IHK Industrie- und Handelskammer Schwarzwald-Baar-Heuberg

Deckblatt zur Dokumentation

Prüflingsnummer *

_____

Vorname **Nachname**

_____

Ausbildungsbetrieb

_____

Ausbildungsort

_____

Zielgruppe (Auftraggeber) für die Präsentation

Persönliche Erklärung

zur Fachaufgabe und zum Report im Rahmen der Abschlussprüfung in den IT-Berufen

Ich versichere durch meine Unterschrift, dass ich die betriebliche Projektarbeit und die dazugehörige Dokumentation selbstständig in der vorgegebenen Zeit erarbeitet habe. Alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen habe, wurden von mir als solche kenntlich gemacht.

Ebenso bestätige ich, dass ich bei der Erstellung der Dokumentation zu meiner Projektarbeit weder teilweise noch vollständig Passagen aus anderen Dokumentationen übernommen habe, die bei der prüfenden oder einer anderen Kammer eingereicht wurden.

_____ _____

Ort, Datum Unterschrift des Prüfungsteilnehmers

Ich habe die obige Erklärung zur Kenntnis genommen und bestätige, dass die betriebliche Projektarbeit einschließlich der Dokumentation in der vorgegebenen Zeit in unserem Betrieb durch den Prüfungsteilnehmer selbstständig ausgeführt wurde.

_____

Ausbilder/-in

Abschlussprüfung Sommer 2020

Fachinformatiker für Anwendungsentwicklung

Dokumentation zur betrieblichen Projektarbeit

Entwicklung von Service-Manager

Vereinfachte Dienst Verwaltung, mithilfe einer intuitiven Oberflächenverwaltung.

Abgabetermin: Schiltach, den 30.05.2020

Prüfungsbewerber:
Christopher Rudolf Karl-Heinz Mogler
Schramberger Str. 13
77761 Schiltach

Ausbildungsbetrieb:
Weiss GmbH Softwarelösungen
Schenkenzeller Str. 163
77761 Schiltach

Inhaltsverzeichnis

Abkürzungsverzeichnis

CMD	Eingabeaufforderung
IIS	Web-Dienst von Microsoft
UI	Benutzerschnittstelle, meist als Grafischeoberfläche
API	Application Programming Interface
MVC	Model View Controller
C#	Programmiersprache der Weiss GmbH Softwarelösungen
ReactJS	React JavaScript
ASP.NET	Active Server Pages .NET
REST	Representational State Transfer
ODBC	Open Database Connectivity
SQL	Structured Query Language
ERP	Enterprise Resource Planning
CRM	Customer Relationship Management

1 Einleitung

1.1 Projektumfeld

Das Projekt wird von Christopher Mogler in der Firma Weiss GmbH Softwarelösungen entwickelt. Sie wurde 1975 von Rolf Weiss mit Hauptsitz in Schiltach gegründet. Zu den ersten Anfängen der Firma wurde für mittelständische Unternehmen Auftragsprogrammierungen auf IBM-Systemen durchgeführt. Im Jahre 1985 wurde die Produktpalette auf ERP-Systeme, für mittelständische und größere Handels bzw. Industrieunternehmen, erweitert.
Die Geschäftsleitung wurde 1998 von Martin Lauble übernommen. Ab dem Jahr 2002 wurde der Schwerpunkt der Software auf das Produkt PowerWeiss für Windows gelegt, welches als CRM-System für Handel, Industrie und Versicherungsagenturen genutzt werden kann.

1.2 Projektziel

Die Firma hat für viele Zusatz Module dazugehörige Dienste. Diese müssen installiert und konfiguriert werden.
Der Support muss sich auf die Kunden-Server drauf schalten und danach die Dienste manuell installieren und konfigurieren. Ziel ist es die Installation, Aktualisierung und Deinstallation zu automatisieren. Deshalb soll ein Programm programmiert werden, welches die anderen Dienste automatisch verwaltet. Über eine Oberfläche kann die Firma Weiss, die einzelnen Module verwalten. Der Dienst auf dem Kunden-Server holt sich die aktuellen Daten über eine API ab.
Die Module werden mit den Daten der API Synchronisiert. Dienste werden nur auf ein Windows-System installiert. Bei neueren Versionen eines Moduls, werden die vorhanden Dateien mit den aktuellen Ausgetauscht. Sollte ein Modul vom Kunden entfernt worden sein, so wird der Service aus dem Windows-System entfernt.

1.3 Projektbegründung

Um Dienste zu installieren muss sich ein Mitarbeiter der Firma Weiss, auf den Server drauf schalten. Die aktuellen Dateien der Dienste kopieren und mit dem Windows bereitgestellten Programm InstallUtil installieren. Da dieses Tool nur über eine CMD verwendet werden kann, ist es sehr fehleranfällig.
Die manuelle Installation ist sehr Zeit aufwendig, deshalb ist eine Automatisierung sehr sinnvoll. Mit einer Automatisierung werden Dienste per Knopfdruck installiert, aktualisiert und deinstalliert oder konfiguriert.

1.4 Projektschnittstellen

Für die Datenspeicherung wird eine SQL-Anywhere-Datenbank verwendet. Der Zugriff auf diese Datenbank, wird über ODBC realisiert.
UI und API werden als Web Applikation bereitgestellt, diese wird über ASP.NET verwaltet. Um ASP.NET zu verwenden wird ein IIS-Dienst von Windows benötigt.
Um Windows-Dienste verwalten zu können, wird auf die Windows-API zugegriffen, über die AdvAPI32 Bibliothek.
UI verwendet als Clientseitiges Framework ReactJS. Für bestimmte Funktionen von ReactJS wird als Steuerung NodeJS eingesetzt.
Das Projekt wurde durch Herrn Richter genehmigt, der Abteilungsleiter der Entwicklung – bei der Firma Weiss. Die Endnutzer sind die Mitarbeiter der Firma Weiss, die geschult werden durch den Autor und als Nachschlagwerk eine Anwenderdokumentation erhalten.

1.5 Projektabgrenzung

Die API und UI wird auf vorhandenen System installiert und ausgeführt. Es wird keine neue Hardware benötigt.
In diesem Projekt werden die Dienste nur bei ausgewählten Kunden installiert, als Pilotierungsprojekt. Nach erfolgreicher Pilotierung, werden die Dienste auch bei anderen Kunden installiert, dies wird aber vom IT-Betrieb übernommen.

2 Projektplanung

2.1 Projektphasen

Das Projekt wurde vom 20.04.2020 bis zum 05.05.2020 realisiert. In dieser Zeit wurde pro Tag ca. Sieben Stunden daran gearbeitet. Die eine Stunde wurde für andere Themen verwendet, z. B. für Fehlerbehebung bei vorhandenen Programmen.

Tabelle 1 zeigt eine grobe Zeitplanung.

Projektphase	**Geplante Zeit**
Planung und Konzeption	20 h
Programmierung	32 h
Abnahme	8 h
Puffer	10h
Gesamt	**70 h**

Tabelle 1: Zeitplanung

Eine detailliertere Zeitplanung findet sich im Anhang **??**: **??** auf Seite ??.

2.2 Abweichungen vom Projektantrag

Im Projektantrag wurde eine Offlineinstallation aufgeführt, diese wurde aber durch die Leitung als nicht relevant angesehen. Daher ist diese nicht etabliert worden.

Um Fehler zu vermeiden, werden die Dateien komplett ausgetauscht, wenn ein Update ansteht. In den Tests konnte kein großer Performance unterschied festgestellt werden.

Vorhanden Windows-Dienste für Module können ohne Änderungen installiert werden.

2.3 Ressourcenplanung

Entwickelt wurde in Schiltach, im Büro des Entwickler-Teams-1.
Von der Firma Weiss, wurden die benötigte Hard- und Software bereitgestellt.

Als IDE kam Visual Studio 2019 im Einsatz. Als Betriebsystem wurde Windows 10 verwendet. Um die Webapplikation auszuführen, wurde IIS verwendet. Als Interpreter für C# wurde .NET Framework verwendet. Dadruch das die UI ReactJS verwendet, wird für bestimmte ReactJS-Funktionen als Steuerung NodeJS verwendet, die Serverseitg läuft.
Als DBSM wurde SQL Anywhere eingesetzt, mit den bereit gestellten Tools: SQL Central und Interactive SQL.
Herr Oehler wurde als organisiatorische Hilfe für den Autor zur Verfügung gestellt.

2.4 Entwicklungsprozess

Da die einzelnen Programme nicht sehr aufwendig waren, wurde das erweiterte Wasserfallmodel eingesetzt.

Die Implementierungsphase wurde in iterativen Zyklen durchgeführt. Ferner ist zu beachten, dass die einzelnen Teile komplett entwickelt wurden. Bei Fertigstellung eines Teil-Projekts, wurde eine Testphase durchgeführt, um zu Prüfen ob die einzelne Funktionen und Schnittstellen Fehler frei funktionieren.

3 Analysephase

3.1 Ist-Analyse

Wenn beim Kunden Dienste installiert werden müssen, so muss ein Mitarbeiter sich auf den Server drauf schalten und diesen installieren. Um Dienste auf Windows-Systeme zu installieren, ist es aktuell nur über das Tool InstallUtil möglich. Dieses Tool hat den großen Nachteil, dass es nur über eine Eingabeaufforderung funktioniert. Daher kann es schnell zu Fehlern kommen, die erst später bemerkt werden.
Ferner ist zu erwähnen, dass der Zugriff auf Kunden-Server nicht immer gewährleistet werden kann: z. B. Fernwartungsanwendungen sind nicht installiert, es existieren aktuelle Internet Probleme.

Es gilt eine Anwendung zu entwickeln, die eine Benutzerfreundliche Oberfläche anbietet und keine Fernwartung auf Kunden-Systeme benötigt.

3.2 Wirtschaftlichkeitsanalyse

Dadruch das ein Mitarbeiter sich nicht mehr auf den Kunden-Server drauf schalten und mögliche Zugriffsberechtigung einfordern muss, wird hier viel Zeit gespart.
Bei einer erst Installation muss nur der Verwaltungsdienst eingerichtet werden, danach können die Applikationen über eine Oberfläche installiert werden.
Wegen der Oberfläche sind Fehler nicht mehr so schnell möglich. Wie bei einer Eingabeaufforderung.

3.2.1 „Make or Buy“-Entscheidung

Es gibt Programme die solche Dienstverwaltungen unterstützen, diese beinhalten aber viele andere Funktionen die nicht verwendet werden. Deshalb wurde bewusst entschieden, dieses Programm im eigenen Haus zu Entwickeln.

3.2.2 Projektkosten

Die Kosten für die Durchführung des Projekts setzen sich sowohl aus Personal-, als auch aus Ressourcenkosten zusammen. Der Autor verdient als Auszubildender 950 € im Monat.

$$8\ \text{h/Tag} \cdot 220\ \text{Tage/Jahr} = 1.760\ \text{h/Jahr} \tag{1}$$

$$950\,€/\text{Monat} \cdot 12\ \text{Monate/Jahr} = 11.400\,€/\text{Jahr} \tag{2}$$

$$\frac{11.400\,€/\text{Jahr}}{1.760\ \text{h/Jahr}} \approx 6{,}477\,€/\text{h} \tag{3}$$

Es ergibt sich also ein Stundenlohn von 6,477 €. Die Durchführungszeit des Projekts beträgt 70 Stunden. Für die Nutzung von Ressourcen[1] wird ein pauschaler Stundensatz von 15 € angenommen. Für die anderen Mitarbeiter wird pauschal ein Stundenlohn von 30 € angenommen. Eine Aufstellung der Kosten befindet sich in Tabelle 2 und sie betragen insgesamt 2.663,39 €.

Vorgang	Zeit	Kosten pro Stunde	Kosten
Entwicklungskosten	70 h	6,477 € + 15 € = 21,477 €	1.503,39 €
Fachgespräch	3 h	30 € + 15 € = 45 €	135 €
Abnahmetest	1 h	30 € + 15 € = 45 €	45 €
Anwenderschulung	25 h	30 € + 15 € = 45 €	1.125 €
			2.663,39 €

Tabelle 2: Kostenaufstellung

3.2.3 Amortisationsdauer

Bei einer Zeiteinsparung von 30 Minuten am Tag für jeden der 10 Anwender und 220 Arbeitstagen im Jahr ergibt sich eine gesamte Zeiteinsparung von

$$10 \cdot 220 \text{ Tage/Jahr} \cdot 30 \text{ min/Tag} = 66.000 \text{ min/Jahr} \approx 1.100 \text{ h/Jahr} \tag{4}$$

Dadurch ergibt sich eine jährliche Einsparung von

$$1.100\text{h} \cdot (30 + 15)\,€/\text{h} = 49.500\,€ \tag{5}$$

Die Amortisationszeit beträgt also $\frac{2.663{,}39\,€}{49.500\,€/\text{Jahr}} \approx 0,054$ Jahre $\approx 2,4$ Wochen.

Das Projekt hat sich also innerhalb von zwei bis drei Wochen amortisiert.

3.3 Anwendungsfälle

3.3.1 Installation

Wenn der Kunde einen Dienst benötigt, wird über die Oberfläche der benötigte Dienst ausgewählt mit der passenden Version. Es wird die Konfiguration angepasst. Beim Speichern werden die Daten in die Datenbank gespeichert. Der Dienst beim Kunden selektiert die Daten und installiert automatisch diesen Dienst mit der gelieferten Konfigurationsdatei.

[1] Räumlichkeiten, Arbeitsplatzrechner etc.

3.3.2 Versionsänderung

Sollte beim Kunden eine neue oder andere Version installiert sein, wird dies wieder über die Oberfläche gesteuert. Die Konfigurationsdatei wird mit der neuen Konfiguration ausgetauscht, nach dem die Daten der Konfig-Datei angepasst wurden. Alle Änderungen werden wieder in die Datenbank der Firma Weiss gespeichert. Über die Web-Schnittstelle bekommt der Verwaltungsdienst die Änderungen mitgeteilt und aktualisiert den Dienst.
Beim ändern der Dateien von einem Dienst muss der Dienst gestoppt werden, dies wird vom Verwaltungsdienst übernommen. Und beim erfolgreicher Installation wieder gestartet.

3.3.3 Anpassung

Über die Oberfläche wird die Konfiguration angepasst, wenn diese Fehlerhafte oder veraltete Daten enthält. Beim speichern werden die änderungen in die Datenbank der Firma Weiss gespeichert. Der Dienst bekommt die Änderung mit und überschreibt die aktuelle Konfigurationsdatei.
Der Dienst wird davor gestoppt und nach dem austausch wieder gestartet.

3.3.4 Deinstallation

Bei einer Deinstallation wird über die Oberfläche der Dienst entfernt. Der Status in der Datenbank wird als žu löschengeändert. Beim Kunden wird der Dienst gestoppt und aus der DienstListe von Windows entfernt.

3.3.5 Statusmeldung

Die Stati der aktuellen Dienste werden regelmäßig überliefert und in die Datenbank gespeichert. Über die Oberfläche kann in der Kundenansicht die Stati der Dienste gesehen werden.

4 Entwurfsphase

4.1 Zielplattform

Als Datenbanksystem wird SQL Anywhere verwendet, weil es bereits bei jedem Kunden installiert und in der Firma Weiss standardisiert ist.
C# wurde als Programmiersprache ausgewählt, diese ist nämlich sehr Performant. Dazu bietet sie sehr gute Web-Frameworks an, die für die API und UI verwendet werden. Auch sind die benötigten Ressourcen (Runtime, etc.) bereits bei den Kunden vorhanden und müssen nicht mehr nach installiert werden.

Die API und UI werden als Webapplikation realisiert. Die API, weil diese auf den REST-Standard aufbaut und mit HTTP am einfachsten zu realisieren ist. UI als Web-Applikation hat den Vorteil, dass die Wartung schneller und einfacher ist. Es müssen keine Zusatzprogramme, Treiber, Software, etc. installiert, aktualisiert oder gewartet werden. Es wird nur ein aktuellen Browser benötigt. Damit steigt die Benutzer Freundlichkeit. Die Oberfläche ist dazu System-übergreifend verwendbar.
Der Dienst ist als Windows-Dienst programmiert. Dieser Dienst wird nur auf Windows-Server installiert und muss daher keine Systemübergreifende Programmierung vorweisen.

4.2 Architekturdesign

Als Webframework wurde ASP.NET verwendet. Es ist frei verfügbar und Open-Source, kann für gewerbliche Tätigkeiten genutzt werden und ist sehr Performant.
Für die Oberfläche wurde ReactJS verwendet. Es basiert wie ASP.NET auf dem MVC Konzept. Auch ist ReactJS eine Clientseitiges Framework, dass bedeutet das die komplette Darstellung der Seite über den Client erstellt wird. Was den Vorteil bietet für schnellere Interaktionen mit der Seite, ohne lange Ladezeiten.

Das MVC-Konzept ist die unterteilung des Quellcodes in drei Komponenten: Model (Datenmodell), View (Ansicht), Controller (Steuerung). Die Zielsetzung ist die Vereinfachung und Wiederverwendbarkeit des Quellcodes. Das Model speichert die Daten, die View präsentiert die Daten und der Controller koordiniert alles.

4.3 Entwurf der Benutzeroberfläche

Die Webseite wurde durch Pencil, einem Mockup-Programm. Siehe Anhang A.2.1: Konzept auf Seite ii.
Da die Oberfläche sich auf Benutzerfreundlichkeit orientiert und das Programm nur für die Firma Weiss entwickelt wurde, wurde auf Coporate Design verzichtet.
Die Oberfläche wurde als Webapplikation realisert, um die Usabilty zu verbessern. Es wird nur ein aktuellen Browser benötigt. Des weiteren kann man unterwegs mit Handy darauf zugreifen, was für die Techniker praktisch ist, die öfters unterwegs sind.

4.4 Datenmodell

4.5 Geschäftslogik

4.6 Maßnahmen zur Qualitätssicherung

5 Implementierungsphase

5.1 Implementierung der Datenstrukturen

Die Datenbanktabellen wurden mit dem Tool, von SQL Anywhere, SQL Central erstellt. Die Tabellen konnten dadurch einfach über eine Grafischeoberfläche erstellt werden, ohne eine Zeile SQL zu schreiben. Mit dem Tool kannn man die Skripte danach automatisch generieren lassen, um diese .

5.2 Implementierung der Benutzeroberfläche

5.3 Implementierung der Geschäftslogik

6 Abnahmephase

7 Einführungsphase

- Welche Schritte waren zum Deployment der Anwendung nötig und wie wurden sie durchgeführt (automatisiert/manuell)?
- Wurden ggfs. Altdaten migriert und wenn ja, wie?
- Wurden Benutzerschulungen durchgeführt und wenn ja, Wie wurden sie vorbereitet?

7.1 Anwendungen

7.1.1 Oberfläche / UI

Die Oberfläche wurde auf dem Cloud-Server der Firma Weiss hochgeladen. Auf diesen Cloud-Server läuft ein Windows-Betriebssystem mit der IIS-Applikation.
Die Webanwendung kann nur aus dem lokalen Netz abgerufen werden, um Unautorisierte Zugriffe zu verhindern. Ausserhalb der Firma zugreifen zu können, wird ein VPN-Tunnel benötigt.

7.1.2 API

Die API läuft wie die Oberfläche auf dem Cloud-Server der Firma Weiss. Die API ist aus dem öffentlichen Netz erreichbar, da die Daten nur über einen SHA-512 Token freigegeben werden.

7.1.3 Verwaltungsdienst

Um die Mitarbeiter zu Schulen, wurde auf einem Testgerät der Firma Weiss dieser Verwaltungsdienst installiert. Dieser ist mit einem Testkunde verknüpft.

In der Schulung wurden die Dienste auf ausgewählten Kunden-Systeme installiert und ausgeführt.

7.2 Schulung

Die Mitarbeiter wurden in einer 2,5 Stündigen Termin geschult. Es wurde die Oberfläche präsentiert und erklärt. Danach ist auf die Installation des Verwaltungsdienst eingegangen.
Die Mitarbeiter mussten selbst auf Kunden-Systeme den Verwaltungsdienst installieren und konfigurieren.
In den letzten 10min ist man auf Fragen und anregungen eingegangen.

8 Dokumentation

9 Fazit

9.1 Soll-/Ist-Vergleich

9.2 Lessons Learned

Der Autor konnte lernen wie eine richtige REST-API aufgebaut wird. Mit den HTTP-Stati und Authenfizierungstokens. Auch wie man über C# auf die Windows-API zugreifen kann und darüber Windows-Dienste installieren kann. Wie man DLLs in C# einbindet und die jeweiligen Funktionen importiert.
Wie man von Grund auf eine Webapplikation entwickelt und diese den Endnutzern erklärt und schult. Aus einer Idee ein komplett strukturiertes Projekt; mit Dokumentation, Zeitmanagment, Ressourcenplanung

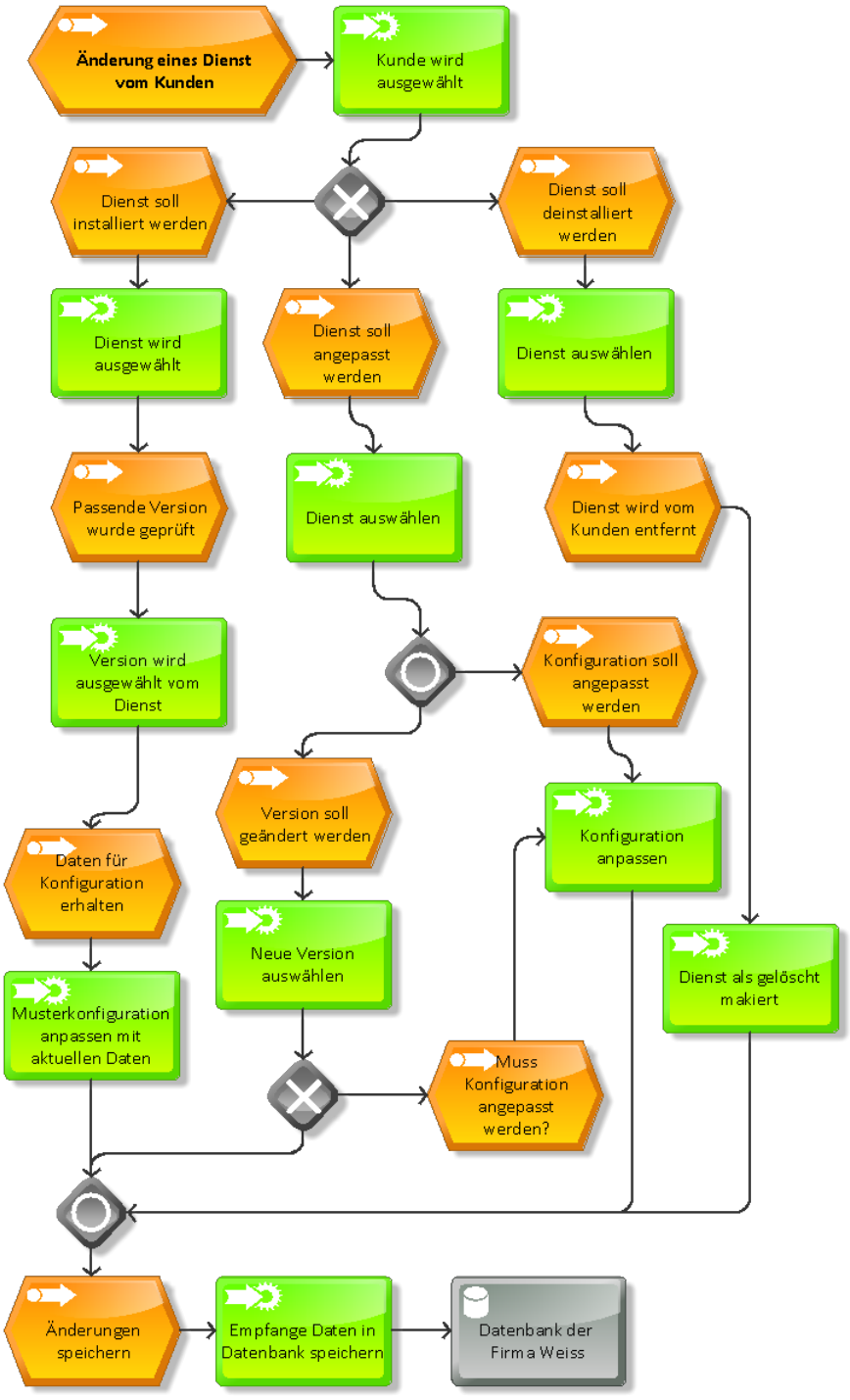
9.3 Ausblick

Der Autor wird nach der Ausbildung die Firma Weiss verlassen, dass Produkt wird an das Entwickler-Team-1 übergeben.
Geplante Erweiterungen sind Fernsteuerung der Dienste: z. B. Stoppen, Starten, Neutstarten.

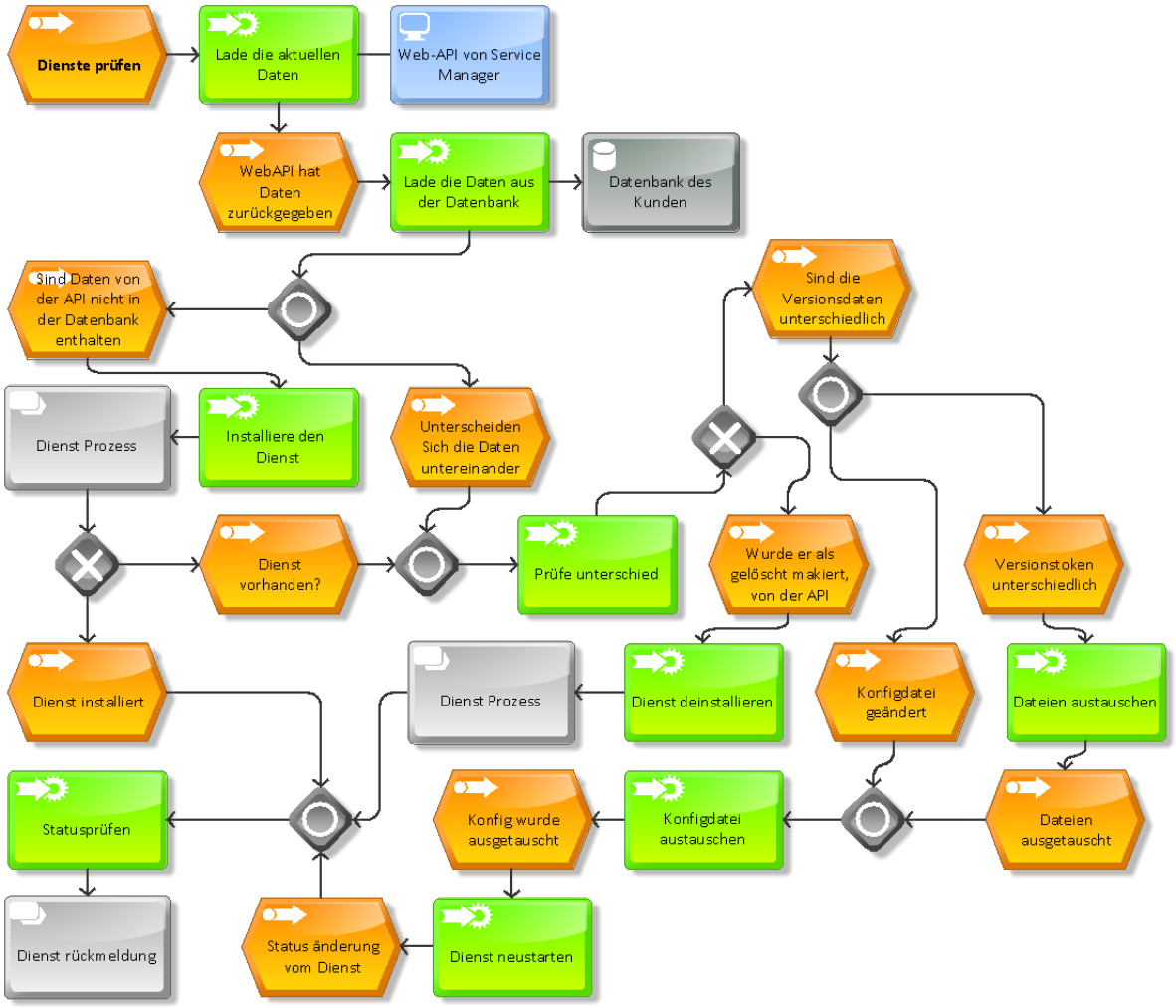
A Anhang

A.1 Anwendung

A.1.1 Verwaltung von Diensten



A.1.2 Dienstprüfung



A.2 Oberfläche

A.2.1 Konzept

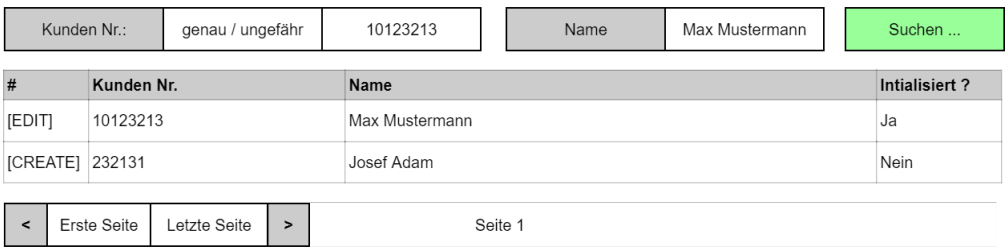


Abbildung 1: Kundenliste

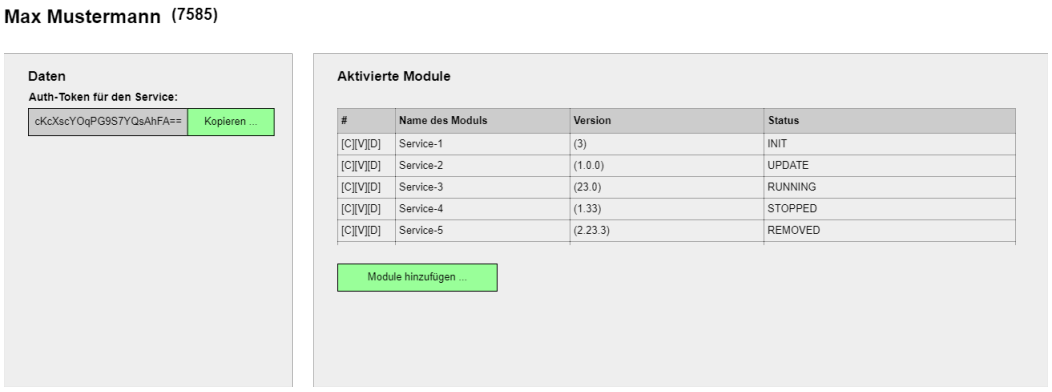


Abbildung 2: Kundenansicht

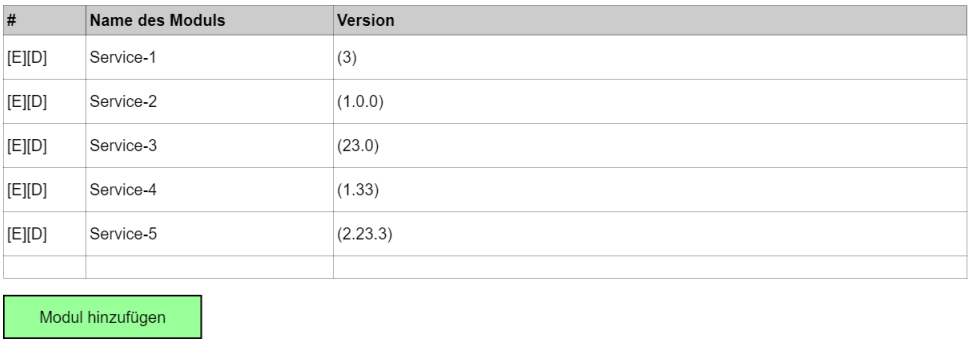


Abbildung 3: Dienstliste

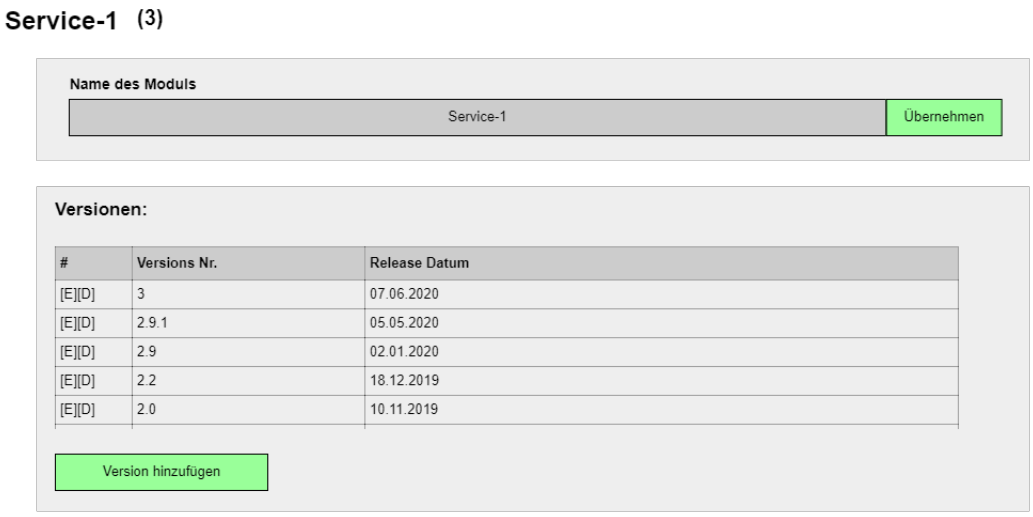


Abbildung 4: Dienstansicht

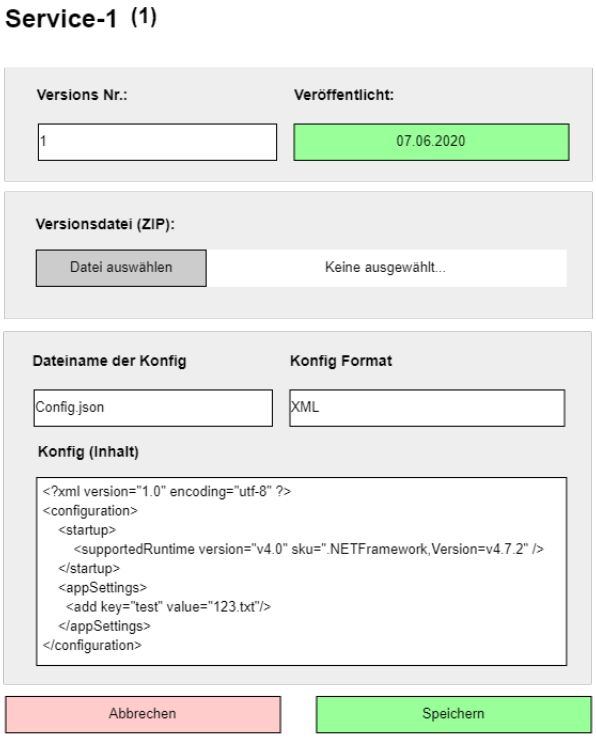


Abbildung 5: Versionseditor

A.2.2 Echtsystem

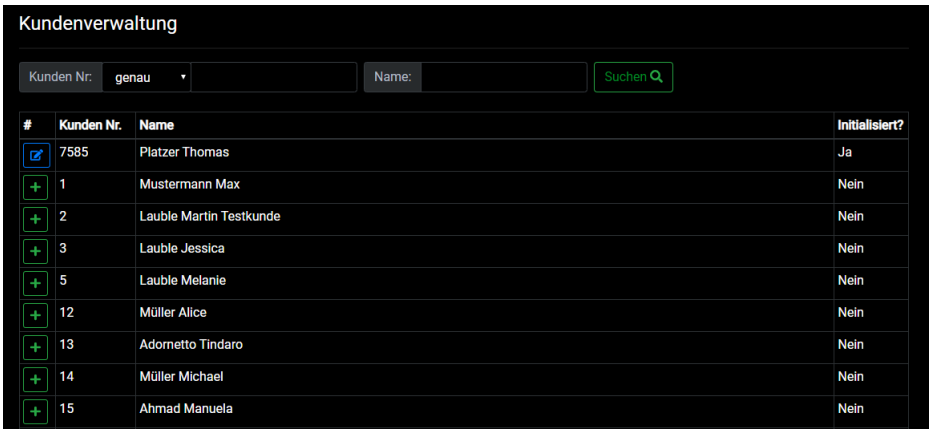


Abbildung 6: Kundenliste

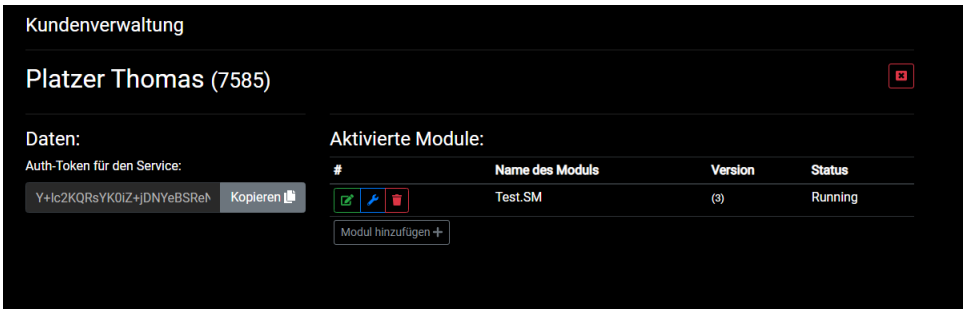


Abbildung 7: Kundenansicht

Abbildung 8: Dienstliste

Abbildung 9: Dienstansicht

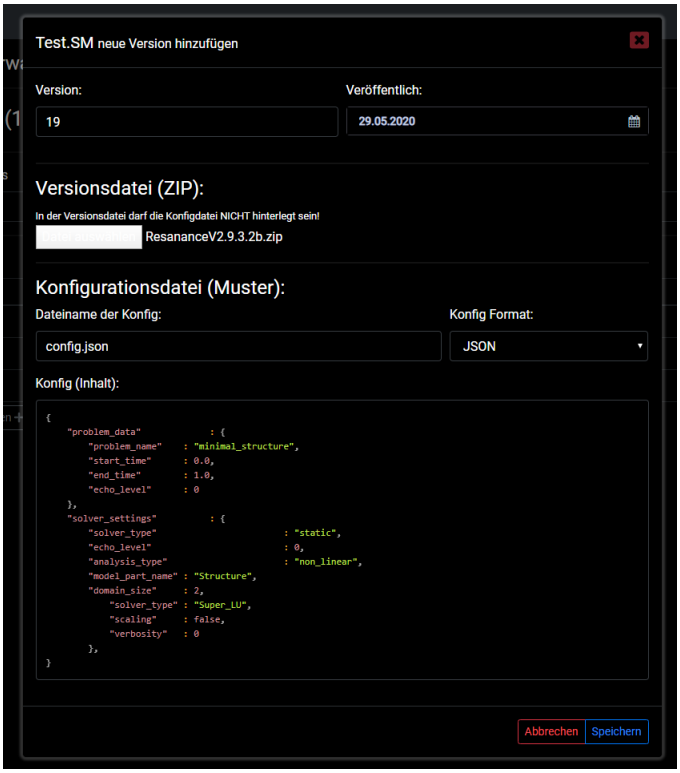


Abbildung 10: Versionseditor

A.3 Klassendiagramme

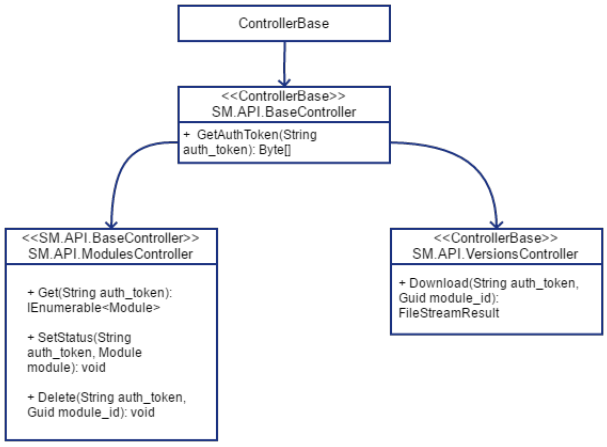


Abbildung 11: Controller-Klassen der API Anwendung

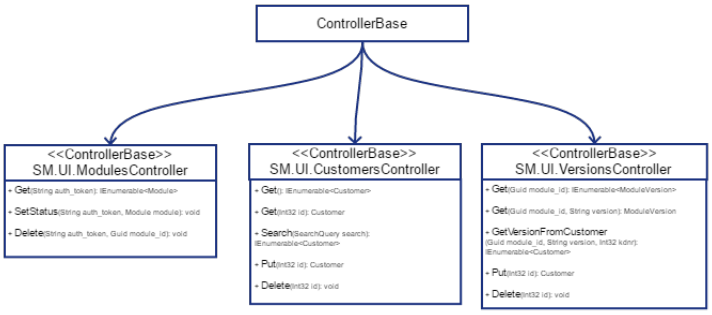


Abbildung 12: Controller-Klassen der Oberflächen/UI Anwendung