     - Die Software, welche auf dem Testserver läuft, benötigt keine Nutzereingabe
     - Die GUI-Anwendung, wird komplett über ihre Oberfläche konfiguriert
       * Die Netzwerkschnittstelle kann hier angegeben werden
       * Der Testserver kann hier angegeben werden
       * Der NTP-Server kann hier angegeben werden
   - Ausgabe
     - Log der erhaltenen Daten sind innerhalb der GUI aufrufbar

Wunschkriterien

1. Die Software ist Modular aufgebaut und ermöglicht einfaches ergänzen weiterer Funktionen
2. Die Testserver-Anwendung ist eine ßelbst dokumentierende"[36] Anwendung
3. Die Softwarelösung ist automatisch bereitstellbar

Abgrenzungskriterien

1. Die Software kann selbst keine Fehler beheben

[36] Dokumentation ist direkt im Code und wird dann für den Nutzer aufbereitet

Produkteinsatz

Anwendungsbereiche

Das Produkt wird zur schnelleren Fehlerbehebung genutzt und soll den manuellen Prozess der Fehlersuche innerhalb der komplexen Netzwerke von Dräger-Kunden erleichtern.

Zielgruppe

Servicemitarbeiter, welche beauftragt werden Fehleruntersuchungen bei Dräger-Kunden mit einem SDC-Netzwerk zu leisten.

Betriebsbedingungen

Die CLI-Anwendung wird, sobald sie benötigt wird, auf einem Server innerhalb des Kundennetzwerks gestartet. Die GUI-Anwendung wird auf dem Rechner der Servicetechniker gestartet und kann mit der CLI-Anwendung kommunizieren. Die Kommunikation wird mittels REST gewährleistet.

Produktumgebung

Das Produkt ist weitgehend unabhängig vom Betriebssystem, sofern folgende Produktumgebung vorhanden ist:

1. Software
    - Python Version 3 inkl. der Paketen, welche in requirements.txt genannt werden
2. Hardware
    - Internetfähiger Rechner, der o. g. Software erfüllt
3. Orgware
    - Keine

Produktfunktionen

Die Benutzer können vor Start einer Anfrage an die REST API ihre Konfiguration völlig frei über die graphische Oberfläche anpassen. Die graphische Oberfläche wird nach Start des Testlaufes alle Ergebnisse sammeln und gesammelt anzeigen sowie loggen. Die Logs sind über die graphische Oberfläche erreichbar.

Produktleistungen

Die GUI-Anwendung loggt alle Rückmeldungen der CLI-Anwendung (REST API) und ermöglicht simplen Zugriff auf alle Logs. Außerdem zeigt die GUI-Anwendung die Ergebnisse über sich anpassende Bilder an, weshalb die Logs ausschließlich im Fehlerfall benötigt werden.

Qualitätszielbestimmungen

	Sehr wichtig	Wichtig	Weniger wichtig	Unwichtig
Robustheit	X			
Zuverlässigkeit	X			
Korrektheit	X			
Benutzerfreundlichkeit		X		
Effizienz			X	
Portierbarkeit	X			
Kompatibilität	X			

Abbildung 1: Tabelle der Qualitätsbestimmungen

Globale Testszenarien und Testfälle

Alle Funktionen werden in einer der Testumgebungen, welche den gegebenen Bestimmungen in Kundennetzwerken entsprechen, getestet. Zusätzlich werden Unit Test für die Funktionen der Software geschrieben.

Entwicklungsumgebung

Plattform

- Windows 10
- Bitbucket
- Python 3

Tools

- JetBrains PyCharm

Hardware

- Laptop, Tastatur & Maus, 2 Monitore, Dockingstation für den Laptop

A.5 UseCase Diagramm

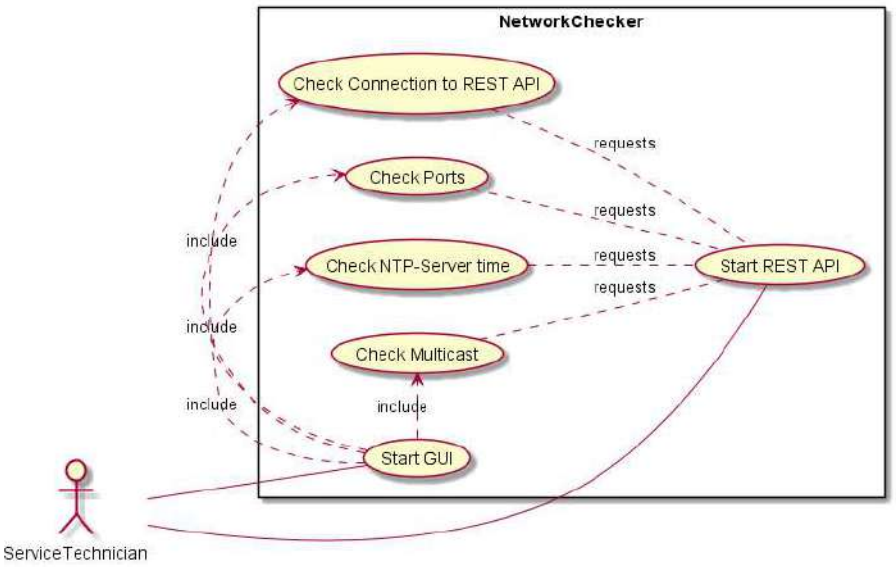

Abbildung 2: UseCase Diagramm

A.6 Sequenzdiagramm

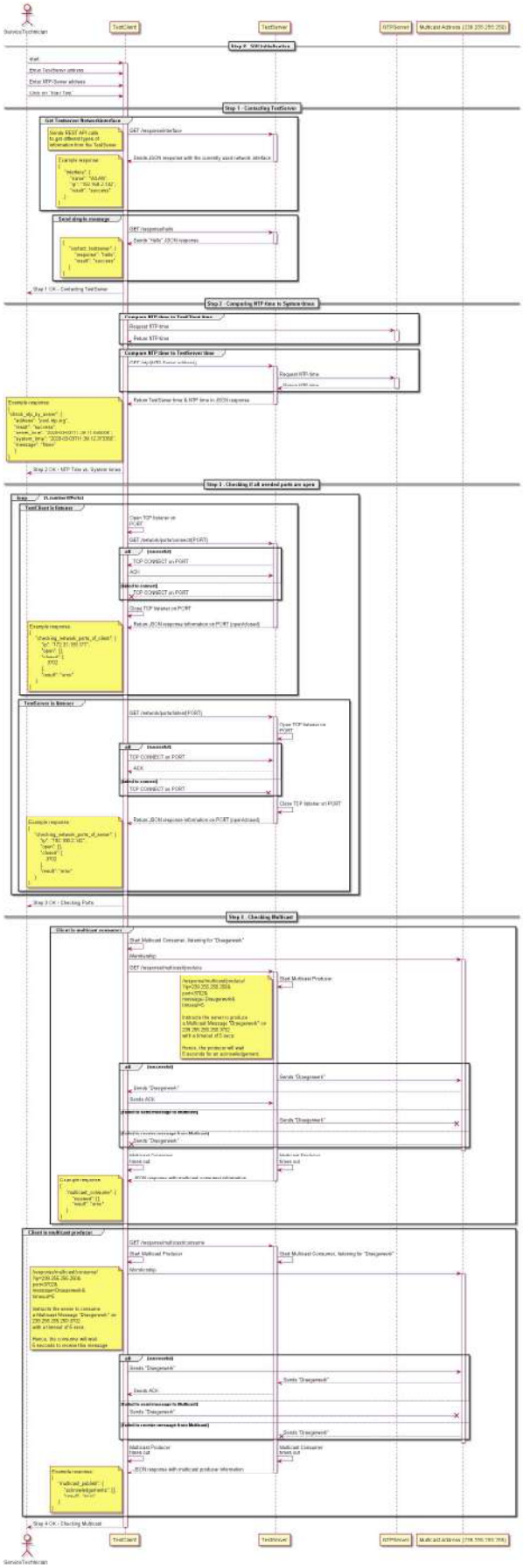

Abbildung 3: Sequenzdiagramm

A.7 Oberflächenentwürfe

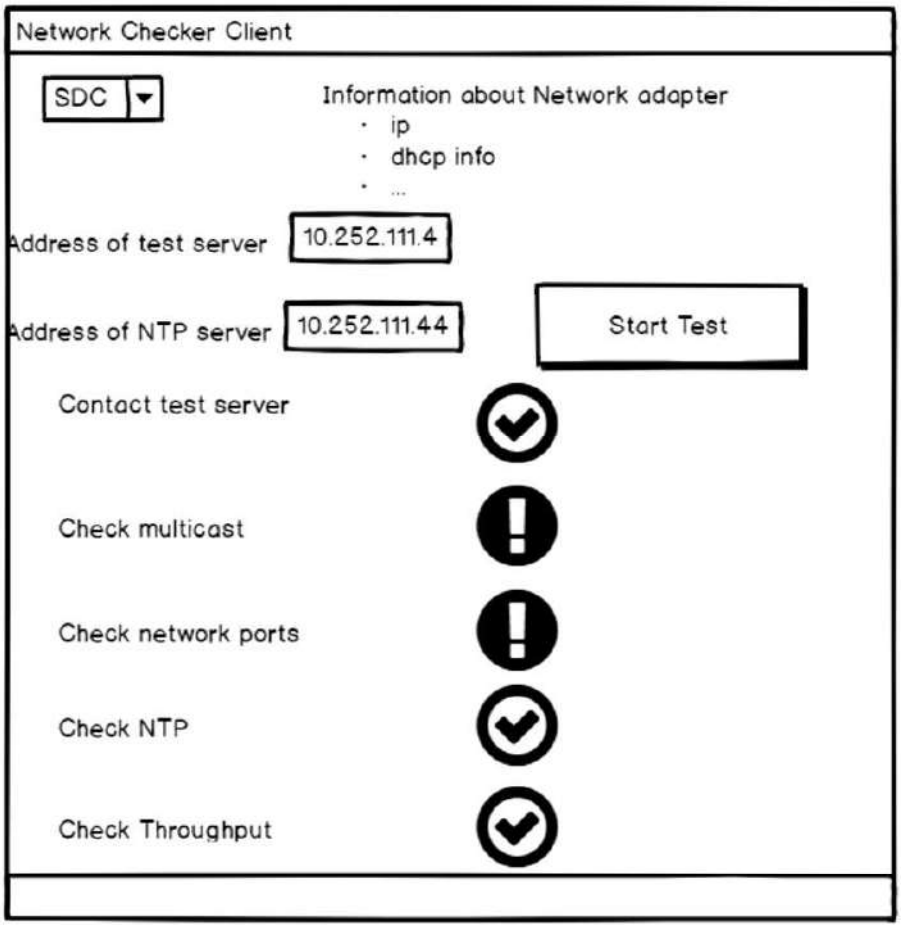

Abbildung 4: Erster Entwurf der Oberfläche

A.8 Screenshots der Anwendung

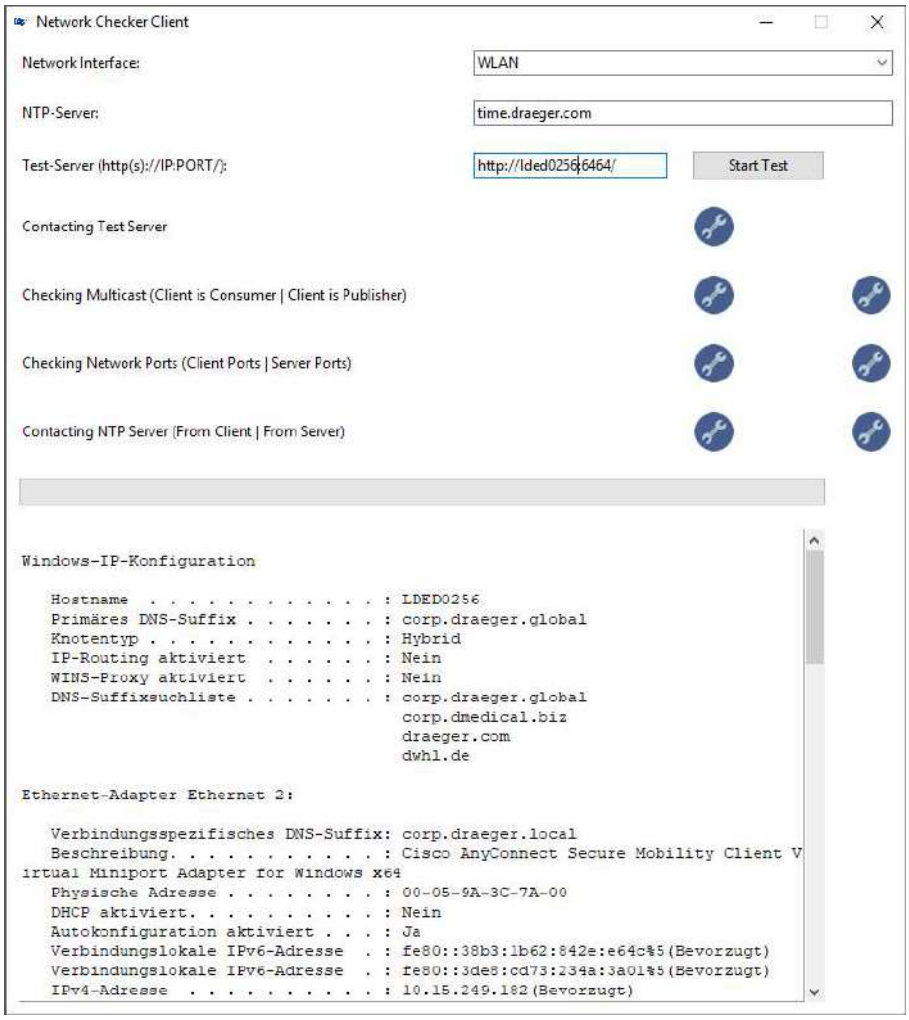

Abbildung 5: Oberfläche vor Start des Testes

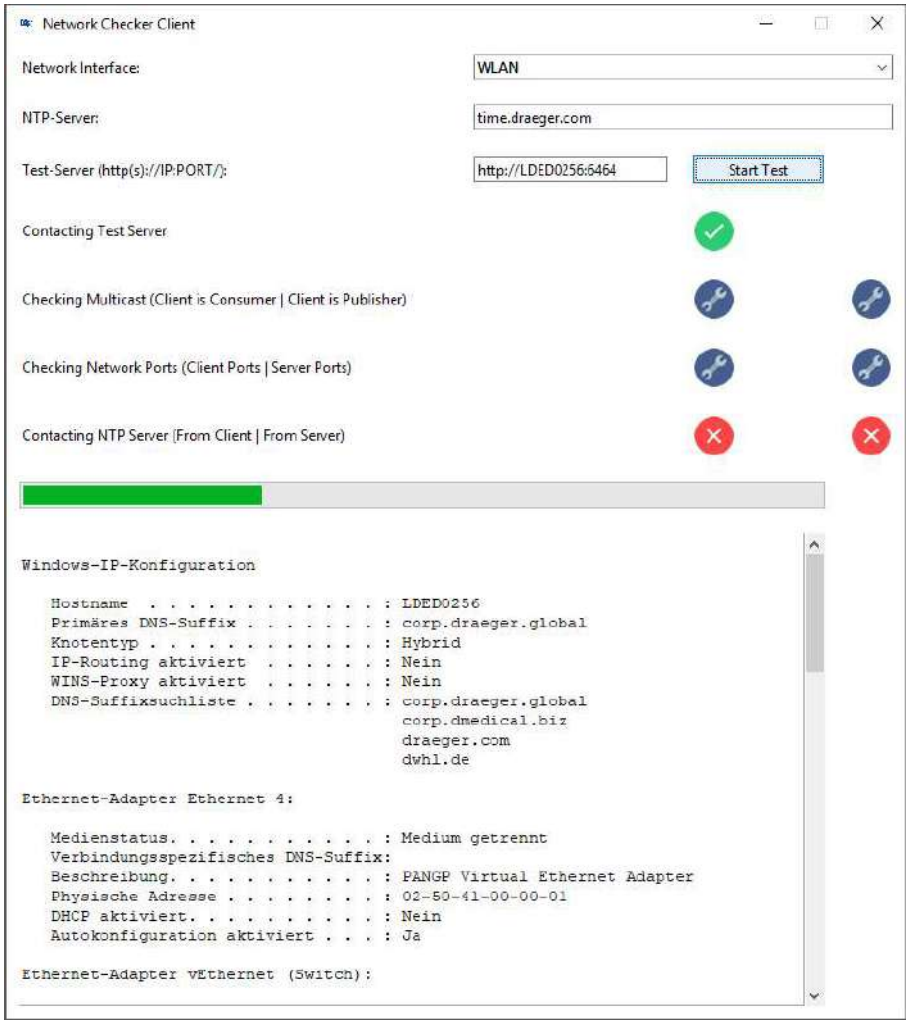

Abbildung 6: Oberfläche während des Tests

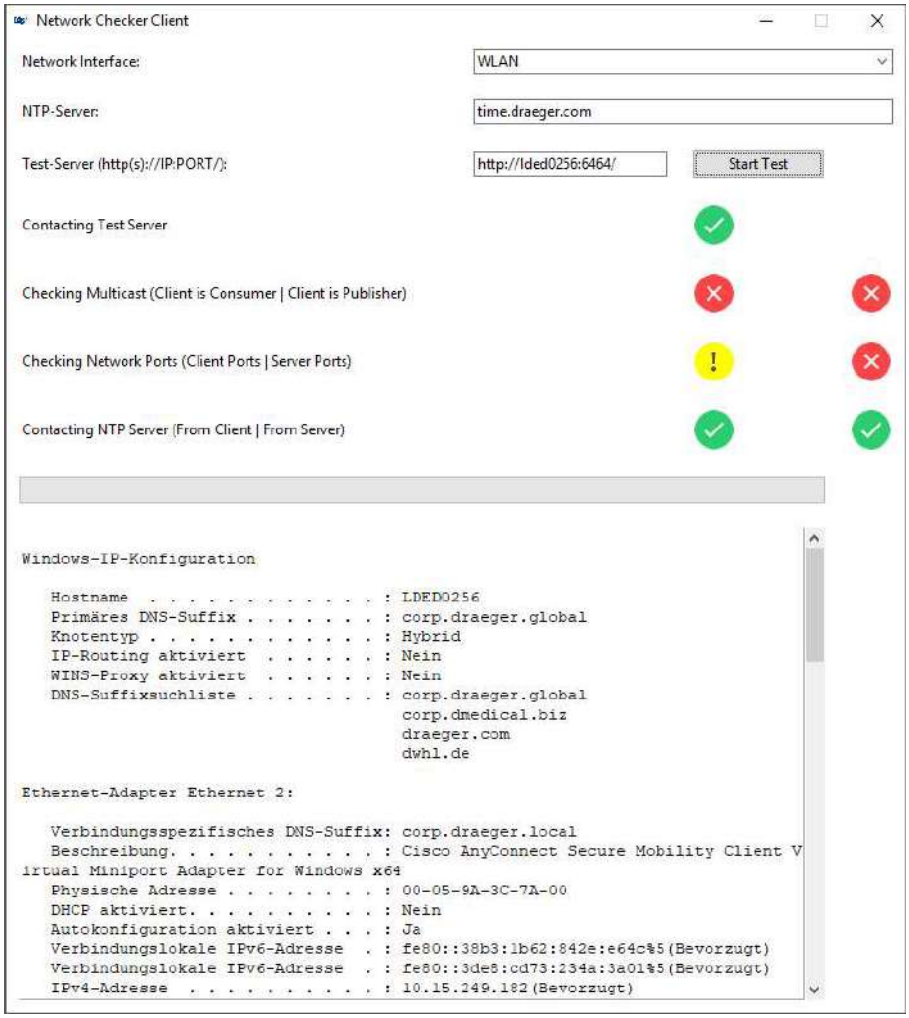

Abbildung 7: Oberfläche nach Beendigung des Tests

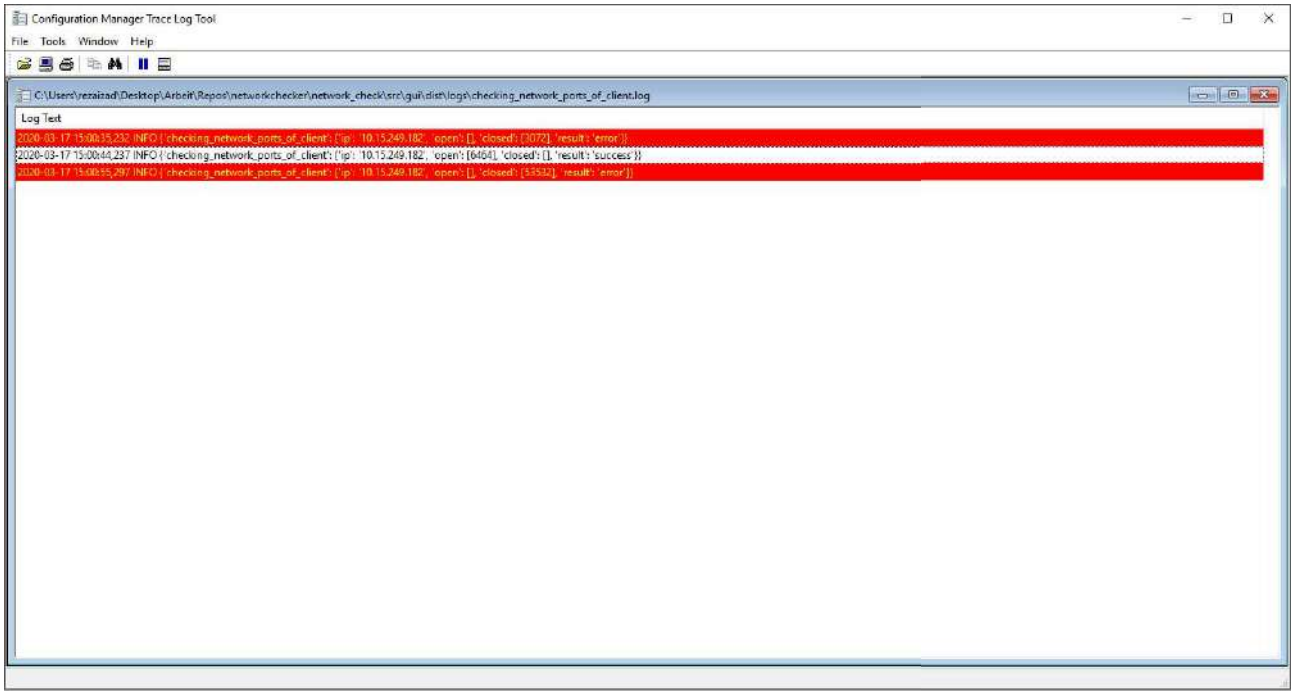

Abbildung 8: Beispiel Log

A.9 Entwicklerdokumentation

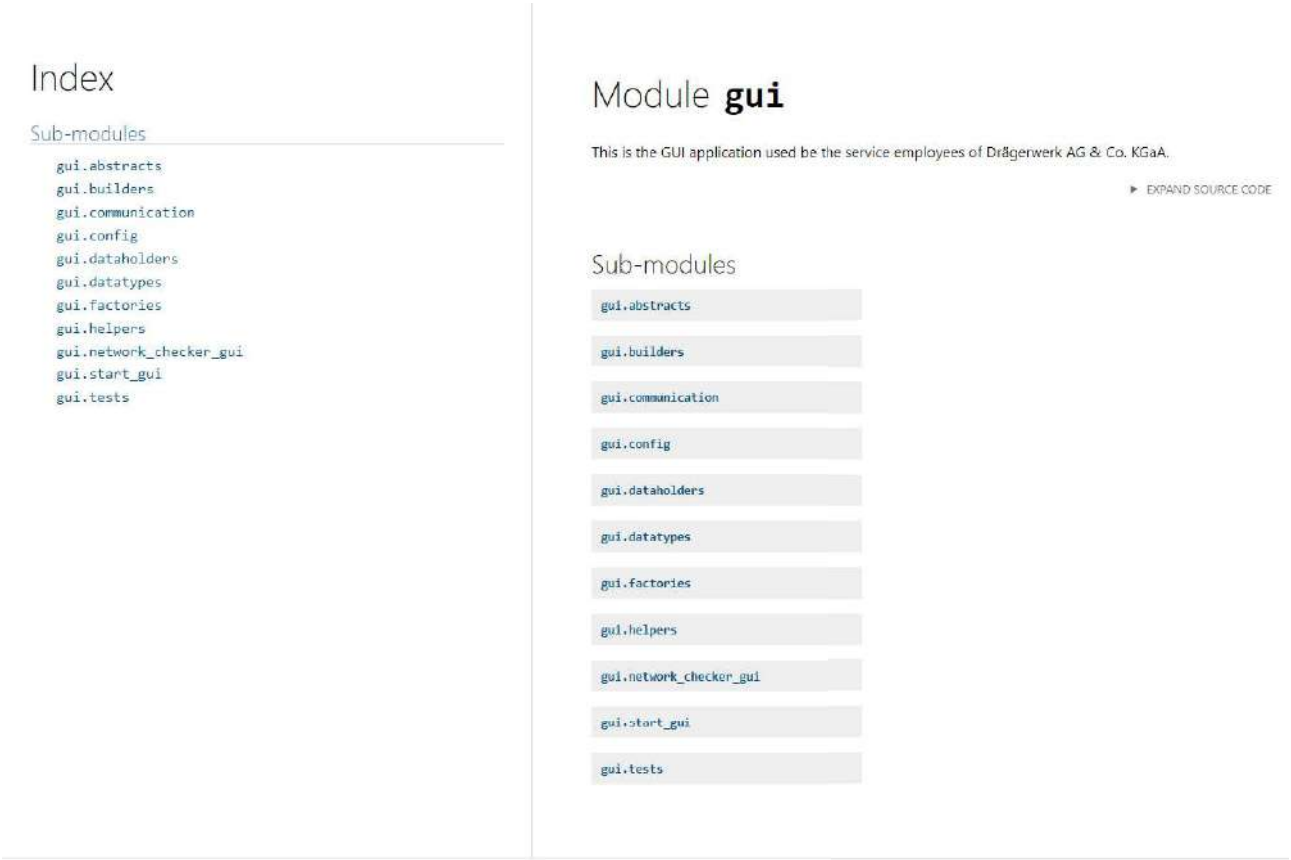

Abbildung 9: Dokumentation des GUI Modules (Graphische Oberfläche)

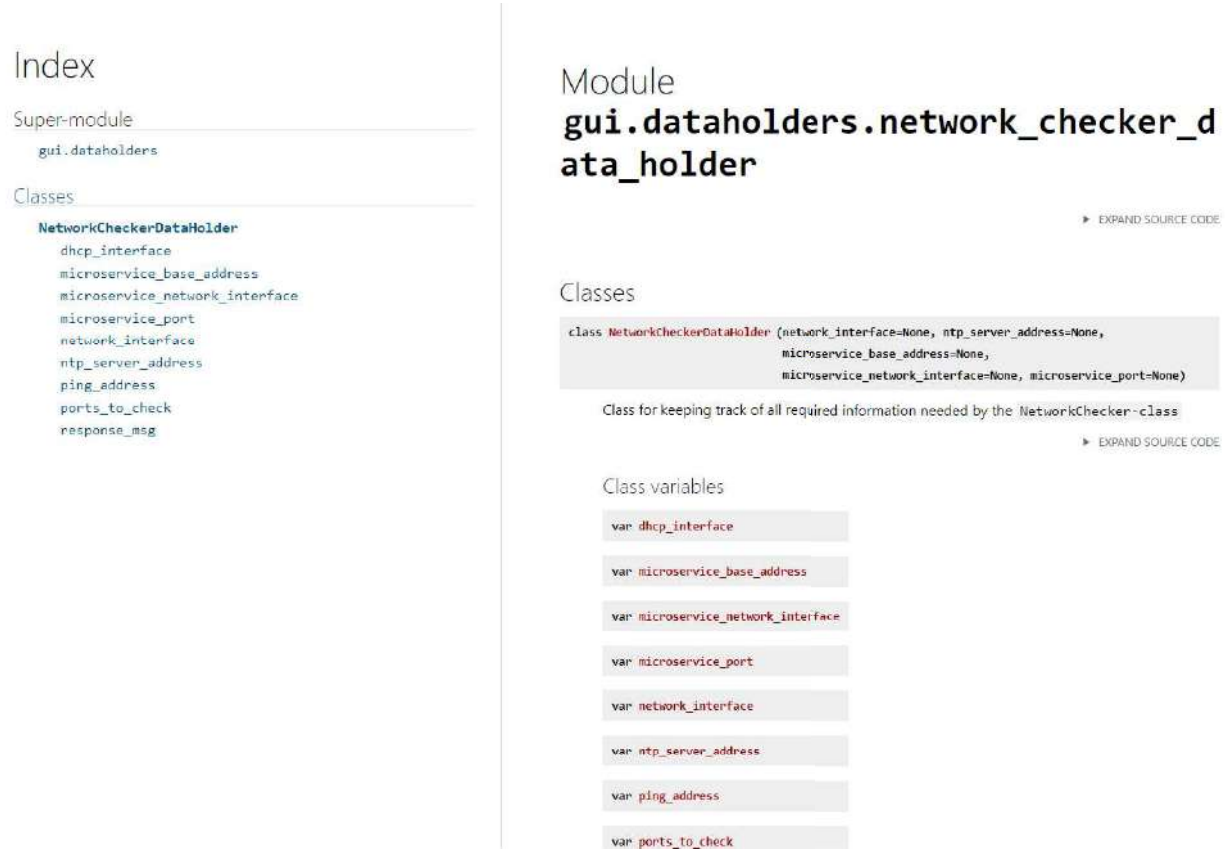

Abbildung 10: Dokumentation einer Datenklasse

```
"""A small module containing a Multicast Consumer
The address range is divided into blocks each assigned a specific purpose or behavior.

    IP multicast address range      Description                 Routable
    224.0.0.0 to 224.0.0.255        Local subnetwork[1]         No
    224.0.1.0 to 224.0.1.255        Internetwork control        Yes
    224.0.2.0 to 224.0.255.255      AD-HOC block 1[2]           Yes
    224.3.0.0 to 224.4.255.255      AD-HOC block 2[3]           Yes
    232.0.0.0 to 232.255.255.255    Source-specific multicast[1] Yes
    233.0.0.0 to 233.251.255.255    GLOP addressing[4]          Yes
    233.252.0.0 to 233.255.255.255  AD-HOC block 3[5]           Yes
    234.0.0.0 to 234.255.255.255    Unicast-prefix-based        Yes
    239.0.0.0 to 239.255.255.255    Administratively scoped[1]  Yes

"""
import ...


class MulticastConsumer:
    def __init__(self, multicast_group: str, port: int, interface: str = '', timeout: float = 1):
        """
        Creates a `MulticastConsumer`, which allows to consume multicast messages

        Parameters
        ----------
        multicast_group :   str
                            The Multicast IP address. E. g. `239.255.255.250`
        port :              int
                            The Multicast Port to use. E. g. `3702`
        interface :         str
                            The name of the interface to create this Consumer Socket on
        timeout :           float
                            Timeout in seconds, defaults to 1 -> 1 sec. timeout.
                            The `MulticastConsumer` will be killed after this timeout
                            has occurred, to ensure we do not block indefinitely when trying
                            to receive data.
        """
```

Abbildung 11: Dokumentation des Multicast Consumers im Code

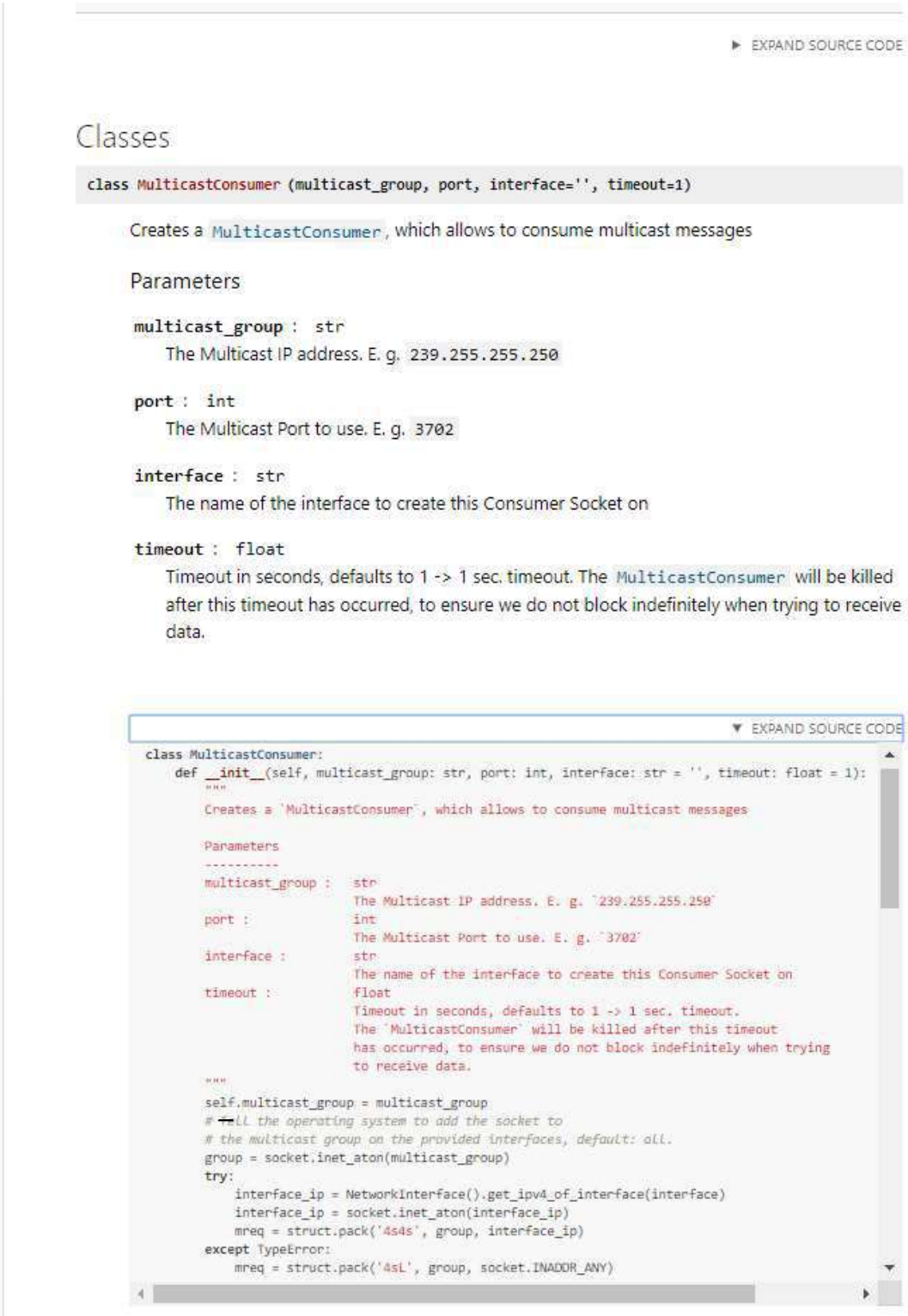

Abbildung 12: Dokumentation des Multicast Consumers mit Codezugriff

A.10 REST API Dokumentation

Abbildung 13: Swagger Dokumentation der REST API

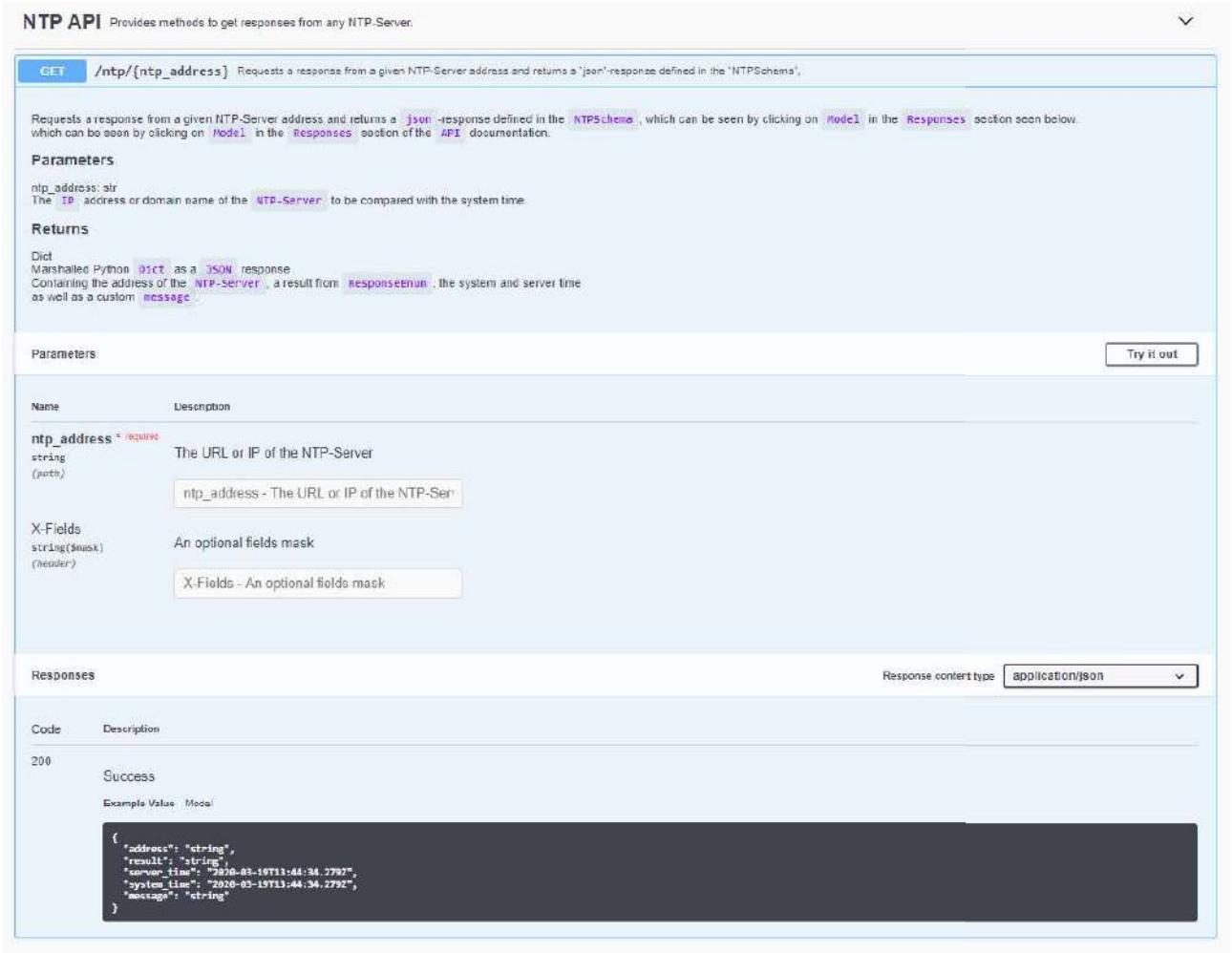

Abbildung 14: Dokumentation der NTP-Abfrage mittels REST API

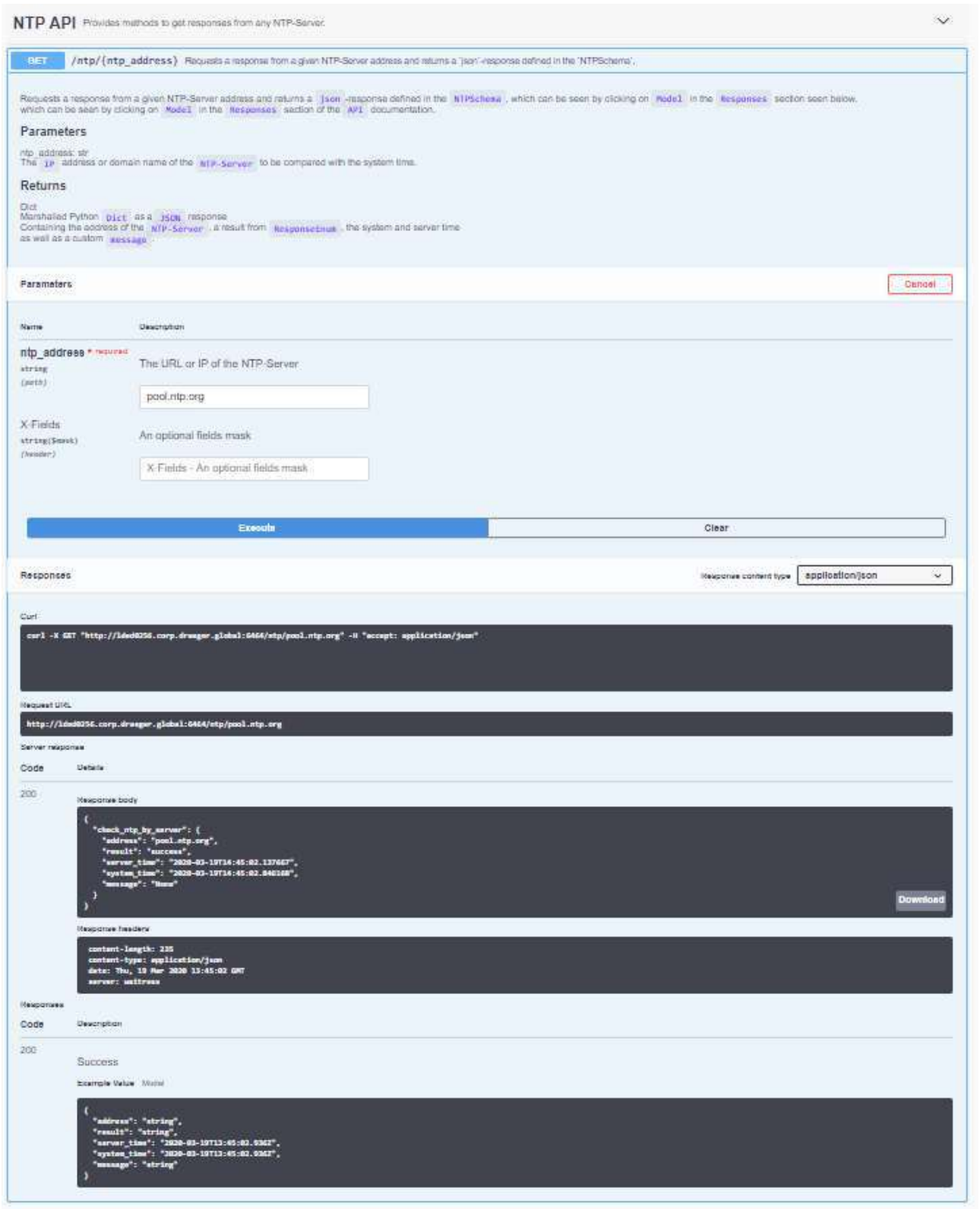

Abbildung 15: Ausführen der NTP-Abfrage innerhalb der Swagger Dokumentation

A.11 Testfall und sein Aufruf in der Entwicklungsumgebung

Der Autor hat die Dokumentationsstrings aus dem Quellcode entfernt, um die Länge möglichst kurz zu halten.

```python
@pytest.fixture(scope='module')
def test_client():
    app = create_app(TestConfig)

    # Flask provides a way to test your application by exposing the Werkzeug test Client
    # and handling the context locals for you.
    testing_client = app.test_client()

    # Establish an application context before running the tests.
    ctx = app.app_context()
    ctx.push()

    yield testing_client  # this is where the testing happens!

    ctx.pop()
```

Listing 1: Fixture zum Erstellen eines Flask-Testclients

```python
class TestCheckHello:
    def test_no_input___returns_hello(self, test_client):
        response = test_client.get('response/hello')
        assert response.status_code == 200
        assert response.json['contact_testserver'] == {'response': 'hello', 'result': ResponseEnum.success.value}

    @pytest.mark.parametrize('userinput', [
        'a-simple-test-string', 'other-string', '222', '22.341'
    ])
    def test_input___returns_input(self, test_client, userinput: str):
        response = test_client.get(f'response/hello/{userinput}')
        assert response.status_code == 200
        assert response.json['contact_testserver'] == {
            'response': userinput, 'result': ResponseEnum.success.value}

    def test_wrong_route___returns_error(self, test_client):
        response = test_client.get('response/hello/sub/test')
        assert response.status_code == 404
```

Listing 2: Testklasse, um die "HelloFunktion der API zu testen

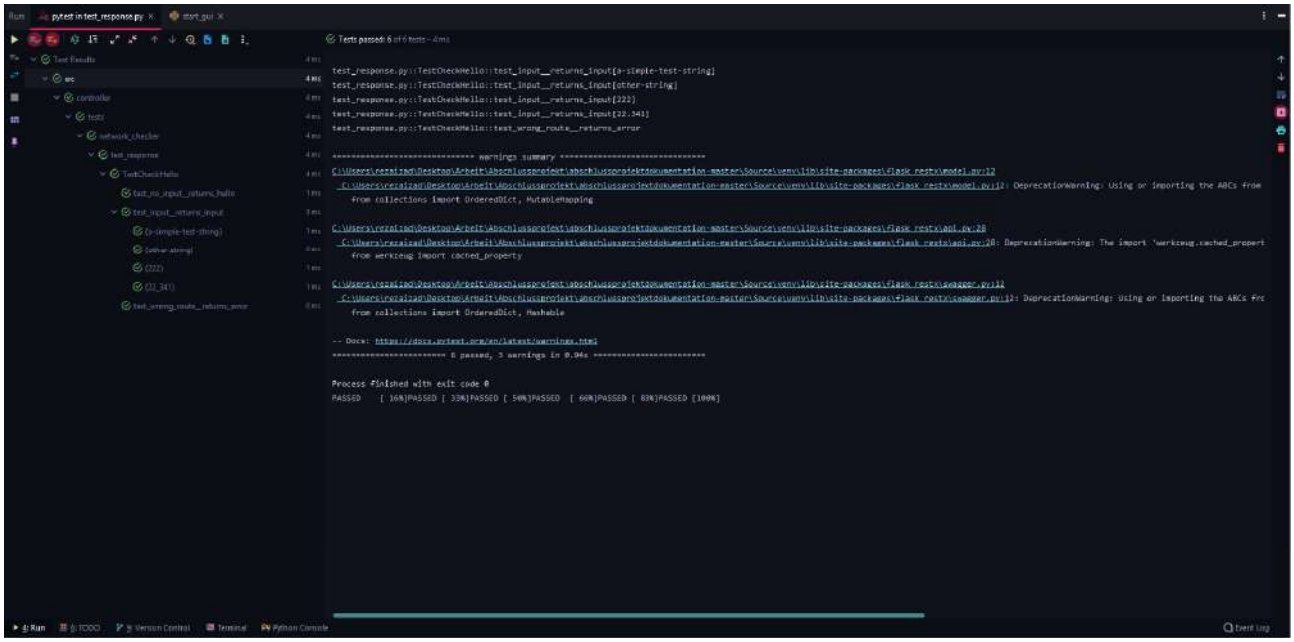

Abbildung 16: Aufruf des Testfalls

A.12 CheckHello-Klasse inkl. Namespace Definition

```
response_namespace = Namespace('Response API', description='Different methods to get a response from the server.')

hello_schema = response_namespace.model('HelloSchema', {
    'response': fields.String(description='The response, which should be provided back by the server.'),
    'result': fields.String(description=f'{ResponseEnum.success.value} -> Response was sent. (Always)'),
})


@response_namespace.route('hello', defaults={'response': 'hello'})
@response_namespace.route('hello/<response>')
class CheckHello(Resource):
    @response_namespace.doc('get_simple_response', model=hello_schema,
                            description='Returns a response from the test-server. Defaults to *hello*, if no response '
                                        'is defined.',
                            params={'response': 'The response, which should be provided back by the server. '
                                                'Defaults to *hello*.'})
    @response_namespace.marshal_with(hello_schema, envelope='contact_testserver')
    def get(self, response: str):
        response = {
            'response': response,
            'result': ResponseEnum.success.value
        }
        loggers['contact_testserver'].info(response)
        return response
```

Listing 3: CheckHello-Klasse inkl. Namespace Definition

A.13 NetworkInterface-Klasse

Die Kommentare wurden entfernt, damit weniger Platz eingenommen wird.

```
class NetworkInterface:
    class ___NetworkInterface:
        def ___init___(self):
            self.response, self.interface_info = subprocess_call(command)
            self.windows_if_list = get_windows_if_list()
            self.interface_name_list = [interface['name'] for interface in self.windows_if_list]
            self.working_interface = None

        def get_interface(self, interface_name: str) -> Dict:
            try:
                return self.windows_if_list[interface_name]
            except KeyError:
                logger.error(f"Unknown Interface {interface_name}")
                return {}

        def get_ips_of_interface(self, interface_name: str) -> Union[list, None]:
            for interface in self.windows_if_list:
                if interface['name'] == interface_name:
                    return interface['ips']

        def get_ipv4_of_interface(self, interface_name: str) -> Union[str, None]:
            ips = self.get_ips_of_interface(interface_name)
            if ips is not None and len(ips) > 1:  # first entries are IPv6, last entry is IPv4
                return ips[-1]
            return None

        def get_ipv6_of_interface(self, interface_name: str) -> Union[str, None]:
            try:
                return self.get_ips_of_interface(interface_name)[0]
            except IndexError:
                return None

        def get_working_interface(self):
            self.working_interface = get_working_if()
            return self.working_interface

    instance = None

    def ___new___(cls):
        if NetworkInterface.instance is None:
            NetworkInterface.instance = NetworkInterface.___NetworkInterface()
        return NetworkInterface.instance

    def ___getattr___(self, name):
        return getattr(self.instance, name)

    def ___setattr___(self, name, **kwargs):
```

```
48        return setattr(self.instance, name, **kwargs)
```

Listing 4: NetworkInterface-Klasse

A.14 Klassendiagramm

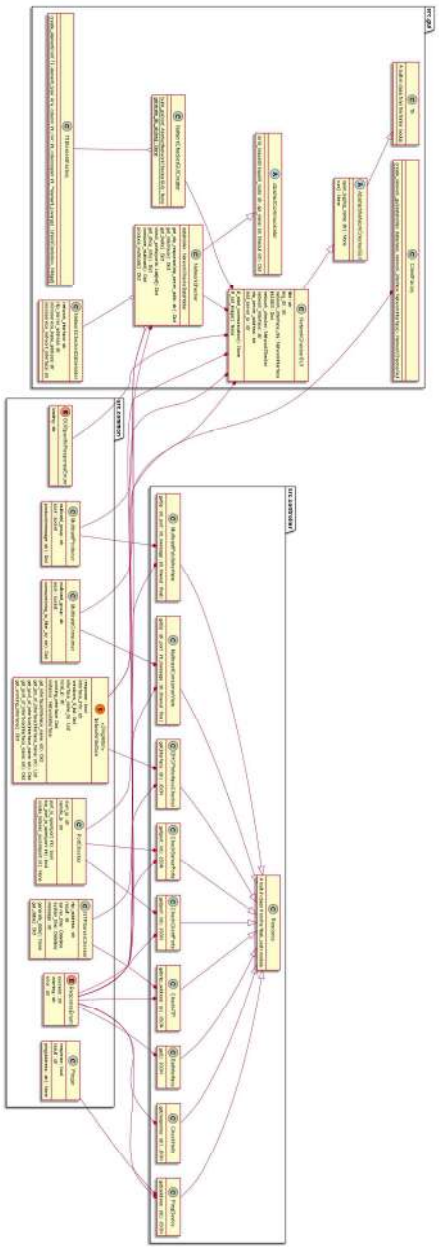

Abbildung 17: Klassendiagramm der Gesamtanwendung

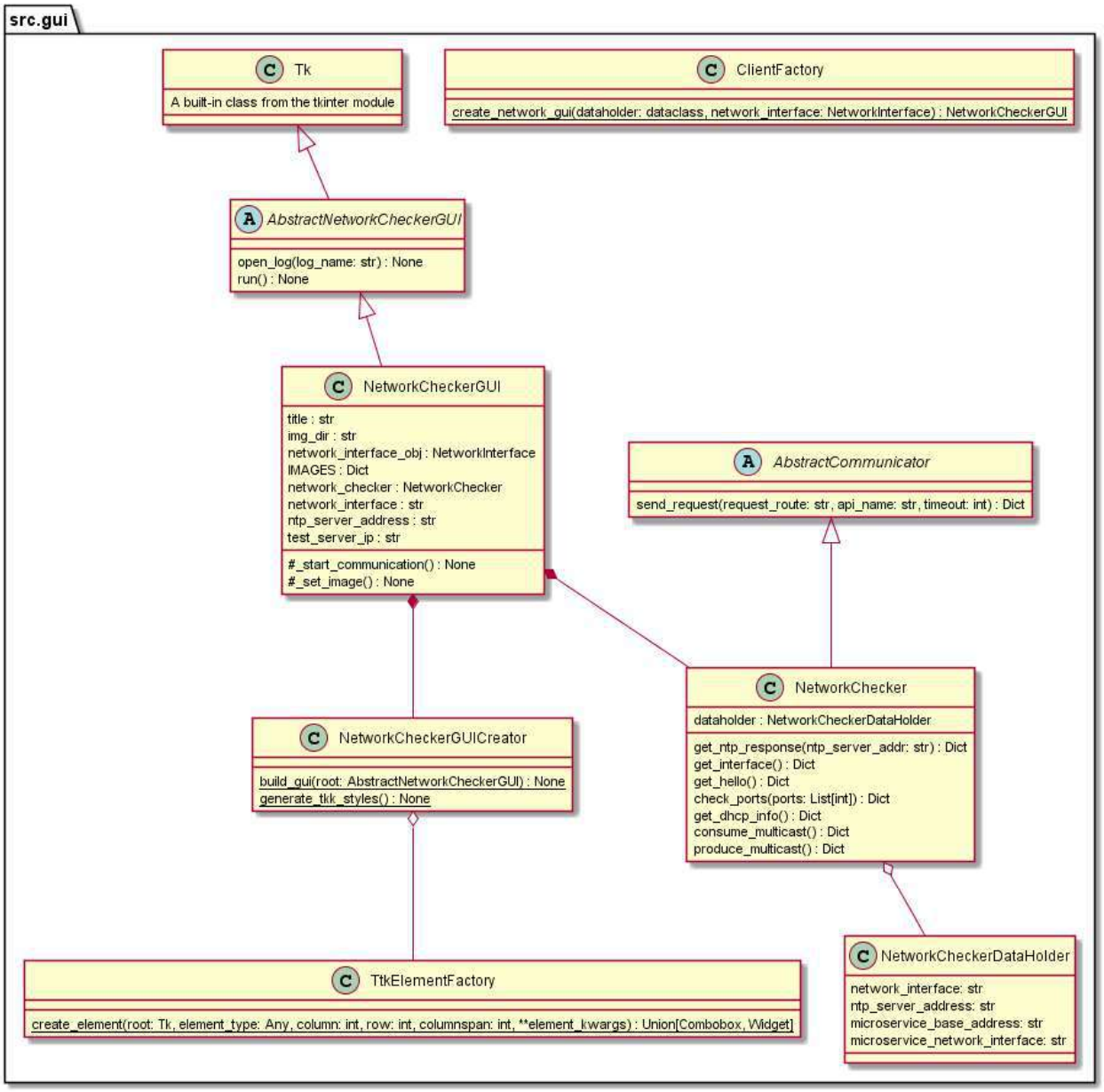

Abbildung 18: Klassendiagramm der GUI

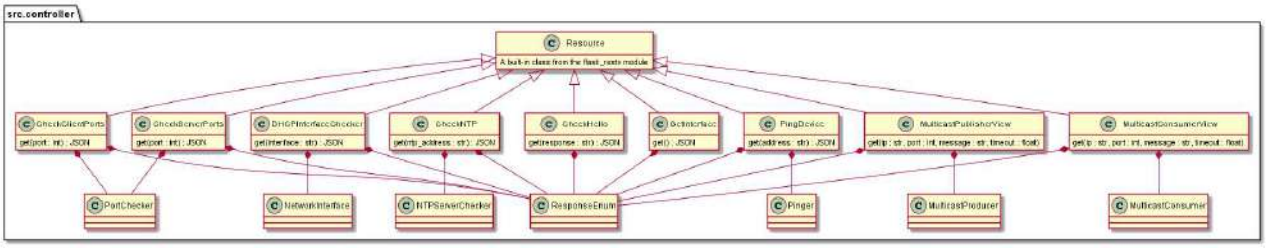

Abbildung 19: Klassendiagramm der REST API

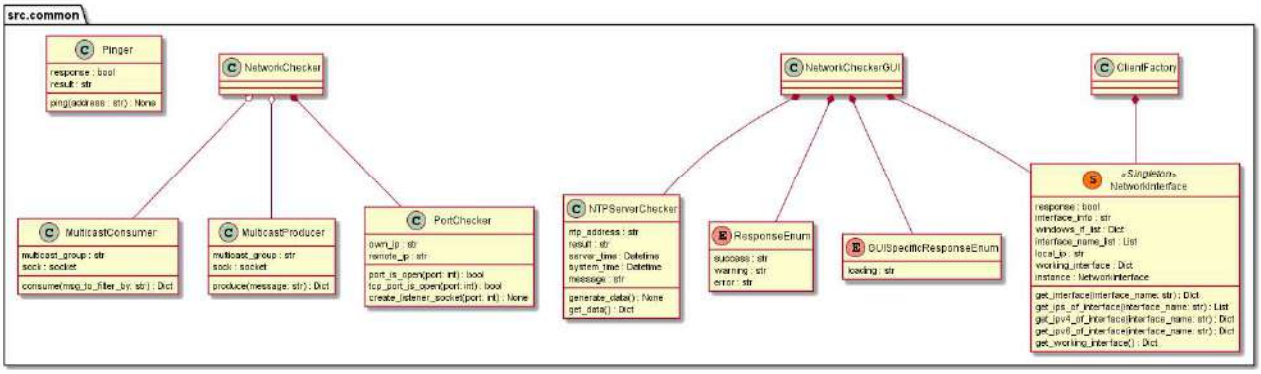

Abbildung 20: Klassendiagramm der geteilten Klassen

A.15 Benutzerdokumentation

Ausschnitt aus der Benutzerdokumentation:

Usage

Testserver (API)

The `Testserver` can be started via its executable `NetworkCheckerAPI.exe`.

To start it, you will have to provide the network interface to use and optionally the port to use.

```
NetworkCheckerAPI.exe -h
usage: NetworkCheckerAPI.exe [-h] [-p PORT] interface

Serves the NetworkChecker API

positional arguments:
interface             The Interface to use.

optional arguments:
-h, --help            show this help message and exit
-p PORT, --port PORT  The port to assign the NetworkChecker API to.

Provided by Drägerwerk AG & Co. KGaA
```

Example usages:

```
> NetworkCheckerAPI.exe Ethernet --port 9999
> NetworkCheckerAPI.exe WLAN --port 1234
> NetworkCheckerAPI.exe CORP    (will use port 6464)
```

For further information, you can check the APIs own Readme `./src/controller/README.md`

Abbildung 21: Anleitung zum Starten der REST API

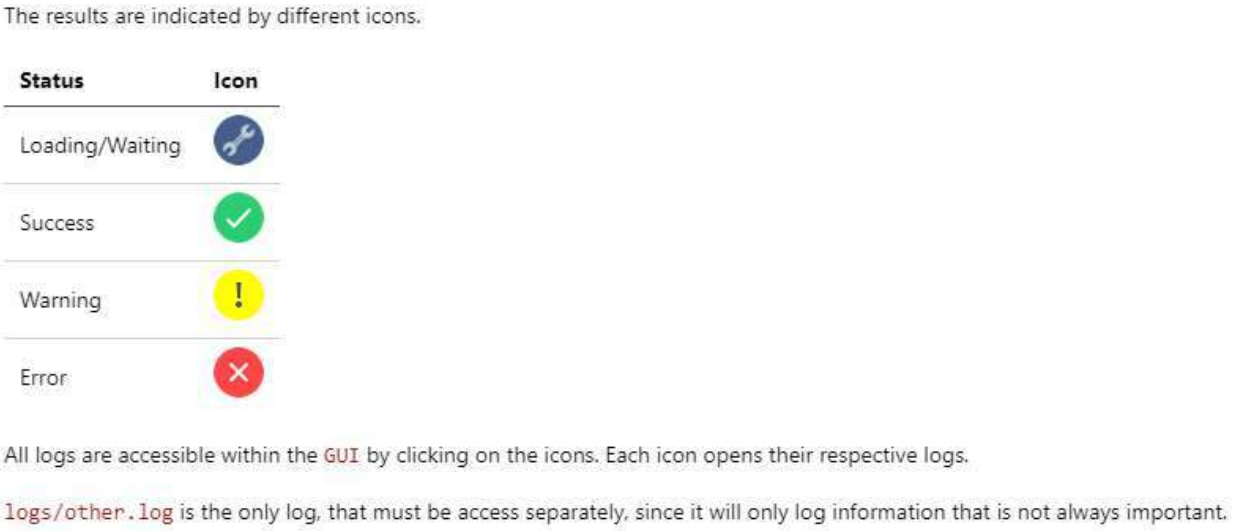

Abbildung 22: Mögliche Ergebnisse