

Prüfungsteil A

Prüfling (private Anschrift):

Ausbildungsbetrieb:

Bestätigung über durchgeführte Projektarbeit

diese Bestätigung ist mit der Projektdokumentation einzureichen

Ausbildungsberuf (bitte unbedingt angeben):

Projektbezeichnung:

Projektbeginn: _____ Projektfertigstellung: _____ Zeitaufwand in Std.: _____

Bestätigung der Ausbildungsfirma:

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

vom: _____ bis: _____ selbständig ausgeführt hat.

Projektverantwortliche(r) in der Firma:

Vorname	Name	Telefon	Unterschrift
---------	------	---------	--------------

Ausbildungsverantwortliche(r) in der Firma:

Vorname	Name	Telefon	Unterschrift
---------	------	---------	--------------

Eidesstattliche Erklärung:

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

Ort und Datum: _____ Unterschrift des Prüflings: _____



Niederrheinische Industrie-
und Handelskammer
Duisburg · Wesel · Kleve

Abschlussprüfung Sommer 2021

Fachinformatiker für Anwendungsentwicklung
Dokumentation zur betrieblichen Projektarbeit

Entwicklung von Versionskontrolle

Neuentwicklung einer Server-Client Anwendung, zur automatischen
Konfiguration und Installation von Software auf der Basis von Asp .Net 5.0

Abgabetermin: Emmerich am Rhein, den 15.04.2021

Prüfungsbewerber:

Dennis van Elk
Seminarstraße 4
46446 Emmerich am Rhein



Ausbildungsbetrieb:

PROBAT-WERKE EMMERICH
von Gimborn Maschinenfabrik GmbH
Reeserstraße 93
46446 Emmerich am Rhein

Dieses Werk einschließlich seiner Teile ist **urheberrechtlich geschützt**. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtgesetzes ist ohne Zustimmung des Autors unzulässig und strafbar. Das gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen sowie die Einspeicherung und Verarbeitung in elektronischen Systemen.

Inhaltsverzeichnis

Abbildungsverzeichnis	III
Tabellenverzeichnis	IV
Abkürzungsverzeichnis	V
1 Einleitung	1
1.1 Projektumfeld	1
1.2 Projektbeschreibung	1
1.3 Projektziel	1
1.4 Projektbegründung	2
1.5 Projektschnittstellen	2
1.6 Projektabgrenzung	2
2 Projektplanung	3
2.1 Projektphasen	3
2.2 Abweichungen vom Projektantrag	3
2.3 Ressourcenplanung	3
2.4 Entwicklungsprozess	4
3 Analysephase	5
3.1 Ist-Analyse	5
3.2 Wirtschaftlichkeitsanalyse	6
3.2.1 „Make or Buy“-Entscheidung	6
3.2.2 Projektkosten	7
3.2.3 Amortisationsdauer	7
3.3 Nicht-monetäre Vorteile	8
3.4 Anwendungsfälle	8
3.5 Lastenheft/Fachkonzept	9
4 Entwurfsphase	9
4.1 Auswahl der Zielplattformen	9
4.1.1 Auswahl der Programmiersprache	9
4.1.2 Genutzte Zielplattformen	9
4.2 Projekttypen	10
4.3 Architekturdesign	10
4.4 Entwurf der Benutzeroberfläche	10
4.5 Datenmodell	11
4.6 Geschäftslogik	12
4.7 Maßnahmen zur Qualitätssicherung	12
4.8 Pflichtenheft/Datenverarbeitungskonzept	12

5	Durchführungssphase	12
5.1	Implementierung der Datenstrukturen	12
5.2	Implementierung der Benutzeroberfläche	13
5.3	Implementierung der Geschäftslogik	13
6	Abnahme und Einführungsphase	14
6.1	Abnahmephase	14
6.2	Einführungsphase	14
6.3	Dokumentation	15
7	Fazit	15
7.1	Soll-/Ist-Vergleich	15
7.2	Gewonnenes Wissen	15
7.3	Ausblick	16
	Literaturverzeichnis	17
	Eidesstattliche Erklärung	18
A	Anhang	i
A.1	Detaillierte Zeitplanung	i
A.2	Verwendete Ressourcen	ii
A.3	Umfrage zur Erfassung der benötigten Features	iii
A.4	Lastenheft (Auszug)	iv
A.5	Konstruktor von MainWindow aus IT-Helper	v
A.6	Microservice Architektur	viii
A.7	Use-Case-Diagramm	ix
A.8	Pflichtenheft (Auszug)	x
A.9	Design Mockup	xi
A.10	Screenshots der Anwendung	xi
A.11	Datenbankmodell	xiii
A.12	Datenmodell Visualisierung	xv
A.13	Klassendiagramm	xvi

Abbildungsverzeichnis

1	Vereinfachtes ER-Modell	11
2	User Interface Mockup	xi
3	Screenshot #1	xi
4	Screenshot #2	xii
5	Datenbankmodell	xiii
6	Datenvisualisierung	xv
7	Klassendiagramm	xvii

Tabellenverzeichnis

1	Zeitplanung	3
2	Auflistung Stunden	7
3	Kostenaufstellung	7
4	Soll-/Ist-Vergleich	15

Abkürzungsverzeichnis

AD	Active Directory
AMS	das von Probat genutzte ERP System
CRUD	Create, Read, Update und Delete
DLL	Dynamic Link Library
DRY	Don't Repeat Yourself
EFCore	Entity Framework Core
ERD	Entity-Relationship Diagram
ERM	Entity-Relationship-Modell
IDE	Integrated Development Environment
KISS	Keep It Simple Stupid
MVC	Model View Controller
MVVM	Model View View Model
NAS	Network Attached Storage
ORDBMS	Objektrelationales Datenbankmanagementsystem
ORM	Object-Relational Mapping
UI	User Interface

1 Einleitung

Diese Projektdokumentation beschreibt den Ablauf des Abschlussprojektes, der IHK-Ausbildung zum Fachinformatiker für Anwendungsentwicklung vom Autor, Dennis van Elk. Der Ausbildende Betrieb sind die PROBAT WERKE von Gimborn Maschinenfabrik GmbH, im folgenden Text Probat genannt.

1.1 Projektumfeld

Der Auftraggeber des Projektes ist die ESA-IT Abteilung von Probat. Probat ist ein international agierender Betrieb, welcher Anlagen- und Röstersysteme individuell für seine Kunden herstellt. Diese Systeme werden von Workstations kontrolliert, welche von der ESA-IT eingerichtet, getestet und verschickt werden. Zusätzlich ist die ESA-IT verantwortlich für die Einrichtung und Wartung von Notebooks, welche beim Kunden vor Ort, von Servicetechnikern zum einrichten, reparieren und warten von Probat Systemen genutzt werden. Auf diesen Notebooks, auch Servicetechniker Notebooks genannt, wird spezielle Software von Rockwell und Siemens zur Anlagensteuerung installiert.

1.2 Projektbeschreibung

Das Projekt befasst sich mit der Entwicklung einer Software, welche automatisiert Setupdateien, Skripte und weiteres in einer vordefinierten Reihenfolge mit Startargumenten ausführen kann. Die Anwendung heißt Versionskontrolle und soll auf einer schon existierenden Software namens IT-Helper basieren. Versionskontrolle wird ausschließlich von der ESA-IT Abteilung eingesetzt und dient der Automatisierung, sowie der Qualitätskontrolle zur Einrichtung von Workstations und Servicetechniker Notebooks. Hierbei soll darauf geachtet werden, dass die Software sich ohne Anpassungen in eine Microservice Architektur implementieren lässt. Bei der Ausführung muss zusätzlich beachtet werden, dass die Programme in einer bestimmten Reihenfolge und mit jeweils unterschiedlichen Startargumenten ausgeführt werden.

1.3 Projektziel

Das Ziel des Projektes war es, die Mitarbeiter der ESA-IT mit Hilfe von Versionskontrolle zu entlasten. Dies sollte durch zwei Unterziele verwirklicht werden. Das erste Ziel war es, die Anwendung zuverlässig und mit minimalen Nutzer eingaben zu gestalten. Das zweite Ziel war die Konfiguration, sowie das Warten und Instandhalten der Software so einfach und aufwandsarm wie möglich zu gestalten.

1.4 Projektbegründung

Versionskontrolle wurde in Auftrag gegeben, um den Aufwand den das einrichten von Windows Maschinen verursacht zu minimieren. Durch die Automatisierung, soll die Software den Mitarbeitern bis zu drei Arbeitsstunden pro Maschine sparen. Zusätzlich dazu ist Versionskontrolle zuverlässiger, da es den Installationsprozess und die Einrichtung jedes Programms überwacht und den Status an den Mitarbeiter weitergibt.

1.5 Projektschnittstellen

Die Anwendung greift auf ein Probat internes Active Directory über das LDAP Protokoll zu, um Nutzer der Anwendung zu Authentifizieren, ohne die dazugehörigen Passwörter speichern zu müssen.

Die Client Anwendung von Versionskontrolle wurde als eine „self-packaged executable“ Datei bereitgestellt, welche auf einem externen Laufwerk abgelegt und ausgeführt wird. Erfasste Daten werden von dem Client an das Backend von Versionskontrolle geschickt und dort in einer Postgres SQL Datenbank abgespeichert. Der Client nimmt außerdem auch Setup Dateien an und speichert diese, auf das von der Abteilung bereitgestellte NAS¹, ab.

Das Backend wird durch mehrere Docker Container über den von Probat gehosteten Docker Stack bereitgestellt.

Die Nutzer, wurden außerdem zusätzlich geschult, um die Software besser nutzen zu können. Diese Mitarbeiter entscheiden auch nach der Testphase, ob das Projekt als gelungen gewertet wird, oder ob noch Nachbesserungen ausstehen.

1.6 Projektabgrenzung

Dieses Projekt ist eigenständig und besitzt keine Abhängigkeiten oder Anforderungen gegenüber zu anderen Projekten. Zusätzlich sollte klar gestellt sein, dass sich das Projekt ausschließlich mit der Entwicklung und Verteilung der Software befasst, jedoch aber nicht mit der Einrichtung der Hardware, welche von der ESA-IT übernommen wurde.

¹Network Attached Storage (NAS)

2 Projektplanung

2.1 Projektphasen

Für die Umsetzung dieses Projekts, standen dem Autor 70 Stunden zur Verfügung. Dieser Zeitraum wurde in einzelne Phasen unterteilt, die bereits im Projektantrag festgelegt wurden. Eine grobe Unterteilung kann der Folgenden Tabelle entnommen werden.

Projektphase	Geplante Zeit
Projektanalyse	7 h
Projektentwurf	12 h
Projektdurchführung	33 h
Projektverteilung	3 h
Projektabschluss	12 h
Puffer	3 h
Gesamt	70 h

Tabelle 1: Zeitplanung

Eine detailliertere Zeitplanung findet sich im Anhang A.1: Detaillierte Zeitplanung auf Seite i.

2.2 Abweichungen vom Projektantrag

Die Zeitplanung weicht stark von der ursprünglichen Zeitplanung aus dem Projektantrag ab. Der Grund dafür ist ein Wechsel in der Projektplanung. Ursprünglich war es gedacht, den Entwurfs und den Durchführungszeitraum ineinander überlaufen zu lassen, um eine dynamischere Entwicklung zu fördern. Dies könnte aber zu Designfehlern führen, welche viel Zeit kosten können. Deshalb hat sich der Autor, um im Zeitrahmen zu bleiben, entschieden und den Zeitraum für die Entwurfsphase verlängert und für die Durchführung verkürzt. Außerdem wurde die Planungsphase in die Analysephase umbenannt, da dies besser zu den Tätigkeiten, welche in dieser Phase ausgeführt werden, passt. Es wurde auch Zeit aus der Projektverteilung und des Abschlusses des Projektes genommen, da dort manche der anfallenden Arbeiten auf Mitarbeiter verteilt wurden. Eine zeitliche Aufteilung, wie viel Zeit welcher Mitarbeiter an welcher Phase beteiligt war, kann der Tabelle 2, im späteren Kapitel Projektkosten 3.2.2 entnommen werden.

2.3 Ressourcenplanung

Bei der Auswahl der Ressourcen, wurde darauf geachtet so kostensparend wie möglich zu sein, da dieses Projekt keine hohe Relevanz für die Herstellung von Röstmaschinen oder Anlagensystemen hat. Um Kosten zu sparen, wurde nur Software und Hardware genutzt, die Probat entweder schon

besaß oder frei verfügbar (z. B. Open Source) war. Eine genaue Auflistung der verwendeten Ressourcen sind in der Übersicht, im Anhang A.2: Verwendete Ressourcen auf Seite ii, zu finden

2.4 Entwicklungsprozess

Da die geforderten Funktionen von Versionskontrolle während der Entwicklung stark variieren können, hat sich der Autor für die Kanban Methode entschieden. Diese Methode ermöglicht es, Aufgaben dynamisch zuzuweisen. Dadurch kann man weniger komplexe Aufgaben den Entwicklern zuweisen, welche noch nicht so viel Erfahrung haben und die komplexen Aufgaben den Entwicklern zuweisen, welche die meiste Erfahrung haben. Zusätzlich dazu, ermöglicht Kanban das Priorisieren von Aufgaben, sowie eine klare Abgrenzung dieser. Das kommt der Zusammenarbeit über Git zugute, da jeder Beteiligte genau weiß, welche Dateien er bearbeiten und welche er nicht bearbeiten soll. Dies spart vor allem beim Mergen² viel Zeit, da es dort weniger Konflikte gibt. Die Aufgabenzuweisung erfolgt über GitLab, vom Autor, der durch die Weboberfläche ein Kanban Board bearbeiten und die Aufgaben Priorisieren und Zuweisen kann.

Um zu gewährleisten, dass die Gesamtzeit eingehalten wird, wurden die einzelnen Aufgaben unterteilt nach Komplexität, geschätzter Dauer und Relevanz für das Projekt. z. B. wurde bei der Entwicklung von Versionskontrolle Wert darauf gelegt, den Quellcode für die Interaktionen zwischen Front und Backend dynamischer und wiederverwendbarer zu gestalten. Für die Kommunikation zwischen dem Backend und dem Frontend bot sich deshalb GraphQL an, welches in der Lage ist, Queries und Mutations anhand von Datenmodellen durchzuführen.

Der Hauptgrund für die Entwicklung dieses Projektes, war die Automatisierung einer von Menschen ausgeführten Arbeit. Deshalb kann das Projekt nur dann als Erfolg gewertet werden, wenn die Software die Aufgaben mindestens, wenn nicht sogar besser als ein Mensch ausführen kann. Um dies zu gewährleisten, wurde bei der Projektplanung eine Umfrage gemacht, welche die dafür benötigten Funktionen der Software erfassen sollte. Der Fragebogen ist im Anhang ?? auf Seite ?? zu finden.

Um die Qualität des Endproduktes zu steigern, wurden mehrere Methoden angewandt. Damit eine positive Nutzererfahrung geschaffen werden konnte, wurden während der Durchführungsphase mehrere Whitebox Tests, von dem Entwickler Team, durchgeführt. Zusätzlich dazu, wurde in der Abschlussphase ein Blackbox Test von den zukünftigen Nutzer durchgeführt. Dieser Test entschied über die Abnahme der Software, oder ob diese Korrigiert werden muss..

²Ein Prozess bei dem zwei aktuelle Stände eines Projektes verglichen werden, um diese zu vereinigen

3 Analysephase

3.1 Ist-Analyse

Wie bereits in Abschnitt 1.2 erwähnt, existiert bereits ein ähnliches Programm mit dem Namen IT-Helper. Da diese Software mehrere Abläufe ermöglicht, wird in den Folgenden Unterpunkten versucht alle sinnvollen Abläufe abzudecken, am Ende wird dann auch noch einmal auf den allgemeinen Zustand der Software und des Quellcodes eingegangen.

Einrichtung

Bei der Einrichtung wird die ausführbare Setup Datei von dem einrichtenden auf ein externes Laufwerk Kopiert. Danach wird die Datei auf dem Datenträger ausgeführt und es kann über eine Schaltfläche eine Datei hinzugefügt werden. Im darauf folgendem Dialog können die Setup Datei, die Obergruppe, die Version und das Betriebssystem ausgewählt werden. Hierbei muss der Nutzer beachten, dass das Setup auf dem Datenträger abgespeichert ist und, dass die Version eins zu eins mit der Version, die später in einem Registrierungsschlüssel, vom Setup abgespeichert wird übereinstimmt. Danach speichert man den Eintrag ab und führt die Datei über das User Interface (UI) aus. Dieser Prozess muss für jedes Setup wiederholt werden.

Nutzung

Beim Programmstart, liest IT-Helper die schon installierten Programme und deren Versionen aus der Registry aus und vergleicht diese, mit der hinterlegten Version. Der Nutzer wird dann über eine Farbliche Codierung über den aktuellen Stand der Version informiert. Dieser Prozess, ist jedoch nicht zuverlässig und gibt Aufgrund von unflexibler Programmierung, oft das falsche Ergebnis zurück. Um Software zu installieren, muss der Nutzer nun eine Schaltfläche neben der gewünschten Software anklicken, welche das Setup startet. Von hier aus muss der Nutzer das Setup alleine vervollständigen.

Zustand der Software

Die Software ist Aufgrund eines beachtlichen Alters, fehleranfällig und teilweise unbenutzbar geworden. Der Zugriff auf eine MySQL Datenbank, welche diverse Volumen Lizenz-Schlüssel gespeichert hatte, ist nicht mehr möglich, weil die für den Datenaustausch genutzte Klassen-Bibliothek nicht mehr gepflegt wurde und nun nicht mehr mit der neuesten Windows Version kompatibel ist. Leider ist das Programm so aufgebaut, dass ohne diesen Zugang, das Starten der Anwendung

blockiert wird und somit auf neueren Maschinen nicht mehr ausführbar ist. Jedoch ist die Software, selbst wenn man sie gestartet bekommt fehlerhaft. IT-Helper erkennt manche der installierten Anwendungen nicht und zeigt oft, aufgrund von Eingabe Fehlern, die Version falsch an.

Zustand des Quellcodes

Der Quellcode von IT-Helper ist für eine weitere Verwendung nicht mehr zu gebrauchen. Bei der Programmierung der Software wurde kein Wert auf Don't Repeat Yourself (DRY) und Keep It Simple Stupid (KISS) gelegt, an vielen Stellen wird Code wiederholt und Datenbank Abfragen werden im UI Thread ohne ein Asynchrones Muster ausgeführt, als Beispiel hierfür wurde der Konstruktor des Hauptfenster als Anhang A.5: Konstruktor von MainWindow aus IT-Helper auf Seite v beigelegt.

Fazit

Da der Quellcode von IT-Helper zu alt ist und von der Qualität nicht ausreicht, wird Versionskontrolle komplett ohne Vorlage neu entwickelt.

3.2 Wirtschaftlichkeitsanalyse

Da zurzeit in Erwägung gezogen wird einen extra Arbeitsplatz für die ESA-IT zu schaffen, um diese mit der Arbeitslast zu entlasten, war es empfehlenswert, dieses Projekt durchzuführen. Ob es aber auch aus einem wirtschaftlichen Aspekt sinnvoll ist, wird im folgenden festgestellt.

3.2.1 „Make or Buy“-Entscheidung

Versionskontrolle dient als Lösung zu einem sehr spezifischen Problem. Es gibt zwar bereits Lösungen, um Software automatisiert zu installieren, aber keine ist exakt auf den Anwendungsfall von Probat anpassbar. Als Beispiel kann man innerhalb einer Domäne, über die Gruppenrichtlinien, Software verteilen. Dafür müssen diese Maschinen aber innerhalb der Domäne sein. Im Fall von Probat ist das aber leider Unmöglich, da die interne IT Abteilung, welche die technische Infrastruktur innerhalb von Probat verwaltet, das hinzufügen der zu verkaufenden bzw. zu reparierenden Workstations als ein zu hohes Sicherheitsrisiko für das interne Netz von Probat ansieht. Jegliche Software die in Frage kommen würde, scheitert an dieser Grenze, deshalb ist es Klar, dass Versionskontrolle im Hause von Probat programmiert werden muss.

3.2.2 Projektkosten

Im Nachfolgenden werden die ungefähren Kosten des Projekts berechnet. Hierfür wurden für den Stundenlohn und für die Ressourcenkosten der Mitarbeiter, von der Personalabteilung verallgemeinerte Beispiel Sätze bereitgestellt. Um die Kosten zu kalkulieren, wurde über das von Probat genutzte ERP System (AMS) in Erfahrung gebracht, wie Lange jeder Mitarbeiter an dem Projekt beteiligt war. Leider schließt das nicht die Auszubildenden ein, welche über den Zeitraum des Projekts die Stunden manuell festhalten mussten. Die dabei erfassten Stunden können der nachfolgenden Tabelle 2 entnommen werden.

Mitarbeiter	Planung	Entwurf	Durchführung	Verteilung	Abschluss	Gesamt
Auszubildender 1. Lehrjahr	0	2	43	0	5	50
Auszubildender 3. Lehrjahr (Autor)	7	12	36	3	12	70
Werkstudent	0	5	40	5	0	50
ESA-IT Mitarbeiter (Als Kollektiv)	1	0	0	0	21	22

Tabelle 2: Auflistung Stunden

Diese Daten wurden in Gruppen zusammengefasst und mit den vorher erwähnten Stundensätzen aus der Personalabteilung verrechnet. Die Kalkulation und die Stundenkosten können der folgenden Tabelle 3 entnommen werden.

Mitarbeiter	Personal- kosten die Stunde	Ressourcen Kosten die Stunde	Stunden Gesamt	Personal- kosten	Ressourcen- kosten	Gesamt
Auszubildende	10.00 €	15.00 €	120	1,200.00 €	1,800.00 €	3,000.00 €
Werkstudent	20.00 €	15.00 €	50	1,000.00 €	750.00 €	1,750.00 €
Mitarbeiter	25.00 €	15.00 €	22	550.00 €	330.00 €	880.00 €
Projekt Durchführungskosten gesamt						5,630.00 €

Tabelle 3: Kostenaufstellung

3.2.3 Amortisationsdauer

Das Ziel des Projekts ist es die Arbeitszeit, die nötig ist um einen Windows PC einzurichten auf ein Minimum zu reduzieren. Um festzustellen, wie viele Kosten es spart muss erst erfasst werden, welche Kosten das einrichten erzeugt. Hierzu wurden die Mitarbeiter über die schon weiter oben erwähnte Umfrage befragt. Das Resultat der Umfrage, war 3 Stunde pro PC und 3 PC's pro Woche. Leider kann in der folgenden Rechnung, die bessere Qualität nicht mit eingerechnet werden.

Rechnung Bei einer Zeiteinsparung von 3 Stunden pro PC in der Woche für 3 Computer und 50 Arbeitswochen im Jahr, ergibt sich eine gesamte Zeiteinsparung von

$$3 \cdot 50 \text{ Wochen/Jahr} \cdot 3 \text{ h/Woche} = 450 \text{ h/Jahr} \quad (1)$$

Dadurch ergibt sich eine jährliche Einsparung von

$$450 \text{ h} \cdot (25 + 15) \text{ €/h} = 18000 \text{ €} \quad (2)$$

Die Amortisationszeit beträgt also $\frac{5630.00 \text{ €}}{18000 \text{ €/Jahr}} \approx 0,32 \text{ Jahre} \approx 16 \text{ Wochen}$.

Hinweis: Werte wurden bei Ergebnissen nach zwei Nachkommastellen aufgerundet, um die Richtigkeit

des wirtschaftlichen Fazits zu gewährleisten

3.3 Nicht-monetäre Vorteile

In der Nutzung der Software liegen weitere Vorteile für die ESA-IT und somit auch für Probat. Ein großer Vorteil der gewonnen wird, ist die Qualität, da Versionskontrolle nicht nur Beihilfe leistet, Programme zu installieren, sondern auch in der Lage ist, Registrierungsschlüssel und Ordnerstrukturen zu überprüfen, um den Zustand einer Installation an den Benutzer weiterzugeben.

Ein weiterer Nutzen der Software ist die gesteigerte Konzentration der Mitarbeiter, welche vorher während der Einrichtung von PC's, oft eine oder mehrere Aufgaben nebenher ausgeführt haben. Durch Versionskontrolle sind diese Mitarbeiter in der Lage sich ganz auf ihre Aufgabe/n zu konzentrieren ohne den Installationsprozess überprüfen zu müssen.

Ein weiterer möglicher Vorteil, dessen Wirtschaftlichkeit zurzeit von Probat geprüft wird ist, dass die Software in Verbindung mit einer neuen, später dieses Jahr geplanten Software, namens IT-Log automatisch Hardware und Versanddaten erfassen kann und diese Selbständig in AMS eintragen kann.

3.4 Anwendungsfälle

Eine grobe Übersicht der Anwendungsfälle, welche von Versionskontrolle abgedeckt werden, wurde in der Analysephase des Projekts als Use-Case Diagramm visualisiert. Dieses ist im Anhang A.7: Use-Case-Diagramm auf Seite ix zu finden. Das Diagramm stellt die Funktionen dar, welche vom Nutzer ausgeführt werden können.

3.5 Lastenheft/Fachkonzept

Am Ende der Analysephase wurde, über die gewonnenen Daten aus der Umfrage, ein Lastenheft erstellt. Auszüge aus diesem Lastenheft befinden sich im Anhang A.4: Lastenheft (Auszug) auf Seite iv.

4 Entwurfsphase

4.1 Auswahl der Zielplattformen

4.1.1 Auswahl der Programmiersprache

Da Versionskontrolle eine Software ist, welche ausschließlich innerhalb von Probat eingesetzt wird, muss es in der Lage sein, auf jeden beliebigen Windows 10 PC zu funktionieren. Die niedrigste Angeforderte Windows Version ist 1709, also muss Versionskontrolle alle Windows Versionen die danach kommen unterstützen. Diese Anforderung gilt jedoch nur für den Client, welcher die Setups herunterlädt und installiert. Das Backend wird hingegen in mehreren Docker Containern über den, von Probat gehosteten Docker Stack bereitgestellt. Das heißt, dass die Backend Anwendungen mit Linux kompatibel sein müssen. Wichtig bei der Auswahl der Zielplattform ist auch, ob die verwendete Programmiersprache innerhalb von Probat standardmäßig genutzt wird. Die hauptsächlich verwendeten Sprachen für Server/Client Anwendungen sind C++, C#, Java. Java wird hierbei aber nicht beachtet, da es ohne die vorher installierte JVM kein lauffähiges Programm bereitstellen kann. In Sachen Kompatibilität ist C++ der Spitzenreiter, da es beim kompilieren in Maschinensprache verwandelt wird, jedoch ist auch C# kompatibel mit allen erfordernten Windows Versionen. Also sind die haupt Entscheidungsaspekte, die Lesbarkeit vom Quellcode und der Aufwand, um diesen Code zu schreiben. In beiden dieser Punkte liegt C#, durch seine fast Menschenähnliche Syntax, vorne. C# ist somit die Ausgangssprache, mit der die gesamte Anwendung programmiert wird.

4.1.2 Genutzte Zielplattformen

Client Als erstes steht die Auswahl der Client Anwendung im Vordergrund. Da Versionskontrolle eine Anwendung wird, welche auf jeder Windows 10 Version 1709 Installation, ohne zusätzliche Features funktionieren soll, wird als Zielplattform .NET 5.0 verwendet. Um die maximale Kompatibilität zu gewährleisten wird die Executable als Self-packaged executable erstellt d.h., dass alle Abhängigkeiten wie z. B. Dynamic Link Library (DLL) Dateien in der Auszuführenden Datei verpackt sind. Das erhöht zwar die Größe dieser Datei, sorgt aber auch dafür, dass diese ohne Abhängigkeiten auf jedem Windows PC, welche eine Version von Windows 7 Service Pack 1 oder höher nutzt, ausgeführt werden kann.

Backend Das Backend von Versionskontrolle, muss im Gegensatz zur Client Anwendung auf einem Linux Betriebssystem funktionieren. Dies stellt aber kein Problem dar, denn .Net 5.0 Anwendungen sind ausführbar auf Windows, Linux und Mac OS Maschinen. Somit ist auch hier .Net 5.0 die Eingesetzte Zielplattform. Ein Vorteil der sich daraus ergibt ist, dass ein Teil des Quellcodes, welches im Backend genutzt wird, z. B. das Datenbank Modell, wiederverwendet werden kann.

4.2 Projekttypen

Für die Anwendung wurden die modernsten Projekttypen genommen, welche Zurzeit für eine Windows Desktop Anwendung verfügbar sind. Das wäre für den Client eine WPF Anwendung und für das Backend eine ASP.Net Anwendung. Beide Projekttypen werden von der genutzten Integrated Development Environment (IDE) (Visual Studio 2019 Professional) und der Zielplattform unterstützt.

4.3 Architekturdesign

Der Client wird das Model View View Model (MVVM) Design nutzen, dieses basiert auf dem Model View Controller (MVC) Design. MVVM sorgt für einen Übersichtlicheren Quellcode und erzwingt durch seine Struktur einen nicht blockenden Programmierstil d. h. dass Berechnungen, nicht auf dem UI Thread ausgeführt wird. Das führt dazu, dass die Oberfläche der Anwendung nicht einfriert und eine besser Nutzererfahrung geschaffen wird.

Das Backend wird als Microservice Architektur Designt. Das bietet vor allem Vorteile im Bereich der Wiederverwendbarkeit, z. B. wird für dieses Projekt ein Authentifizierungs-Service aus einem anderen Projekt vollständig recycelt, hierfür müssen nur einige Umgebungsvariablen über die Docker-Compose/Stack Datei geändert werden. Auch der Reverseproxy „Nginx“, benötigt durch diese Architektur eine minimale Einrichtung und stellt das gesamte Backend als einen Server nach außen hin dar. Ein Diagramm, wie das Backend funktioniert, genauso wie die Anbindung zum Client, ist im Anhang A.6: Microservice Architektur auf Seite viii zu finden.

4.4 Entwurf der Benutzeroberfläche

Beim Design der Benutzeroberfläche, wurde auf ein minimalistisches Design geachtet. Der Nutzer sollte in der Lage sein, die Software intuitiv zu Steuern und nicht durch unnötig komplexe Animationen oder Design Entscheidungen verwirrt werden. Der Designentwurf kann im Anhang A.9: Design Mockup auf Seite xi gefunden werden.

4.5 Datenmodell

Die Datenstruktur, wurde Anhand eines Entity-Relationship Diagram (ERD) festgehalten. Zur Darstellung in dieser Dokumentation, wurde das ERD überarbeitet, nur die Entitäten und die Kardinalitäten zu beinhalten. Zu finden ist diese in Abbildung 1. Die originale Fassung befindet sich im Anhang A.11: Datenbankmodell auf Seite xiii

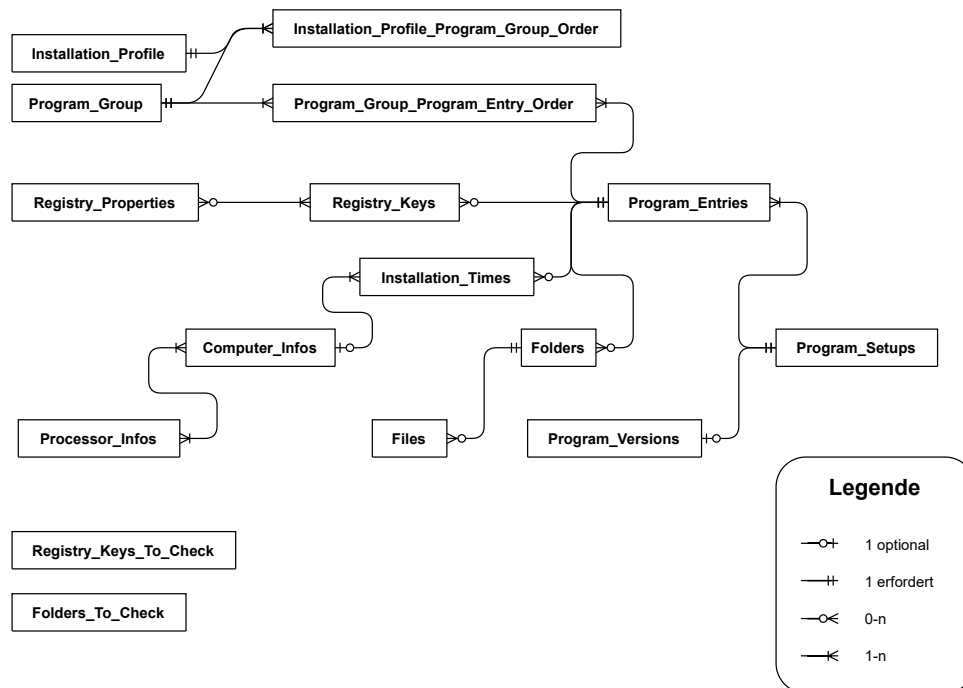


Abbildung 1: Vereinfachtes ER-Modell

In diesem Entity-Relationship-Modell (ERM) sind die Entitäten: ProgramSetups, ProgramEntries, ProgramGroups, InstallationProfiles und ProgramVersion die wichtigsten, denn sie ermöglichen die Kernfunktionalität. ProgramSetups speichert die Metadaten eines Setups, wie z. B. den Namen, den Dateipfad auf dem NAS oder den Dateityp. ProgramEntries speichert Konfigurationen der einzelnen Setups, z. B. Startargumente und genauer definierte Namen. Die darüber liegende Entität ProgramGroups dient als Gruppierende Entität, um Setup Konfigurationen als Gruppe abzuspeichern. Die wieder darüber liegende Entität InstallationProfiles, dient auch als Gruppierende Entität um Gruppen zu Profilen zusammen zu schließen. Eine Visualisierung dieses Datenmodells befindet sich im Anhang A.12: Datenmodell Visualisierung auf Seite xv.

Diese vorher genannten Datenmodelle, werden benötigt, um die Sortierung und Installation von Software, so intuitiv wie möglich zu machen. Jedoch umfasst das alleine nicht alle Funktionen, die Versionskontrolle besitzen soll. Denn es soll z. B. auch in der Lage sein die Version eines Programms zu überprüfen und dem Nutzer mitzuteilen, ob diese geupdated werden kann. Um diese Funktion zu ermöglichen, wird der ProgramSetups Entität die ProgramVersions Entität in einer 1:1 Beziehung angehängt.

4.6 Geschäftslogik

Das Datenmodell, welches sich aus dem ERD ergibt, konnte schon im voraus als Klassendiagramm visualisiert werden. Dieses Klassendiagramm ist im Anhang A.13: Klassendiagramm auf Seite xvi beigelegt. Dieses ist eine erste Fassung und hat sich im Laufe des Projekts verändert.

4.7 Maßnahmen zur Qualitätssicherung

Um die Qualität der Software zu steigern, wurden Whitebox und Blackbox Tests durchgeführt. Die Whitebox Test fanden während der Entwicklung statt und trieben diese mit an und die Blackbox Tests wurden am Ende ausgeführt, als Qualitätssicherung zur Abnahme der Software. Beide Tests starteten damit, dass die Nutzer den normalen Anwendungsfall ausführen sollten und endete damit, dass die Nutzer die Software nach freiem Willen benutzen sollten.

Zusätzlich wurden während der Entwicklung Code-Reviews durchgeführt. Diese fanden immer nach einem Commit in die Versionsverwaltung GitLab statt und wurden vom Werkstudenten oder dem Autor durchgeführt. Hierbei wurde ganz besonders auf das DRY und das KISS Prinzip geachtet, welche zur Wartung, Erweiterbarkeit und Übersicht beitragen geachtet.

4.8 Pflichtenheft/Datenverarbeitungskonzept

Am Ende der Entwurfsphase, wurde aus den Ergebnissen des Lastenhefts, ein Pflichtenheft erstellt. Dieses beruht auf ein vordefiniertes Schema, welches von Probat genutzt wird. Dieses ist im Anhang A.8: Pflichtenheft (Auszug) auf Seite x zu finden.

5 Durchführungssphase

5.1 Implementierung der Datenstrukturen

Die Datenstrukturen wurden vom Autor über eine Separate Klassenbibliothek für das Frontend, sowie das Backend Projekt bereitgestellt. Als Vorlage diente hierzu das in 4.5 Datenmodell gezeigte ERD. In dieser Klassenbibliothek finden sich verschiedene Klassen, um das fehlerfreie Kommunizieren zwischen Server und Client zu garantieren. Auszüge, der vom Autor erstellten Klassen sind im Anhang ?? auf Seite ?? zu finden.

5.2 Implementierung der Benutzeroberfläche

Die Umsetzung des Designs übernahm der Auszubildende aus dem ersten Lehrjahr, hierbei musste kein Wert auf das Corporate Design gelegt werden, da die Anwendung niemals außerhalb von Probat eingesetzt wird. Der Autor übernahm hierbei die Implementierung der MVVM Verlinkungen zum Datenmodell. Screenshots der Anwendung während der Entwicklung, mit Dummydaten können im Anhang A.10: Screenshots der Anwendung auf Seite xi gefunden werden.

5.3 Implementierung der Geschäftslogik

Beim Programmieren der Anwendung, wurde besonders viel Wert auf die Wiederverwendbarkeit gelegt. Hierfür entwickelte der Autor generische Methoden und Klassen, welche die Verbindung vom Client zum GraphQL Backend steuern. Diese Methoden/Klassen wurden so abstrakt wie möglich gehalten, um den erstellten Quellcode in jeglichen .Net 5.0 Anwendungen verwenden zu können. Zum Abstrahieren des Quellcodes, hat der Autor zuerst die Schritte, des Datenmodells, sowie der Verbindung zum Backend voneinander getrennt. Dann entwickelte er, unter Verwendung der GraphQL.Client Klassenbibliothek einen Singleton Service, welcher die Verbindung zum Server herstellt, Anfragen schickt, und die Ergebnisse davon in brauchbare Datenmodelle deserialisiert. Dieser GraphQL Service wird von dem DataProvider Singleton Service genutzt, welcher die Möglichkeit bietet, Listen zu Datenänderungen, zu abonnieren. Dies führt zu einem Automatischen Update des Datenmodells auf der Client Seite. Der DataProvider stellt außerdem Funktionen bereit, um Daten zu manipulieren. Diese Funktionen aktualisieren nicht nur das Datenmodell des Clients, sondern auch das vom Backend. Alle diese Generischen Methoden basieren auf das IGraphQLQueryable Interface, welches Standardimplementierungen für Funktionen zur Datenmanipulierung bietet. Diese Funktionen nutzen ein vom Autor programmierte Objekt-Erweiterungsklasse, welche das Serialisieren eines Objekts in eine GraphQL Query ermöglicht. Außerdem ermöglicht es eine dynamischere Nutzung der Datenmodelle, da über die Klassenbibliothek System.Runtime.Reflection Eigenschaftsinformationen ausgelesen werden und anschließend nach Bedarf dynamisch Werte gelesen und gesetzt werden können. Das Interface IGraphQLQueryable bietet außerdem noch Query Eigenschaften, welche alle Create, Read, Update und Delete (CRUD) Operationen abdecken. Die bisher genannten Datentypen, stammen alle aus der Klassenbibliothek vom Autor und wurden bereits in Kapitel 5.1 Implementierung der Datenstrukturen erwähnt. Der Quellcode zu den Klassen ist wie in dem Kapitel erwähnt im Anhang ?? auf Seite ??.

Beispiel: Hinzufügen eines ProgramSetups Als erstes wird ein neues Objekt vom Typen ProgramSetup erstellt, danach wird das Objekt der Create Funktion vom DataProvider übergeben. Dieser ruft nun Registrierungs Methoden auf, welche Fremdschlüssel auf anderen Objekten setzen. Nachdem das Abgeschlossen ist, fragt der DataProvider die Eigenschaft CreateQuery auf dem ProgramSetup Objekt ab. Diese enthält, Dank der Standard Implementierung vom IGraphQLQueryable Interface, die passende CreateQuery, mit dem Serialisierten ProgramSetup. Diese Query wird dann

weiter an die SendQuery Methode, des GraphQLServices weitergegeben, welcher diese an das Backend schickt und das Ergebnis in den mitgegebenen Typen umwandelt. Das Empfangen und Casten, in diesen Dateitypen, erfolgt über die Newtonsoft.Json Klassenbibliothek. Das Objekt, welches nun eine eigene ID besitzt, wird wieder zurück zum DataProvider geleitet, wo es dann zuletzt dem Daten Cache hinzugefügt wird.

Beispiel: Hinzufügen einer ProgramVersion Als erstes wird die Programmversion vom Betriebssystem ausgelesen und in ein Objekt vom Typen ProgramVersion gespeichert. Das dazugehörige ProgramSetup ist als Fremdschlüssel, schon in dem Modell gespeichert. Das Objekt wird nun dem DataProvider in der Create Methode übergeben, wo dieser die Registrierungs Methode des Objektes, definiert im Interface IGraphQLQueryable, aufruft. Diese Methode registriert die ID des ProgramVersion Objekts, im dazugehörigen Fremdschlüssel vom ProgramSetup Objekt. Nachdem dieser Prozess abgelaufen ist, wird die Version in einer Query serialisiert und über den GraphQLService an das Backend geschickt. Das resultierende Ergebnis wird dann im Cache vom DataProvider gespeichert. Danach wird noch die DataChanged Methode vom DataProvider aufgerufen, um die Daten in der UI zu aktualisieren.

Beispiel: Ändern eines ProgramEntry Das Objekt wird über die Update Methode an den DataProvider übergeben, dort wird es über schon zuvor erwähnte Funktionen serialisiert und an das Backend geschickt. Anschließend wird das zurückgeschickte Objekt im Cache an Stelle des alten Objekts eingesetzt und eine Methode zum updaten der UI wird aufgerufen.

6 Abnahme und Einführungsphase

6.1 Abnahmephase

Die Abnahmephase lief reibungslos ab. Beim Testen von der ESA-IT wurden keine gravierenden Mängel erkannt, allerdings, wurden kleinere Mängel entdeckt, z. B. wurde die UI nicht sofort geupdated, nachdem ein neuer Eintrag hinzugefügt wurde. Dies geschah erst, nachdem man die Seite gewechselt hat. Diese mussten erst vor der offiziellen Abnahme gefixt werden.

6.2 Einführungsphase

Die Anwendung, wurde nach den letzten Bug fixes offiziell abgenommen und in die Produktions Umgebung des Docker Stack hochgeladen. Hierfür wurde eine CI/CD Pipeline eingerichtet, welche vollautomatisch bei jeder Änderung im Masterbranch des Git Verzeichnisses eine neue Version hoch lädt. Die dafür erstellte .gitlab-ci.yml Datei ist im Anhang ?? : ?? auf Seite ?? zu finden. Während des Durchlaufs, der Pipeline, werden Build Tests durchgeführt, um einen möglichen Ausfall,

durch eine fehlerhafte Änderung zu vermeiden. Erst wenn alle Build Tests erfolgreich abgeschlossen sind, wird das alte, durch das neue Backend ersetzt.

6.3 Dokumentation

Versionskontrolle wurde durch das Benutzerhandbuch, erstellt vom Auszubildenden aus dem ersten Lehrjahr, und der Projektdokumentation festgehalten.

7 Fazit

7.1 Soll-/Ist-Vergleich

Das Projekt war ein voller Erfolg, alle Anforderungen des Pflichtenhefts, wurden erfüllt. Der Zeitplan ging auch auf, jedoch ist im folgenden Soll/Ist Vergleich zu erkennen, dass der Puffer an den Durchführungszeitraum angehängt wurde.

Phase	Geplant	Tatsächlich	Differenz
Projektanalyse	7 h	7 h	
Projektentwurf	12 h	12 h	
Projektdurchführung	33 h	36 h	+3 h
Projektverteilung	3 h	3 h	
Projektabschluss	12 h	12 h	
Pufferzeit	3 h	0 h	-3 h
Gesamt	70 h	70 h	

Tabelle 4: Soll-/Ist-Vergleich

7.2 Gewonnenes Wissen

Der Autor konnte in diesem Projekt, viele Dinge über das Projektmanagement, Zeitplanung und agiler Softwareentwicklung lernen. Die wiederverwendbaren Klassen für die Zusammenarbeit zwischen Client und Server über GraphQL werden in den nächsten Projekten für Probat sehr nützlich sein und Zukünftige Entwicklungskosten senken. Dadurch war dieses Projekt in Hinsicht auf das gewonnene Produkt, sowie in Hinsicht auf das gewonnene Wissen ein voller Erfolg.

7.3 Ausblick

Da Versionskontrolle viele Möglichkeiten im Bereich der Automatisierung bietet, wurden schon Pläne für die Integration in ein anderes Projekt gemacht. Dort wird Versionskontrolle nicht nur PC's einrichten, sondern auch Daten über diesen Sammeln, welche dann automatisiert in AMS eingepflegt werden. Dies wird zu einer weiteren Entlastung der ESA-IT führen und die Qualität weiter steigern. Es ist sogar denkbar, über diese erfassten Daten festzustellen, wann bereits eingerichtete Computer fehlerhaft werden oder ganz funktionsunfähig werden.

Literaturverzeichnis

Eidesstattliche Erklärung

Ich, Dennis van Elk, versichere hiermit, dass ich meine **Dokumentation zur betrieblichen Projektarbeit** mit dem Thema

Entwicklung von Versionskontrolle– Neuentwicklung einer Server-Client Anwendung, zur automatischen Konfiguration und Installation von Software auf der Basis von Asp .Net 5.0

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.

Emmerich am Rhein, den 15.04.2021

DENNIS VAN ELK

A Anhang

A.1 Detaillierte Zeitplanung

Projektanalyse	7 h
1. Meeting mit zukünftigen Nutzern	1 h
2. Analysieren des Ist Zustands	3 h
3. Wirtschaftlichkeitsprüfung und Amortisierungsrechnung des Projektes	1 h
4. Erstellen des Lastenhefts	2 h
Projektentwurf	12 h
1. Erstellen eines Programmablauf Plans	2 h
2. Datenbankentwurf	3 h
2.1. ER-Modell erstellen	2 h
2.2. Tabellenmodell erstellen	1 h
3. Benutzeroberflächen entwerfen	1 h
4. Erstellen einer Microservice Architektur	1 h
5. Erstellen des Pflichtenhefts	4 h
6. Erstellen eines Use-Case Diagramms	1 h
Projektdurchführung	33 h
1. Umsetzung der Oberfläche in WPF als XAML Quellcode	1 h
2. Programmierung des Backends	22 h
2.1. Implementierung des Datenbank Modells unter Verwendung von EFCore	5 h
2.2. Erstellen von GraphQL Typen	10 h
2.3. Erstellen von GraphQL Queries und Mutations	6 h
2.4. Testen der GraphQL Queries und Mutations	1 h
3. Implementierung von GraphQL im Frontend	10 h
3.1. Services erstellen die, die Verbindung steuern	5 h
3.2. Services testen	5 h
Projektverteilung	3 h
1. Upload zu einer Quellcode Verwaltung	1 h
2. CI/CD Integration	2 h
Projektabschluss	12 h
1. Bedienschulungen	1 h
2. Testphase der abzunehmenden Mitarbeiter	1 h
3. Erstellen der Dokumentation	10 h
Pufferzeit	3 h
1. Puffer	3 h
Gesamt	70 h

A.2 Verwendete Ressourcen

Hardware

- Arbeitsplatz mit Desktop PC und zwei Bildschirmen
- Alter Server mit 6TB Festplattenspeicher
- Vorhandene Testsysteme

Software

- Betriebssystem - Microsoft Windows 10 Pro 20H2
- IDE - Microsoft Visual Studio 2019 Professional
- Texteditor - Microsoft Visual Studio Code
- Objektrelationales Datenbankmanagementsystem (ORDBMS) - PostgreSQL
- Nutzer Verwaltung - Active Directory (AD)
- Verwaltung der Datenbank - pgAdmin 4 v5.1
- Diagramme - Draw.io (Diagrams.net)
- Versionsverwaltung - Git/Gitlab
- NAS Betriebssystem - TrueNAS von Open Storage
- Docker Umgebung - Docker Desktop/Docker Swarm
- Automatische Deployments - Gitlab CI/CD
- Object-Relational Mapping (ORM) - EFCore
- Packet Manager - NuGet Package Manager
- API Testwerkzeug - Postman

Personal

- Entwickler - Auszubildender Fachinformatiker für Anwendungsentwicklung (1. Lehrjahr)
- Entwickler/Code-Review - Werksstudent Informatik Softwaresysteme
- Entwickler/Code-Review/Organisator - Dennis van Elk (Author)
- Tester - Drei Mitarbeiter der ESA-IT

A.3 Umfrage zur Erfassung der benötigten Features

Umfrage zu Versionskontrolle

Feature	Wichtigkeit (1-5)
1. Authentifizierung der Nutzer	
2. Zentralisierte Datenverwaltung, zur Minimierung von	
3. Installieren der Software in der richtigen Reihenfolge	
4. Einrichtung von einzelnen Konfigurationen für die Setup Dateien	
5. Einrichtung von Gruppen bestehend aus Konfigurationen	
6. Einrichtung von Installationsprofilen bestehend aus Programmgruppen	
7. Einhalten der Reihenfolge innerhalb eines Installationsprofil/Programmgruppe	
8. Anzeige von Versionen	
9. Ein funktion zum deinstallieren von Software	
10. Verbleibende Zeit anzeige	
11. Den Fortschritt der Installation loggen	
12. Anzeige, ob die Version veraltet ist	

Falls ein zusätzliches Feature gewünscht ist, dieses bitte als Issue auf GitLab:

<https://gitlab.probat.com/internal-software/versionskontrolle/-/issues>

Wieviele PC's werden Wöchentlich mit dem Programm IT-Helper eingerichtet?

Wielange dauert das Einrichten eines PC's mit IT-Helper

Welche Probleme treten zurzeit bei IT-Helper auf

A.4 Lastenheft (Auszug)

Es folgt ein Auszug aus dem Lastenheft, über die funktionalen Anforderungen der Anwendung:

Die Anwendung muss folgende Anforderungen erfüllen:

1. Allgemeine Funktionsanforderungen

- 1.1. Die Nutzung der Anwendung darf nur für authentifizierte Nutzer möglich sein.
- 1.2. Datenänderungen im Client sollen sofort mit dem Backend synchronisiert werden.
- 1.3. Die Software muss alle Elemente der Liste mit ausgewählten Elementen in der richtigen Reihenfolge installieren können.

2. Verwaltung der Setupdaten

- 2.1. Die Anwendung muss Setupdaten als Eintrag festlegen können und die dazugehörige Setupdateien auf ein NAS hochladen.
- 2.2. Jedes Setup kann mehrere Konfigurationen zugewiesen bekommen.
- 2.3. Das Gruppieren der Konfigurationen muss ermöglicht und die Reihenfolge dabei beachtet werden.
- 2.4. Das zusammenstellen von mehreren Gruppen zu einem Installationsprofil unter Einhaltung der Reihenfolge, muss möglich sein.

3. Darstellung der Daten

- 3.1. Die Anwendung muss eine Liste aller Setups, Konfigurationen, Gruppen und Installationsprofile anzeigen.
- 3.2. Das selektieren einzelner Elemente, die installiert werden sollen, muss möglich sein.
- 3.3. Der Status von Dateien zum Upload soll über einen Ladebalken erfolgen.
- 3.4. Die Anwendung muss bei der Installation von Software eine ungefähre verbleibende Zeit angeben.

4. Sonstige Anforderungen

- 4.1. Die Anwendung muss auf jeder Windows 10 Version nach 1709 vollständig funktionieren.

A.5 Konstruktor von MainWindow aus IT-Helper

Das ist der originale Quellcode von MainWindow.cs aus IT-Helper

```
1 public MainWindow()
2 {
3     string installedVersion = System.Windows.Forms.Application.ProductVersion;
4
5     T_SEARCH_THREAD = new Thread(new ThreadStart(DoSearch));
6     try
7     {
8         LoadWindow wnd_load = new LoadWindow();
9         wnd_load.Show();
10
11         Ping pingSender = new Ping();
12         bool pingSuccess = false;
13
14         PingReply reply = pingSender.Send(MySqlServer);
15         pingSuccess = reply.Status == IPStatus.Success;
16
17
18         if (pingSuccess)
19         {
20             string currentVersion = "";
21             string downloadLink = "";
22
23
24             MySqlConnection conn = new MySqlConnection(MySqlConnectionStrings);
25             conn.Open();
26             MySqlCommand cmd = new MySqlCommand("SELECT downloadlink, version FROM 'update' ORDER BY
                version DESC LIMIT 1", conn);
27             MySqlDataReader reader = cmd.ExecuteReader();
28             while (reader.Read())
29             {
30                 currentVersion = reader["version"].ToString();
31                 downloadLink = reader["downloadlink"].ToString();
32             }
33             reader.Close();
34             conn.Close();
35
36             int[] iVersion = new int[4];
37             int[] nVersion = new int[4];
38
39             if (installedVersion.Split('.') .Length <= 4)
40             {
41                 for (int i = 0; i < installedVersion.Split('.') .Length; i++) { int.TryParse(installedVersion.Split('.') [
                    i], out iVersion[i]); }
42             }
43             if (currentVersion.Split('.') .Length <= 4)
44             {
```

```

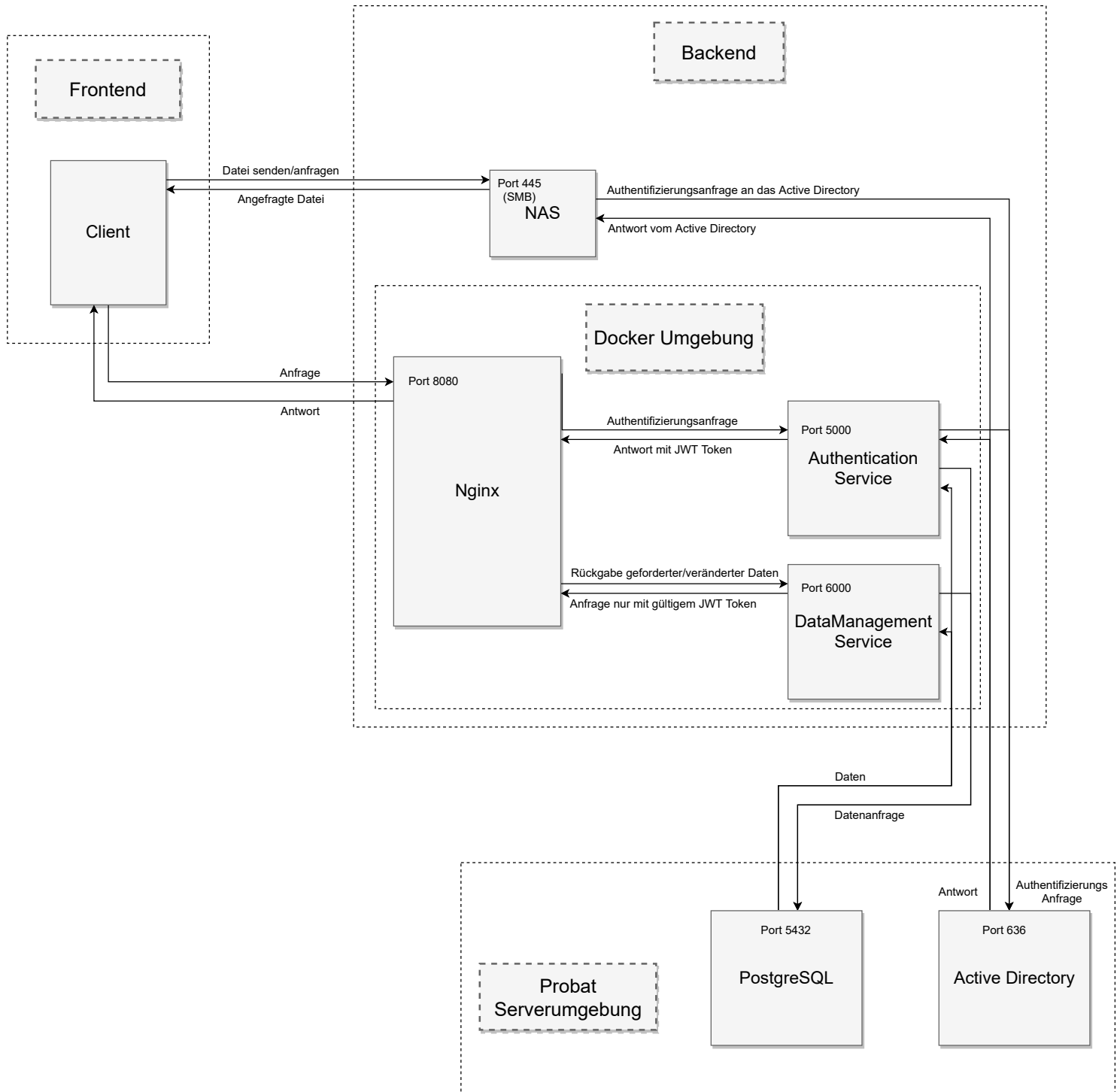
45         for(int i = 0; i < currentVersion.Split('.') .Length; i++) { int.TryParse(currentVersion.Split('.') [i],
46             out nVersion[i]); }
47     }
48     int idx = iVersion.Length;
49     bool doUpdate = false;
50
51     for (int i = 0; i < idx; i++)
52     {
53         doUpdate = nVersion[i] > iVersion[i];
54         if (doUpdate) { break; }
55     }
56     if (doUpdate)
57     {
58         using(System.Net.WebClient webClient = new System.Net.WebClient())
59         {
60             if (File.Exists(UPDATE_FILENAME)) { File.Delete(UPDATE_FILENAME); }
61             webClient.DownloadFile(new Uri(downloadLink), UPDATE_FILENAME);
62             MessageBox.Show($"Ein Update wurde heruntergeladen.\n\nInstallierte Version: {installedVersion}
63                 \nVerfügbare Version: {currentVersion}\n\nDas Update wird jetzt installiert.");
64             Process.Start("installUpdate.bat");
65             Environment.Exit(0);
66         }
67     }
68     else
69     {
70         MessageBox.Show("Der Host unter \"" + MySqlServer + "\" konnte nicht erreicht werden!\nIst eine
71             Netzwerkverbindung mit dem PROBAT-Netzwerk verfügbar?", "Netzwerkfehler", MessageBoxButtons.
72             OK, MessageBoxIcon.Error);
73         Environment.Exit(0);
74     }
75
76     Version OS_WIN7 = new Version(6, 1);
77     if (Environment.OSVersion.Platform == PlatformID.Win32NT && Environment.OSVersion.Version >=
78         OS_WIN7)
79     {
80         OS_Version = ListenProgramme.state.Windows_7;
81     }
82     else
83     {
84         OS_Version = ListenProgramme.state.Windows_XP;
85     }
86     InitializeComponent();
87     Add_Software_to_Listview();
88     lv_setups.ItemsSource = visible_programs;
89     chk_show_normal.IsChecked = true;
90     Thread.Sleep(1000);
91     wnd_load.Close();
92     write_into_console("Programm gestartet", true);
93     tab_choose_group.Visibility = System.Windows.Visibility.Visible;

```

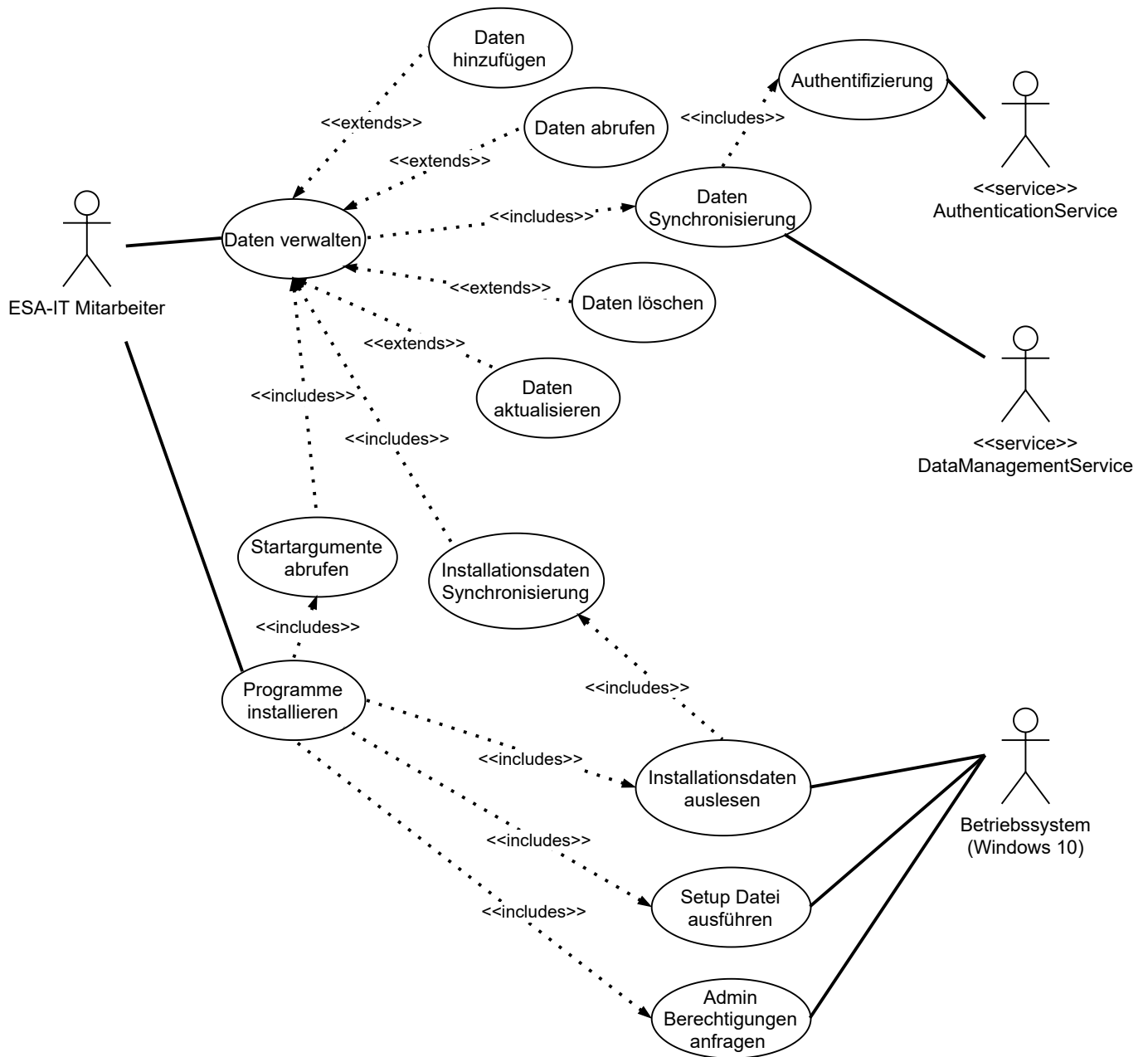


```
90     this.Title = $"PROBAT Versionskontrolle v{installedVersion}";
91
92 }
93 catch (System.FormatException ex)
94 {
95     MessageBox.Show(ex.Message.ToString(), "Unbekannter Fehler", MessageBoxButtons.OK, MessageBoxIcon.
96         Error);
97     write_into_console("EXCEPTION: " + ex.Message + " | TYPE: " + ex.TargetSite);
98
99     Environment.Exit(0);
100 }
101 catch(PingException)
102 {
103     MessageBox.Show($"Der MySQL-Server unter {MySqlServer} konnte nicht gefunden werden.", "
104         Netzwerkfehler", MessageBoxButtons.OK, MessageBoxIcon.Error);
105     Environment.Exit(0);
106 }
107
108 dragMgr = new ListViewDragDropManager<ViewSetup>(this.lv_setups);
109 }
```


A.6 Microservice Architektur



A.7 Use-Case-Diagramm



A.8 Pflichtenheft (Auszug)

Zielbestimmung

1. Musskriterien

1.1. Liste mit Setups, Konfigurationen, Gruppen und Profile

- In der Liste wird jeweils der Name und die Beschreibung des Elements angezeigt.
- Der Status der Installation soll farblich angezeigt werden, rot für nicht installiert, gelb für alte Version, grün für neueste Version und blau für neuerere Version als zurzeit installiert werden kann.
- Die Liste soll sich bei Datenänderungen automatisch updaten
- Es soll durch ein Suchfeld möglich sein, nach Elementen zu suchen.
- Der Nutzer muss in der Lage sein können, alle Daten, die in der Liste stehen, zu bearbeiten.

1.2. Installation von Software

- Der Nutzer soll aus der bereits erwähnten Liste Elemente auswählen und die einer Warteschlange hinzufügen können.
- Mit einem Klick auf die Schaltfläche „Installation starten“ soll der Prozess der Installation gestartet werden.
- Die Installation muss, wenn möglich fehlerfrei ablaufen.
- Nach Abschluss der Installation müssen Daten über den PC, wie z. B. hinzugefügte Registrierungsschlüssel und hinzugefügte Ordner gesammelt werden.

1.3. Sonstiges

- Setup Dateien sollen alle auf ein vorher bestimmtes NAS abgespeichert werden.
- Die Anwendung soll leicht erweiterbar sein.
- Die Anwendung muss auf allen Windows 10 Versionen nach 1709 funktionieren.

Produkteinsatz

1. Anwendungsbereiche

Die Client Anwendung wird hauptsächlich für Workstations und Servicetechniker Notebooks eingesetzt.

2. Zielgruppen

Versionskontrolle wird von der ESA-IT Abteilung genutzt.

3. Betriebsbedingungen

Das Backend von Versionskontrolle soll 24/7 die Woche durchlaufen. der Client hingegen nur, wenn er tatsächlich genutzt wird.

A.9 Design Mockup

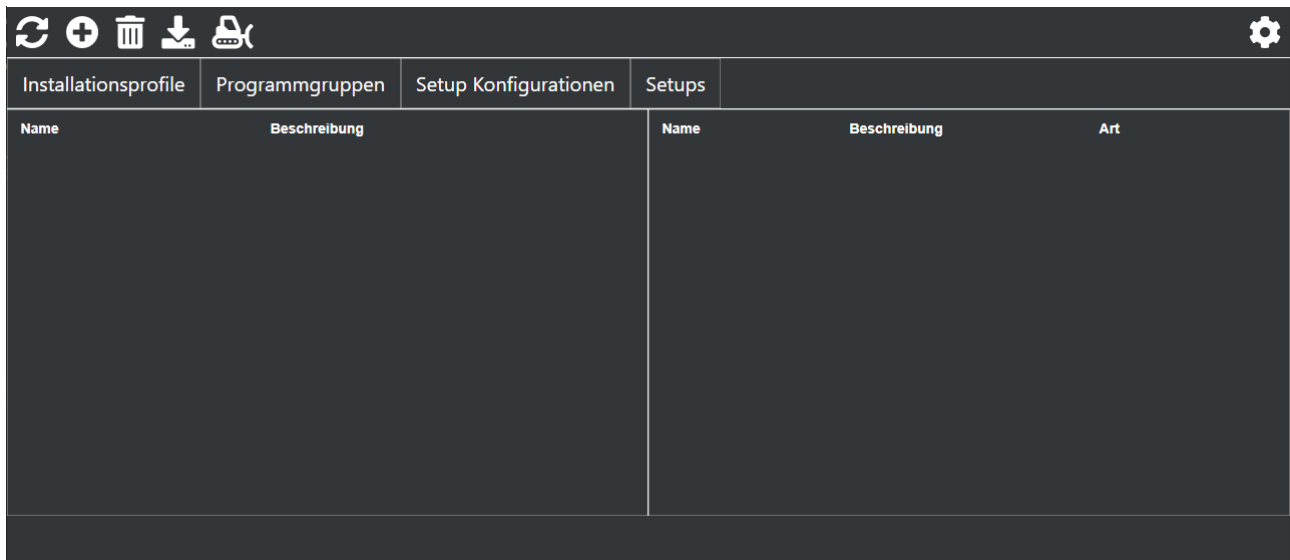


Abbildung 2: User Interface Mockup

A.10 Screenshots der Anwendung

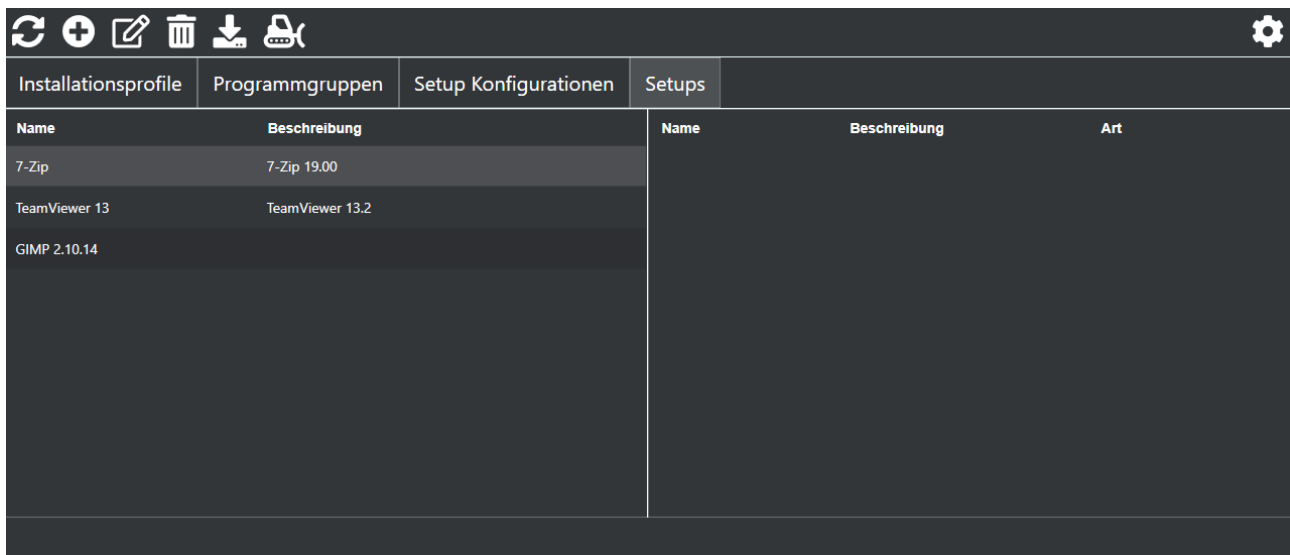
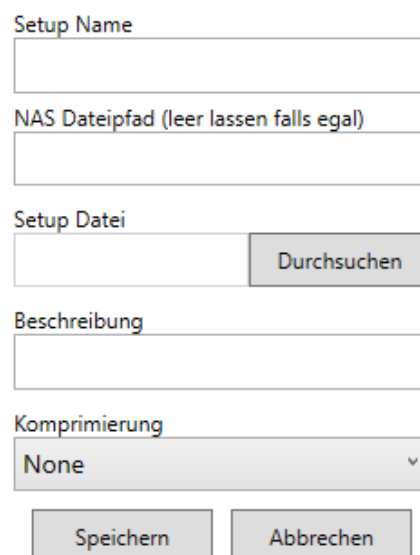


Abbildung 3: Screenshot #1

A screenshot of a web-based setup form. It contains several input fields and buttons. The fields are labeled "Setup Name", "NAS Dateipfad (leer lassen falls egal)", "Setup Datei", "Beschreibung", and "Komprimierung". The "Komprimierung" field is a dropdown menu currently showing "None". There are three buttons: "Durchsuchen" next to the "Setup Datei" field, and "Speichern" and "Abbrechen" at the bottom.

Setup Name

NAS Dateipfad (leer lassen falls egal)

Setup Datei

Beschreibung

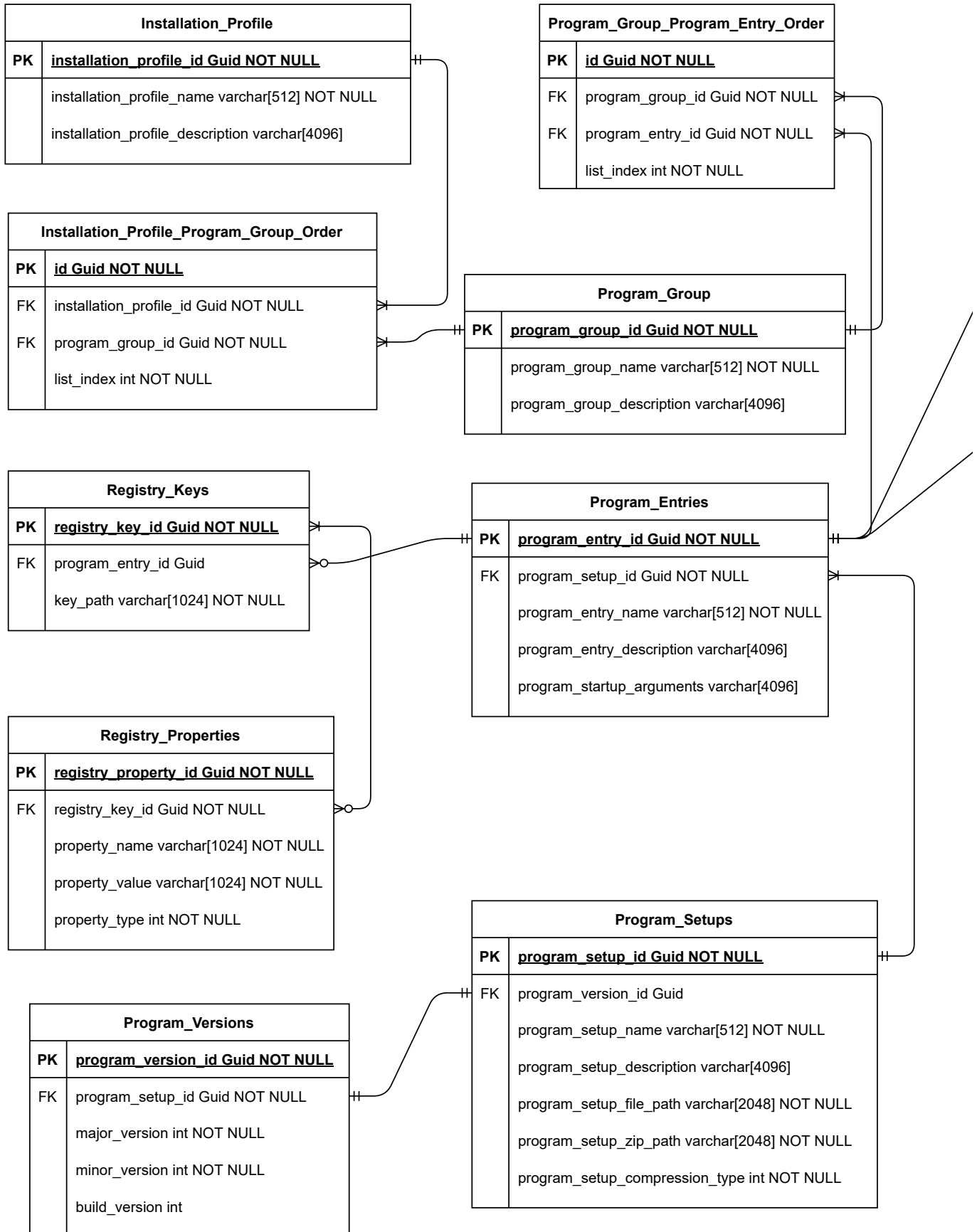
Komprimierung

None ▾

Abbildung 4: Screenshot #2

A.11 Datenbankmodell

Abbildung 5: Datenbankmodell



Folders_To_Check	
PK	<u>folder_id Guid NOT NULL</u>
	folder_path varchar[2048] NOT NULL

Registry_Keys_To_Check	
PK	<u>registry_key_id Guid NOT NULL</u>
	key_path varchar[2048] NOT NULL

Folders	
PK	<u>folder_id Guid NOT NULL</u>
FK	program_entry_id Guid NOT NULL
	folder_path varchar[2048] NOT NULL

Files	
PK	<u>file_id Guid NOT NULL</u>
FK	folder_id Guid NOT NULL
	file_name varchar[512] NOT NULL
	file_size long NOT NULL

Installation_Times	
PK	<u>installation_time_id Guid NOT NULL</u>
FK	program_entry_id
FK	info_id Guid NOT NULL
	time_to_finish interval NOT NULL

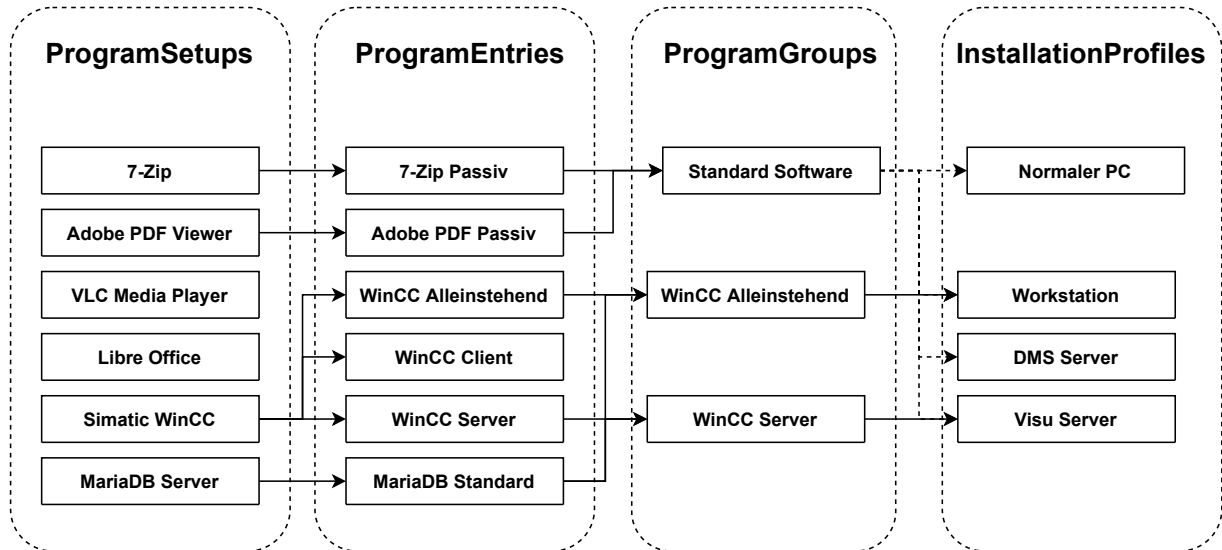
Computer_Infos	
PK	<u>info_id Guid NOT NULL</u>
	model_definition varchar[1024] NOT NULL
	serialnumber varchar[1024] NOT NULL
	memory_mb int NOT NULL

Processor_Infos	
PK	<u>processor_info_id Guid NOT NULL</u>
	processor_name varchar[1024] NOT NULL
	active_cores short NOT NULL
	level_one_cache_mb int
	level_two_cache_mb int
	level_three_cache_mb int
	level_four_cache_mb int

Legende

- + 1 optional
- ++ 1 erfordert
- < 0-n
- < 1-n

A.12 Datenmodell Visualisierung



Hinweis: Alle hier verwendeten Daten sind frei erfunden und dienen ausschließlich der Visualisierung

Abbildung 6: Datenvisualisierung

A.13 Klassendiagramm

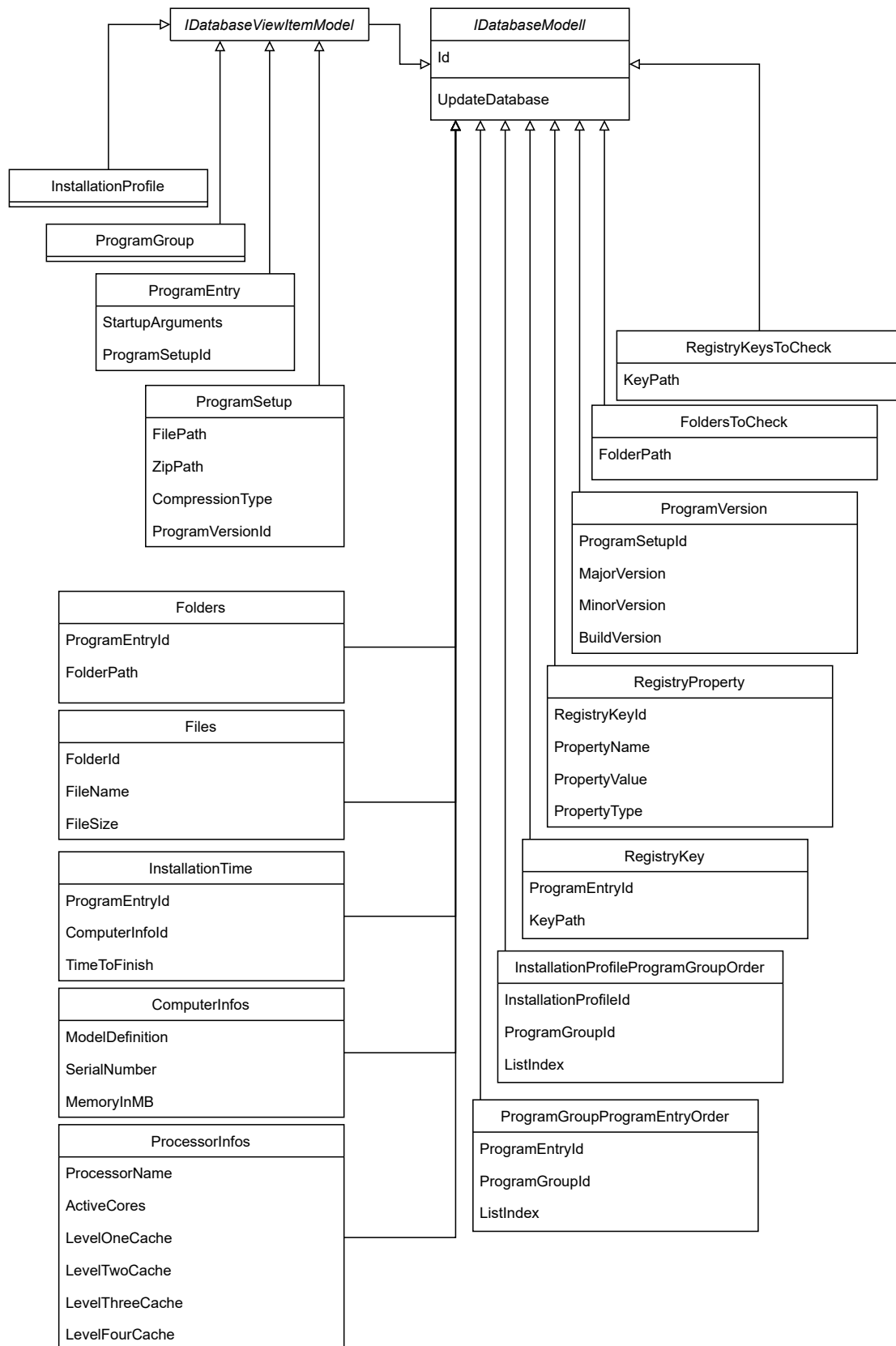


Abbildung 7: Klassendiagramm