HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

Stücklisten-Management für Maschinen

Studienarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Herbstsemester 2017/2018

Autoren: Fabian Gübeli
Luca Salzani

Betreuer: Prof. Dr. Daniel P. Politze

Inhaltsverzeichnis

Abstract

Meldet heute eine Maschine ein kritisches Problem so werden zunächst erste Daten zum Fall erhoben. Ist das Problem komplexer, müssen Servicetechniker meist zweimal vor Ort gehen um dieses zu lösen da die benötigten Ersatzteile nicht vorher bekannt sind.

Bei einem Serviceeinsatz wird oft auch die Stückliste einer Maschine im Feld verändert und stimmt dann nicht mehr mit der ursprünglich ausgelieferten überein. So verliert ein Maschinenhersteller nach und nach das Wissen über den aktuellen Stand seiner Produkte. Mittels einer Softwarelösung sollen diese Änderungen nun während eines Serviceeinsatzes erfasst und entsprechend weiterverarbeitet werden können, sodass stets ein digitales und aktuelles Abbild der Maschinen im Feld zur Verfügung steht.

Als Beispiel für den Software Prototyp wurde eine Stückliste des an der HSR entwickelten Legoroboters zur Verfügung gestellt. Diese Stückliste ist in ihrer Struktur sehr ähnlich wie solche die im produktiven Umfeld eingesetzt werden. Basierend auf diesen Daten sollte die Applikation entwickelt und entsprechende Showcases definiert werden.

Die Webapplikation wurde mit dem Frontend Framework Angular unter Berücksichtigung der Material Design Richtlinien erstellt um eine benutzerfreundliche und intuitive Oberfläche zu gestalten. Dabei wurde der Schwerpunkt auf ein responsive Design gelegt. Somit können der Servicetechniker mit einem Smartphone und der Servicemanager im Büro die Software gleichermassen benutzen. Das Frontend interagiert mit einer mit ASP.NET Core programmierte REST-Schnittstelle welche durch seine Erweiterbarkeit sowie der Portierbarkeit auf andere Plattformen punktet. Es wurde bewusst eine REST-Schnittstelle verwendet um weiteren Applikationen den Zugriff auf die API zu ermöglichen. Bei der Datenhaltung wurde das Entity Framework in Verbindung mit einer Microsoft SQL Server Datenbank eingesetzt. Diese bietet eine unkomplizierte Einbindung der ASP.NET Core API.

Der erarbeitete Prototyp legt den Grundstein für weitere Arbeiten auf dem Gebiet der Maschinenlebenszyklen. Die Software ist in der Lage anhand vier definierter Showcases die Funktionalitäten aufzuzeigen. Das zugrundeliegende System ist bereit für weitere Anforderungen und kann mit zusätzlichen API Endpunkten erweitert werden. Auch Anpassungen am Datenmodell sind möglich. Im aus Komponenten bestehenden Frontend ist es denkbar, weitere Ansichten darzustellen und dabei auch bestehende wiederzuverwenden.

Management Summary

Ausgangslage
Meldet heute eine Maschine ein kritisches Problem so werden zunächst erste Daten zum Fall erhoben. Ist das Problem komplexer, müssen Servicetechniker meist zweimal vor Ort gehen um das Maschinenproblem zu lösen da die benötigten Ersatzteile nicht vorher bekannt sind.

Bei einem Serviceeinsatz wird daher oft auch die Stückliste (= BOM / *engl. Bill of Materials*) einer Maschine im Feld verändert und stimmt dann nicht mehr mit der ursprünglich ausgelieferten überein. Kommen solche Einsätze häufiger vor, so verliert ein Maschinenhersteller nach und nach das Wissen über den aktuellen Stand seiner Produkte. Spätestens bei der Berechnung von Ausfallwahrscheinlichkeiten kann dies zu einem Problem werden. Aber auch für das Erstellen individueller Leistungsangebote ist die genaue Kenntnis über die Situation im Feld sehr wichtig.

Mittels einer Softwarelösung sollen diese Änderungen nun während eines Serviceeinsatzes erfasst und entsprechend weiterverarbeitet werden, sodass stets ein digitales und aktuelles Abbild der Maschinen im Feld zur Verfügung steht.

Vorgehen
Die Webapplikation wurde mit Fokus auf eine benutzerfreundliche und intuitive Oberfläche entwickelt. Beim Frontend wurde der Schwerpunkt auf die Mobileansicht gelegt damit diese durch den Servicetechniker mit einem Smartphone benutzt werden kann. Durch das Design kann die Applikation aber auch vollständig am Desktop benutzt werden. Das Frontend benutzt eine Schnittstelle, welche durch seine Erweiterbarkeit sowie Portierbarkeit auf andere Plattformen punktet. Es wurde bewusst eine Schnittstelle verwendet welche die Integration weiterer Anwendungen vereinfacht.

Ergebnis
Die Anwendung zeigt auf, wie sie einen Servicetechniker bei der Arbeit unterstützen kann und gleichzeitig die Situation im Feld aktuell gehalten wird. Dies wird dadurch erreicht, dass eben genau durch die Kenntnis der aktuellen Stückliste ein Serviceeinsatz besser geplant werden kann. Diese proaktive Unterstützung entlastet den Servicetechniker und hilft auch den Maschinenentwicklern da sie sehen, welche Teile besonders fehleranfällig sind und dementsprechend konstruieren können.

Aufgrund der Entscheidung für eine mobile-optimierte Webapplikation kann das Produkt unterwegs mit einem beliebigen Gerät bedient werden.

Die Datenhaltung wurde auf einem Windows Server realisiert. Eine Datenmigration ist ohne grossen Aufwand möglich, da lediglich die Datenbank ausgetauscht werden muss.

Ausblick
Durch das wart- und erweiterbare Design der Software ist es möglich, den Funktionsumfang zu vergrössern. Eine Anbindung an PLM Systeme ist dabei denkbar und macht den manuellen Import von Stücklisten überflüssig.

Die Entwicklung von nativen Mobileapplikationen ist ebenfalls ein Thema. Dabei kann das bestehende Backend beibehalten werden. Dies würde es erlauben, GPS-Daten auszulesen und somit den Einsatz von Technikern zu planen.

Grosses Potential bringt die Einführung eines Rollenkonzepts mit sich. Dabei werden den einzelnen Rollen eigene, auf ihre Bedürfnisse zugeschnittene Ansichten dargestellt. So würde der Servicetechniker zum Beispiel seine Tagesplanung einsehen können. Der Operator sieht, wo die Servicetechniker im Moment unterwegs sind und kann bei einem Notfall den Techniker anrufen, der

am nächsten am Einsatzort ist. Der Systemadministrator hat einen Überblick über alle Objekte, kann diese anlegen und verwalten. Die Grundfunktionen zum Rollenkonzept sind bereits im Backend entwickelt worden.

Eine weitere Möglichkeit ist die Verfügbarkeit des Systems ohne Internetzugang. Dabei kann der Servicetechniker in einer Produktionshalle ohne Internet seine Arbeiten erfassen und sobald die Konnektivität wiederhergestellt ist werden diese übertragen.

1 Einleitung

1.1 Beschreibung der Ausgangslage

Meldet heute eine Maschine ein kritisches Problem so werden zunächst erste Daten zum Fall erhoben. Ist das Problem komplexer, müssen Servicetechniker meist zweimal vor Ort gehen, um das Maschinenproblem zu lösen. Einmal, um das Problem aufzunehmen und die erforderlichen Massnahmen zu bestimmen. Dann ein weiteres Mal, um mit den erforderlichen Ersatzteilen, die Maschine wieder in Betrieb zu nehmen.

Bei einem Serviceeinsatz wird daher oft auch die Stückliste[1] einer Maschine im Feld verändert und stimmt dann nicht mehr mit der ursprünglich ausgelieferten überein. Kommen solche Einsätze häufiger vor, so verliert ein Maschinenhersteller nach und nach das Wissen über den aktuellen Stand seiner Produkte. Spätestens bei der Berechnung von Ausfallwahrscheinlichkeiten kann dies zu einem Problem werden. Aber auch für das Erstellen individueller Leistungsangebote ist die genaue Kenntnis über die Situation im Feld sehr wichtig.

Mittels einer Softwarelösung sollen diese Änderungen nun während eines Serviceeinsatzes erfasst und entsprechend weiterverarbeitet werden, sodass stets ein digitales und aktuelles Abbild der Maschinen im Feld zur Verfügung steht.

Zur erfolgreichen Bearbeitung dieser Aufgabenstellung soll zunächst eine Anforderungsanalyse durchgeführt, sowie die Vorarbeit analysiert werden. Anschliessend soll das Backend gemäss den Anforderungen entwickelt werden. Dieses stellt eine Schnittstelle bereit welches von einer gleichzeitig zu entwickelnden Frontend Applikation genutzt werden soll. Diese Applikation muss von einem Mobilgerät genutzt werden können.

1.2 Vorarbeit

Im Frühjahrsemester 2017 wurde bereits eine von Prof. Dr. Daniel P. Politze ausgeschriebene Bachelorarbeit mit dem Titel „Entwicklung einer Mobile-App für Service-Einsätze“ durchgeführt. Dabei ging es um die Entwicklung einer ähnlichen App mit dem Ziel für ein Problem, aus vorherigen Problemen und Lösungen, Lösungsvorschläge anzuzeigen. Dies mit dem Ziel, dass die Lösungssuche vereinfacht wird. Diese soll die Grundlage für die zu erarbeitende Applikation darstellen. Die damals erstellte Bachelorarbeit wurde uns zur Verfügung gestellt. Eine Analyse dieser befindet sich in der Problemanalyse.

1.3 Vision

Die Vision ist es, einem Unternehmen welches Maschinen vertreibt und unterhält die Möglichkeit zu bieten, alle Stücklisten ihrer Maschinen im Feld aktuell zu halten. Dies löst das Problem, dass nach einer gewissen Zeitspanne nachdem die Maschine das Werk verlassen hat niemand mehr weiss was genau in den Maschinen verbaut ist. Somit können diese Daten dazu genutzt werden, Analysen über die Standzeit von bestimmten Teilen zu erstellen. Auch werden präventive Serviceeinsätze möglich, wenn ein Maschinenteil die Betriebszeit erreicht hat. Durch eine einfache Importfunktion kann die Stückliste bei der Erfassung der Maschinentypen direkt aus dem PLM in das System geladen werden. Die Nachführung der Stückliste wird jeweils vor Ort durch den Servicetechniker durchgeführt werden. Dieser hat meist ein grosses Smartphone oder ein Tablet bei seinem Serviceeinsatz dabei. Aus diesem Grund wird die Applikation hauptsächlich für solche Displaygrössen optimiert. Damit das System auch wirklich genutzt wird ist es wichtig, dass die Applikation einfach zu bedienen ist und nur minimalen Zusatzaufwand bedeutet.

[1] Auch BOM (engl. Bill of materials)

2 Problemanalyse

2.1 Analyse der Vorarbeit

In der Bachelorarbeit, die im Frühlingssemester 2017 durchgeführt wurde, ist die Grundlage für die Studienarbeit gelegt worden. Aus diesem Grund wurde diese Arbeit analysiert, um zu erkennen welche Komponenten in die Studienarbeit übernommen werden können und welche Teilbereiche Anpassungen benötigen um die Arbeit sauber umsetzen zu können.

Die Bachelorarbeit umfasst ein Backend mit einem ASP.NET Core sowie einer Datenbankanbindung an die Microsoft Azure Cloud und zusätzlich ein Microsoft Azure Store Blob für die direkte Speicherung von Bilddateien. Als Frontend wurde Vue.js in Verbindung mit den nötigen Webtechnologien HTML, CSS und JS eingesetzt. Das Backend sowie das Frontend wurde auf der Microsoft Azure Cloud bereitgestellt.

Folgende Schwächen wurden während der Analyse der Vorarbeit entdeckt.

Usability des Frontend
Diese ist nicht sehr intuitiv zu bedienen. Insbesondere das Attribut-Set ist ohne Durchlesen der Dokumentation der Bachelorarbeit nicht verständlich. Ausserdem ist es mühsam einen Service Case auf eine bereits existierende Maschine zu erstellen, da die Maschinen über die Maschinen-ID im Service Case erstellt werden. Dadurch muss bei einem neuen Service Case für jede bestehende Maschine der Servicerapport auf dieselben Attribute-Sets mit Informationen zum Kunden überprüft werden.

Datenmodel
Dieses wurde so modelliert wie ein Arbeitsrapport aufgebaut ist. Jedoch können die Informationen auch noch von anderen Ansichten verwendet werden. Aus diesem Grund müssen sie abstrakter und modularer aufgebaut werden, damit später ein neues Modul einfacher eingebunden werden kann.

So wurde zum Beispiel die Maschine so modelliert, dass diese alleinstehend keinen Informationsgehalt besitzt. Deren Informationen werden nur über die existierenden Fälle sowie Ersatzteile zusammengetragen. Diese Problematik spiegelt sich auch im Frontend wider. Dort sind in der Übersicht zu den Maschinen keine Stammdaten zu der Maschine sowie der Kunden hinterlegt. Dieser Punkt hat im produktiven Umfeld einen sehr grossen Nachteil für alle Benutzer der Applikation und bedeutet erheblichen Mehraufwand.
Zudem können Stammdaten wie Kunden, die bei jeder Maschine auftreten nur über Attribute-Sets hinzugefügt werden. Daraus folgt, dass die Stammdaten nicht für eine weitere Maschine vom selben Kunden benutzt werden können.

2.2 Umfeldanalyse

Das Projekt ist ein kleiner Teil eines sehr komplexen Prozesses. Es befasst sich mit der Gewinnung von Maschinendaten welche in Benutzung sind. Aufgrund der Erfassung dieser Daten können verschiedenste weitere Schritte eingeleitet werden. Zum Beispiel das Erstellen von Offerten aufgrund des Alters bestimmter Teile in der Maschine. Folgende Grafik soll einen Überblick über die Gesamtsituation geben.

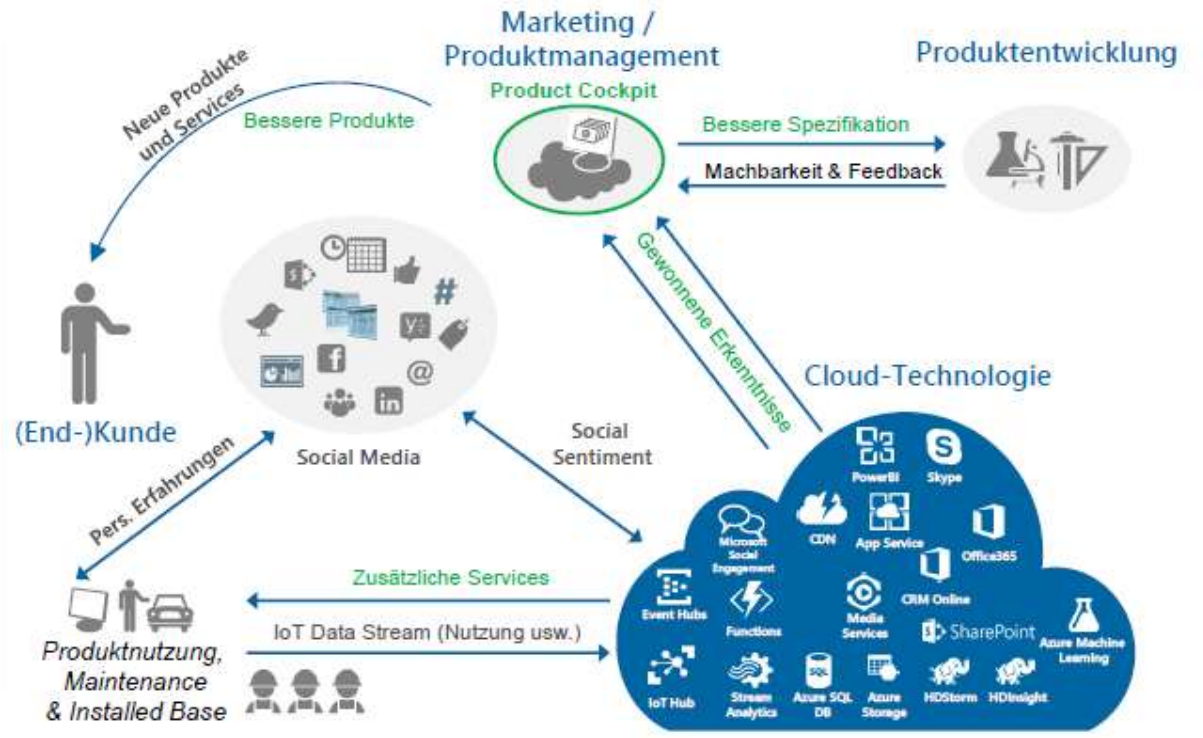

Abbildung 1 - Umfeld, Projektabgrenzung (Bildherkunft: HSR/Microsoft)

Das Projekt befindet sich im unteren linken Bereich bei der „Installed Base“ und dem Datenstrom in die Cloud – oder in diesem Fall ein dedizierter Server. Aus diesen Daten können weitere Erkenntnisse gewonnen werden was bessere Spezifikationen, Produkte und zusätzliche Services zur Folge hat.

2.3 Begriffs- und Rollendefinition

Der *Servicetechniker* beschreibt einen Mitarbeiter der vor Ort eine Installation, Reparatur, Upgrade oder eine andere Arbeit an einer Maschine ausführt. Er erstellt während eines Einsatzes einen *Task Rapport*. Dieser umfasst unter anderem, welche Arbeiten an der Maschine erledigt wurden und welche Maschinenteile der Servicetechniker entfernt oder hinzugefügt hat. Ein Task Rapport gehört immer zu einem *Service Case*. Dieser beschreibt einen Auftrag und beinhaltet das Problem sowie Verweise zu einem Kunden und einer Maschine. Er kann mehrere Task Rapports beinhalten. Nach dem Serviceeinsatz beim Kunden kann ein *Service Rapport* generiert werden, der alle gemachten Arbeiten an einem Tag beim Kunden beinhaltet.

Abbildung 2 - Zusammenhang Service Case, Service Rapport und Task Rapport

Ein *Systemverantwortlicher* verwaltet die Stammdaten der Applikation. Diese umfassen die Maschinentypen, Maschinenteile und Kundendaten. Als *Anwender* der Applikation wird die Zusammenfassung von Systemverantwortlichem und Servicetechniker sowie allen anderen möglichen Rollen verstanden.

2.4 User Stories

US01 Als Servicetechniker möchte ich während einem Serviceeinsatz mobil im System einen Task Rapport erfassen um eine vollständige Änderungshistorie der betreffenden Maschine zu unterhalten.

US02 Als Anwender möchte ich jederzeit abrufen können, welche Maschinenteile in einer Maschine verbaut sind um einen Serviceeinsatz vorzubereiten oder Statistiken zu erstellen.

US03 Als Systemverantwortlicher möchte ich die Stammdaten wie Maschinentypen und Kunden erfassen, bearbeiten und ansehen können um eine Auswertung nach diesen zu ermöglichen.

US04 Als Anwender möchte ich einen Service Case erstellen und verwalten können um die Informationen und Tätigkeiten zu einem Problem zentral zu sammeln.

US05 Als Anwender möchte ich Maschinen erstellen und verwalten können um die Service Cases zentral pro Maschine zu verwalten.

US06 Als Anwender möchte ich einen Kunden erstellen und verwalten können um damit in Verbindung stehende Service Case und Maschinen zu identifizieren.

US91 OPTIONAL: Als Systemverantwortlicher möchte ich im Reporting erkennen, wie lange verbaute Maschinenteile eines bestimmten Artikels bereits im Betrieb sind und in welchen Maschinen dieser verbaut ist.

2.5 Domänenmodell

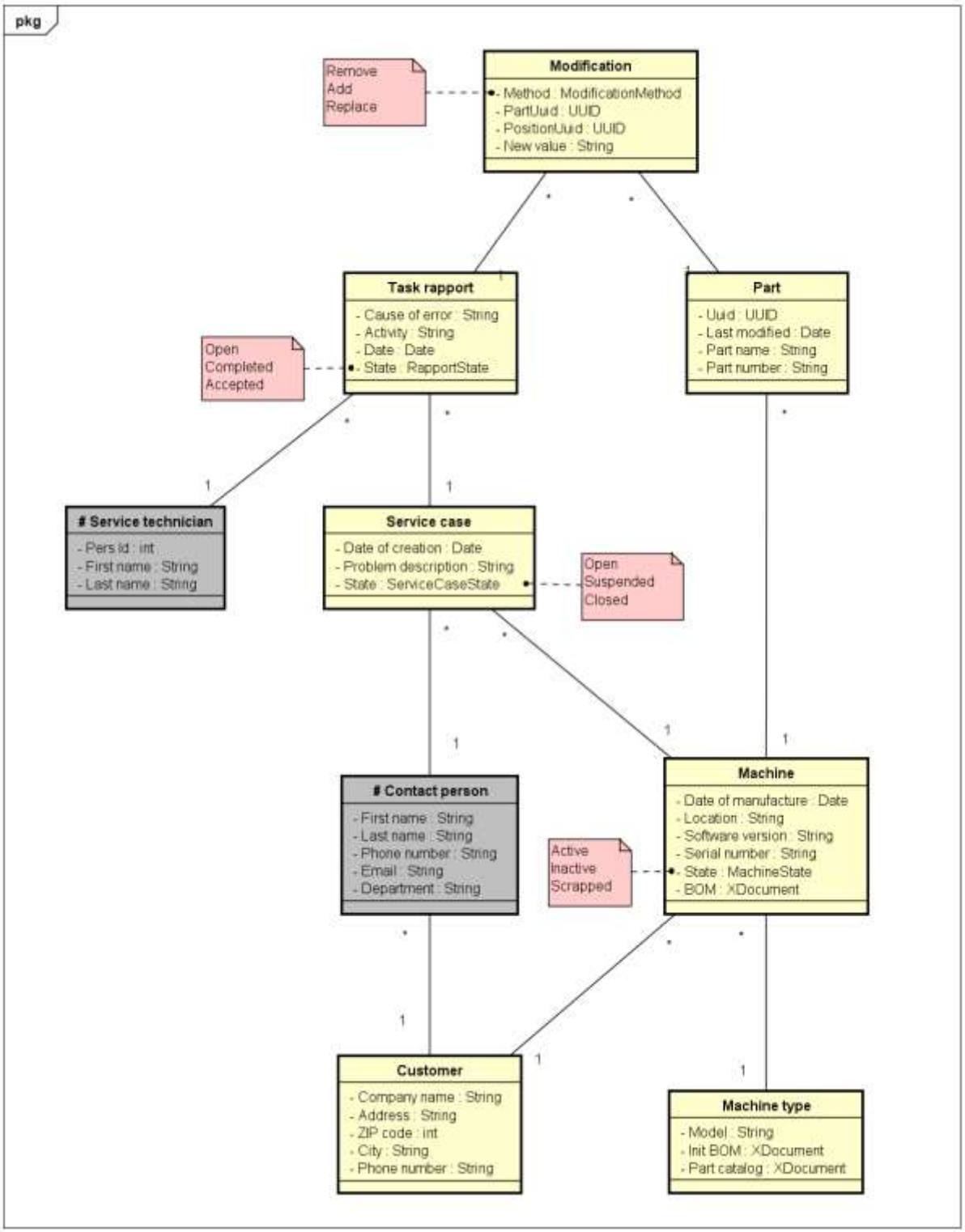

Abbildung 3 – Domänenmodell-Diagramm

In diesem Domänenmodell sind die einzelnen Datentabellen für die Anwendung modelliert. Das Modell bildet die Datenbank ab. Daraus wird die API des Backends abgeleitet. Die grau hinterlegten Tabellen sind für das Projekt optional.

2.6 Funktionale Anforderungen

Funktionale Anforderungen beschreiben die gewünschten Funktionalitäten des Systems, dessen Daten und Verhalten. Sie wurden in Form von Use Cases aus den User Stories abgeleitet. Die Titel sind in Englisch gehalten damit sie für das User Interface nicht mehr übersetzt werden müssen.

2.6.1 Übersicht

Use Case	Ausbaustufe	Vorbedingungen
100 – Create machine type	Bronze	
100a – Import initial bill of materials from file	Bronze	
100b – Import part catalogue from file	Bronze	
110 – Show machine type	Bronze	100
110a – Show initial bill of materials	Silber	100a
110b – Show part catalogue	Silber	100b
120 – Update machine type	Gold	110
200 – Create machine	Bronze	310
210 – Show machine properties	Bronze	200
210a – Show service cases of machine	Bronze	410
210b – Show history of parts	Gold	500a
210c – Show bill of materials	Bronze	100a
220 – Update machine properties	Silber	210
300 – Create customer	Bronze	
310 – Show customer	Bronze	300
310a – Show machines	Bronze	200, 210
310b – Show service case by customer	Bronze	300, 400
320 – Update customer properties	Silber	300, 310
400 – Create service case	Bronze	310
410 – Show service case with all task rapports	Bronze	500
420 – Update service case properties	Silber	
430 – Change service case status	Bronze	
500 – Create task rapport from service case	Bronze	400
500a – Add part	Silber	200, 110b
500b – Remove part	Silber	200
500c – Exchange part	Bronze	200
500d – Update software version	Gold	200
500e – Update location	Gold	200
510 – Read task rapport from service case view	Bronze	500
600 – Create contact person of customer	Gold	
610 – Update contact person of customer	Gold	600
700 – Display service cases in reporting	Gold	400
700a – Filter service cases	Gold	
710 – Get all parts of a specific type with their age	Gold	500a/b/c
720 – Display service rapport	Bronze	510
800 – Create service technician	Gold	
810 – Update service technician	Gold	800

Tabelle 1 – Übersicht der funktionalen Anforderungen, deren Ausbaustufe und Abhängigkeiten

In der ersten Spalte sind die betreffenden Use Cases aufgelistet. Diese werden im nächsten Abschnitt genauer erläutert. Zur Übersicht befindet sich ein Use Case Diagramm im Anhang.

Die zweite Spalte zeigt, in welcher Ausbaustufe der Use Case umgesetzt werden soll. Während der Construction Phase des Projekts werden zuerst die *Bronze*-Use Cases erarbeitet und implementiert,

sodass diese vollständig funktionieren. Wenn diese Funktionalitäten umgesetzt sind wird geplant welche *Silber*-Use Cases implementiert werden. Danach analog die mit *Gold* gekennzeichneten. Die Definition der Ausbaustufen kann am Anfang einer Iteration verändert werden.

In der letzten Spalte ist ersichtlich, welche Abhängigkeiten jeder Use Case hat. Dies ist ein hilfreiches Instrument bei der Planung der Iterationen da auf einen Blick sichtbar ist in welcher Reihenfolge die Funktionalitäten implementiert werden müssen. Auch ist so sichergestellt, dass keine Bronzefunktionalität einen *Gold*-Use Case als Voraussetzung hat.

2.6.2 Use Cases

UC 100 – Create machine type
Wird ein neues Maschinenmodell gebaut oder in den Supportprozess aufgenommen, muss dieser Maschinentyp zuerst in das System aufgenommen werden. Der Systemverantwortliche trägt alle Metainformationen in das System ein. Dieses persistiert die eingegebenen Daten.

Extension UC 100a – Import initial bill of materials from file
Während der Erstellung des Maschinentyps wählt der Systemverantwortliche die aus dem PLM generierte Stückliste im XML Format aus um daraus sämtliche verbauten Teile zu erhalten. Dabei wird für jede Instanz eine eigene XML-Struktur generiert.

Extension UC 100b – Import part catalogue from file
Während der Erstellung des Maschinentyps kann der Systemverantwortliche die aus dem PLM generierte Stückliste im XML Format auswählen, welche alle möglichen Erweiterungen beinhaltet, um daraus den Ersatzteilkatalog zu erstellen.

UC 110 – Show machine type
Ein Systemverantwortlicher lässt sich alle Maschinentypen in einer Liste anzeigen und wählt daraus einen Maschinentyp aus. Der ausgewählte Maschinentyp wird nun mit den Metadaten angezeigt.

Extension UC 110a – Show initial bill of materials
Zum ausgewählten Maschinentyp wird die initiale Stückliste in einer geeigneten Struktur angezeigt.

Extension UC 110a – Show part catalogue
Zum ausgewählten Maschinentyp wird der Ersatzteilkatalog in einer geeigneten Struktur angezeigt.

UC 120 – Update machine type
Ein Systemverantwortlicher ändert die Modellbezeichnung einer Maschine im Falle einer falschen Bezeichnung bei der Erstellung der Maschine oder wenn sich diese während des Betriebs ändert. Ausserdem kann ein neuer initialer BOM oder ein neuer Ersatzteilkatalog hochgeladen werden, welche die vorhandenen überschreiben.

UC 200 – Create machine
Um eine Maschine zu kreieren öffnet der Systemverantwortliche den betreffenden Kunden und signalisiert seine Absicht eine neue Maschine zu erstellen. Danach trägt er die Metadaten der Maschine ein und wählt einen Maschinentyp aus einer Liste. Nach der Bestätigung persistiert das System diese Daten. Dabei wird die initiale Stückliste aus dem Maschinentyp kopiert, mit einer eindeutigen Identifikation gekennzeichnet und für die neue Maschine abgespeichert. Ausserdem werden die Teile in der entsprechenden Tabelle mit dem Erstellungsdatum initialisiert.

UC 210 – Show machine properties
Um sich eine Maschine anzuzeigen öffnet der Anwender den betreffenden Kunden. In einer Liste werden ihm alle Maschinen dieses Kunden dargestellt. Er öffnet die gewünschte Maschine. Das System zeigt alle Metadaten dieser Maschine an.

Extension UC 210a – Show service cases of machine
In der Metadatenübersicht signalisiert der Anwender seine Intention alle Service Cases einer Maschine anzuzeigen. Das System öffnet diese Übersicht und zeigt alle Service Cases der Maschine an.

Extension UC 210b – Show history of activities
In der Metadatenübersicht wird eine Historie aller jemals ausgeführten Arbeiten an der Machine angezeigt.

Extension UC 210c – Show bill of materials
In der Metadatenübersicht ist aufgeführt welche Teile im Moment verbaut sind und wann diese verbaut wurden.

UC 220 – Update machine properties
Um eine Maschine zu bearbeiten öffnet der Anwender den betreffenden Kunden. In einer Liste werden ihm alle Maschinen dieses Kunden dargestellt. Er öffnet die gewünschte Maschine. Das System zeigt alle Metadaten dieser Maschine an. Diese können nun bearbeitet werden. Nach der Bestätigung persistiert das System diese Daten.

UC 300 – Create customer
Der Systemverantwortliche signalisiert die Absicht einen neuen Kunden zu erstellen. Er trägt alle Metainformationen des Kunden in das System ein. Dieses persistiert die eingegebenen Daten.

UC 310 – Show customer
Nach der Auswahl eines Kunden werden alle Daten wie Adresse oder Kontaktinformationen angezeigt.

Extension UC 310a – Show machines
In der Kundendetailansicht wird dem Anwender vom System auch eine Liste aller Maschinen des Kunden angezeigt.

Extension UC 310b – Show service case by customer
In der Kundendetailansicht signalisiert der Anwender seine Intention alle Service Cases des Kunden anzuzeigen. Das System öffnet diese Übersicht und zeigt alle Service Cases des Kunden an.

UC 320 – Update customer properties
Nach der Auswahl eines Kunden werden alle Metadaten angezeigt. Diese können bearbeitet und gespeichert werden. Das System persistiert die eingegebenen Daten.

UC 400 – Create service case
In der Maschinendetailansicht signalisiert der Anwender die Absicht einen neuen Service Case zu erstellen. Er erfasst das Problem und nach dem Speichern setzt das System den entsprechenden Status, das Erstellungsdatum und persistiert die Daten.

UC 410 – Show service case with all task rapports
In einer Service Case Übersicht wählt der Anwender den Service Case aus der ihn interessiert und öffnet ihn. Das System präsentiert eine Detailansicht mit allen Metadaten und allen Task Rapports in einer Liste.

UC 420 – Update service case properties
In der Service Case Detailansicht bearbeitet der Anwender die Metadaten des Service Cases und speichert diese. Das System persistiert die Daten.

UC 430 – Change service case status
In der Service Case Detailansicht kann der Servicetechniker den Status verändern nachdem neue Informationen zum Service Case bekannt wurden oder er den Service Case abgeschlossen hat. Das System persistiert diese Informationen.

UC 500 – Create task rapport from service case
Der Servicetechniker ist vor Ort und erstellt einen Task Rapport indem er in der Service Case Detailansicht diese Absicht signalisiert. Er trägt die Fehlerursache sowie seine Tätigkeiten ein. Nach dem Speichern setzt das System das Erstellungsdatum und persistiert die Daten.

Extension UC 500a – Add part
Im Task Rapport fügt der Servicetechniker neue Teile in die Maschine ein. Er dokumentiert das in seinem Task Rapport. Nach dem Bestätigen wird die Stückliste der Maschine entsprechend angepasst. Das System persistiert diese Daten.

Extension UC 500b – Remove part
Im Task Rapport dokumentiert ein Servicetechniker das Entfernen eines Maschinenteils. Nach dem Bestätigen wird die Stückliste der Maschine entsprechend angepasst. Das System persistiert diese Daten.

Extension UC 500c – Exchange part
Im Task Rapport dokumentiert ein Servicetechniker das Austauschen eines Maschinenteils. Nach dem Bestätigen wird die Stückliste der Maschine entsprechend angepasst. Das System persistiert diese Daten.

Extension UC 500d – Update software version
Im Task Rapport dokumentiert ein Servicetechniker das Updaten der Softwareversion der Maschine. Das System persistiert diese Daten.

Extension UC 500e – Update location
Im Task Rapport dokumentiert ein Servicetechniker das Updaten des Standorts der Maschine. Das System persistiert diese Daten.

UC 510 – Read task rapport from service case view
In der Service Case Detailansicht signalisiert der Benutzer die Absicht, einen Task Rapport lesend zu öffnen. Das System zeigt alle Informationen des Task Rapports an.

UC 600 – Create contact person of customer
In der Kundendetailansicht signalisiert der Anwender, dass er eine neue Kontaktperson hinzufügen möchte. Er trägt die Kontaktinformationen ein und speichert diese. Das System persistiert diese Daten.

UC 610 – Update contact person of customer
In der Kundendetailansicht signalisiert der Anwender, dass er Informationen der ausgewählten Kontaktperson ändern möchte. Das System zeigt alle Informationen der Kontaktperson an. Der Anwender ändert und speichert diese. Das System persistiert die Daten.

UC 700 – Display service cases in reporting
Der Anwender öffnet die Service Case Reporting Ansicht und macht den gewünschten Service Case ausfindig. Er signalisiert der Anwendung, dass er einen bestimmten Service Case detaillierter anschauen möchte. Das System zeigt diese Informationen an.

Extension UC 700a – Filter service cases
In der Reporting Ansicht findet der Anwender über einen Kundenfilter, einen Maschinenfilter, einen Maschinentypfilter und einen Datumsfilter den Service Case. Er signalisiert der Anwendung, dass er einen bestimmten Service Case detaillierter anschauen möchte. Das System zeigt diese Informationen an.

UC 710 – Get all parts of a specific type with their age
Der Anwender öffnet die Reporting Ansicht und wählt ein Artikel aus. Das System zeigt ihm alle verbauten Teile dieses Artikels und ihr Inbetriebnahme Datum an. Zusätzlich ist ersichtlich in welcher Maschine der Artikel verbaut ist.

UC 720 – Display service rapport
Der Anwender öffnet die Kundenansicht und wählt das Datum aus. Nun werden aus allen Task Rapports ein Service Rapport erstellt worauf alle Arbeiten des Tages ersichtlich sind.

UC 800 – Create service technician
In der Administrator-Ansicht signalisiert der Systemverantwortliche, dass er einen neuen Servicetechniker hinzufügen möchte. Er trägt die Informationen ein und speichert diese. Das System persistiert diese Daten.

UC 810 – Update service technician
In der Administrator-Ansicht signalisiert der Systemverantwortliche, dass er Informationen des ausgewählten Servicetechnikers ändern möchte. Das System zeigt alle Informationen an. Der Anwender ändert und speichert diese. Das System persistiert die Daten.

2.7 Nicht-Funktionale Anforderungen

In der nachfolgenden Tabelle sind die verwendeten Umgebungen definiert.

Name	Beschreibung des Zustandes
Normal Condition	Alle Services laufen auf dem Server, der Client verfügt über eine aktive Internetverbindung mit einer Geschwindigkeit grösser als 5 Mbit/s zum Server
Offline Condition	Alle Services laufen auf dem Server, der Client hat keine Internetverbindung zum Server. Dies ist zum Beispiel denkbar, wenn der Servicetechniker vor Ort in einer Produktionshalle mit schlechtem Empfang ist.

Tabelle 2 - Betriebsumgebungen

2.7.1 Studienarbeit/Prototyp

Diese Anforderungen gelten für die Studienarbeit und sind zu erfüllen.

Kompatibilität: Die Anwendung ist mit Google Chrome ab Version 56.0.2924 verwendbar, es werden alle UI Elemente auf dem Bildschirm angezeigt.

Erweiterbarkeit: Die Architektur lässt zu, dass neue Module mit ähnlichem Funktionsumfang wie bestehende Module hinzugefügt werden können.

Skalierbarkeit: Es besteht eine Möglichkeit, die Datenquelle des Backends auszuwechseln.

Änderungssensivität: Alle Anwenderfeatures der Applikation sind responsive designed, sodass diese auf Tablets ab 640 Pixel Displaybreite sowie auf dem Desktop bedienbar sind.

Datenintegrität: Das Backend befindet sich jederzeit in einem konsistenten Zustand. Transaktionen unterliegen dem ACID-Prinzip[2].

Bedienbarkeit: Die Applikation soll von einem Servicetechniker nach einer einstündigen Einführung, durch eine mit der App vertrauten Person, selbständig korrekt bedient werden können.

2.7.2 Produktives Umfeld

Folgende Anforderungen sind zusätzlich für den produktiven Einsatz zu beachten. Die Werte können dabei variieren und sollten von einem Systemarchitekten vor der Inbetriebnahme neu definiert werden.

Verfügbarkeit 1: Alle Features sind unter Offline Condition zu 90% der Zeit innerhalb von 5 Sekunden erreichbar.

Verfügbarkeit 2: Alle API-Requests werden unter Normal Condition zu 90% der Zeit innerhalb von drei Sekunden beantwortet. Unter Offline Condition sind die Features per Definition nicht erreichbar. Dabei darf kein Datenverlust eintreten. Der Benutzer wird über die nicht vorhandene Verbindung informiert und kann durch eine Aktion seine Arbeit wieder fortsetzen.

Migration: Es ist einem Systemadministrator möglich, die Datenbasis mit maximal einer Million Datensätzen innerhalb von acht Stunden auf einen anderen Server zu migrieren.

Sicherheit: Nur privilegierte Personen dürfen auf die API zugreifen.

[2] ACID: Atomicity, Consistency, Isolation, Durability

3 Lösungskonzept

3.1 Technologieentscheid Backend

Folgende Technologie Stacks werden genauer evaluiert und auf ihre Machbarkeit geprüft.

Programmiersprache	Framework	Datenbank	Schnittstelle	Deployment
C#	ASP.NET Core	MS SQL	REST	Azure/IIS
Python	Flask	MySQL	REST	Apache

Tabelle 3 - Technologiealternativen Backend

Es wurde für beide Möglichkeiten ein Prototyp entwickelt welche von der Datenbank mit zwei Tabellen eine REST API zur Verfügung stellt. Nach der Evaluation nach den Kriterien Deployment Möglichkeiten, CI-Readyness, dem Zusammenspiel der Komponenten und natürlich der Erfüllung der nicht-funktionalen Anforderungen lagen folgende Resultate vor.

C# und .NET Stack

Es existieren zwei verschiedene Kernkomponenten um eine Web API mit C# zu erstellen, zum einen der ASP.NET Core und zum anderen das ASP.NET Framework. Der Prototype wurde auf der Basis vom ASP.NET Core entwickelt, da dieser eine bessere Leistung, Skalierbarkeit und Cross-Plattform Support gegenüber dem ASP.NET Framework bietet und dies für die Zukunft dieser App von Bedeutung ist [1].

Als Object-Relational Mapper (O/RM) wurde der Entity Framework Core eingesetzt, damit kann die Anbindung an eine Microsoft SQL Server Datenbank realisiert und in einem späteren Verlauf falls nötig einfach wieder ausgewechselt werden.

C# bietet mit dem Visual Studio 2017 viele Vorteile gegenüber Python um sich Arbeit zu ersparen. Mit dem verwendeten Code-First Prinzip ist es möglich die Datenbank anhand der definierten Models zu generieren und falls gewünscht direkt mit Testdaten zu füllen.

Das Erstellen des eigentlichen Restful Web Services (API), welche in ASP.NET als Controller bezeichnet werden, ist mit dem Visual Studio 2017 eine reine Formsache. Es können bestehende Models ausgewählt werden und die CRUD[3] Operationen in Verbindung mit dem Entity Framework Core dazu automatisch generiert werden. Für die API Dokumentation wurde die Middleware Swagger eingebunden, da für ASP.NET Core ein hervorragender Support besteht. Die Dokumentation wird automatisch aus den definierten Controllern generiert und ist somit immer aktuell ohne sich darum noch explizit kümmern zu müssen.

Zusätzlich wurde im Prototyp noch eine im ASP.NET Core enthaltene Autorisierung-Middleware konfiguriert damit aufgezeigt wird, dass diese in einem späteren Projekt erfolgreich eingesetzt und ausgebaut werden kann.

Die Bereitstellung wird über den in Windows Server integrierten Internet Information Services (IIS) sichergestellt. Darüber hinaus ist es möglich den ASP.NET Core ohne Probleme neben dem zukünftigen Frontend gleichzeitig zu betreiben. Als Alternative kann der ASP.NET Core auch auf der Azure Cloud bereitgestellt werden. Diese wird aber nicht verwendet, da die HSR aktuell keine Lizenzen dafür besitzt.

Python

Python ist eine Programmiersprache mit dem Anspruch, einen gut lesbaren, knappen Programmierstil zu fördern. Die Sprache hat ein offenes Entwicklungsmodell und ist open source [2].

[3] CRUD: Create, Read, Update, Delete

Das Microframework Flask für Python stellt die Webfunktionalität zur Verfügung und dient dem Routing der Anfragen sowie dem Zurücksenden von JSON-Antworten über eine REST Schnittstelle.

Um das Backend bereitzustellen braucht es nur einen Webserver. Dabei ist Apache die bekannteste Lösung und zeichnet sich durch hohe Stabilität und einen sehr leichten Core aus.

Durch die Einfachheit von Python im Einsatzgebiet wird ein schlankes Backend erwartet, welche die Anforderungen hinsichtlich Erweiterbarkeit und Änderungssensitivität zu erfüllen vermag. Das Zusammenspiel der Komponenten mit den Erweiterungen Flask und MySQL funktionierte für den Prototypen einwandfrei. Der Umgang mit den Objekten innerhalb von Python jedoch ist nicht ganz trivial und kann bei komplexeren Vorgängen möglicherweise zu einem Problem werden. Was als sehr positiver Punkt heraussticht ist die Einfachheit der Implementation. Die Implementation von drei lesenden und einer schreibenden Route konnte in unter 70 Codezeilen realisiert werden. Dabei eingerechnet auch alle Imports und die Datenbankverbindung. Die Deployment Möglichkeiten und CI-Readyness sind gross da bei dem Build und dem Starten der Applikation ausser dem Paketmanagement keine Abhängigkeiten bestehen. Dabei hilft auch, dass die Applikation selbst sehr einfach gestartet werden kann.

Ein weiterer Negativpunkt ist die Menge an Frameworks und Erweiterungen für Python welche es schwierig macht, den Durchblick zu behalten. Probleme können auf viele verschiedene Arten mithilfe unterschiedlichster Frameworks und Bibliotheken gelöst werden. Dabei die Beste zu finden und den Code sauber zu halten ist sehr zeitintensiv und setzt viel Erfahrung voraus.

Entscheidung
Der Entscheid fiel am Ende auf den *C# .NET-Stack* da dieser alle benötigten Funktionalitäten bietet.

3.2 Technologieentscheid Frontend

Folgende Technologie Stacks werden genauer evaluiert und auf ihre Machbarkeit geprüft.

- Angular
- Vue.js

Mit beiden Frameworks wurde ein Prototyp entwickelt. Dieser ist in der Lage, von der bereitgestellten API zu lesen und Daten darzustellen sowie Daten über Formelemente zu senden. Bewertet wurden die Technologien nach der Erfüllung der nicht-funktionalen Anforderungen und danach wie gut die Frameworks die Problemstellung erfüllen.

Angular 4
Angular 4 oder heute auch nur noch Angular genannt ist ein weit verbreitetes Web Framework für die Erstellung von Single Page Applikationen. Dementsprechend sind auch der Support und das Angebot von eigenen Modulen der Community gross, was für die Arbeit ein sehr grosser Vorteil ist. Angular bietet von Haus aus Support für alle gängigen Webplattformen und es ist sogar eine native Entwicklung möglich [3].

Der Prototype wurde in der Entwicklungsumgebung WebStorm entwickelt. Die Grundlage bot ein neues Projekt aus der Angular CLI. Somit war bereits die Projektstruktur wie empfohlen eingerichtet und es war innerhalb von kurzer Zeit ein Resultat sichtbar.

Bei der Entwicklung haben sich die Typensicherheit durch Typescript und die vorhandenen Projektstrukturen positiv hervorgehoben. Durch Typescript kann sehr sauber der Rückgabetype eines API Requests definiert werden. Dadurch wird die Weiterverarbeitung um einiges erleichtert. Die Projektstruktur ermöglicht es, die Servicekomponente für API Requests sauber zu entkoppeln und in weiteren Komponenten wiederzuverwenden. Zudem können Komponenten für die Darstellung in

drei Teile unterteilt werden (HTML, CSS und Typescript). Dies erhöht die Austauschbarkeit und Erweiterbarkeit massiv.

In der Community lassen sich viele Bibliotheken finden, die die Entwicklungszeit positiv beeinflussen können. So besteht zum Beispiel eine Material Bibliothek zur Verfügung, die die ganzen Standardkomponenten des Material Designs implementiert. Dadurch kann ohne sehr hohen Designaufwand ein sauberes User Interface erstellt werden [4].

Vue.js
Vue.js wurde bereits für die Vorarbeit verwendet. Davon kann nur ein kleiner Teil wiederverwendet werden, da die dort implementieren Komponenten sich stark von denen dieser Arbeit unterscheiden. Vue.js ist ein JavaScript Framework und steht in Konkurrenz mit ähnlichen Frameworks wie Angular oder React. Ein Vorteil ist, dass das Framework mit 18kb sehr klein ist. Ausserdem stellt es standardmässig bereits einen Router zur Verfügung und bietet grosse Flexibilität. Bisher ist es ein nicht sehr bekanntes Framework und daher gibt es neben der doch sehr ausführlichen Dokumentation des Vue.js Teams auch eher wenige Ressourcen.

Das komponentenbasierte Konzept eignet sich gut für die Navigation. Für den Inhalt jedoch müssen alle Komponenten einzeln geschrieben werden. Auch der Datenaustausch und die Koordinierung zwischen einzelnen Komponenten könnten sich schwierig gestalten.

Die Entwicklung des Prototyps funktionierte nur mässig gut, da Vue.js das Verbinden auf eine Schnittstelle nicht automatisch unterstützt. Daher wurden verschiedene Pakete benötigt um überhaupt Web Requests zu senden und die Antwort empfangen zu können.

Entscheidung
Nach der Evaluation wurde *Angular* als Frontendframework festgelegt, da es allen Anforderungen gerecht wird und ausserdem eine Material Bibliothek bietet.

Während der Implementationsphase wurde Angular 5 veröffentlicht. Diese enthält verschiedene Bug fixes und macht Angular noch performanter. Ausserdem wurde der Http-Client überarbeitet was das Konsumieren der API vereinfacht [5]. Daher wurde das Projekt auf Angular 5 aktualisiert.

3.3 Werkzeuge

Als Entwicklungsumgebungen werden *Visual Studio 2017 Enterprise* von Microsoft für das ASP.NET Backend sowie *WebStorm 2017.2* von JetBrains für das Frontend in Angular eingesetzt. Beide eignen sich sehr gut für die Entwicklung der jeweiligen Technologien und bieten viele angenehme Features wie IntelliSense, Dokumentation und statische Codeanalyse. Alle Teammitglieder verwenden dieselben Versionen der Software um unerwarteten Fehlern vorzubeugen.

Im Backend werden *EntityFramework Core* und *ASP.NET Core* von Microsoft eingesetzt. Das Entity Framework erlaubt das Arbeiten mit Datenbankobjekten innerhalb des Backends und ASP.NET Core stellt die API zur Verfügung. Auch wird *Swagger* als API Dokumentationssoftware eingesetzt. Dies hat sich in den letzten Jahren durchgesetzt und bietet angenehme Features wie automatische API Dokumentation und das Absetzen von Requests zu Testzwecken während Integrationstests.

Als unterstützende Werkzeuge fungieren *Git* als Versionskontrollsystem welches keine ernstzunehmende Konkurrenten hat, *Jenkins* als CD/CI-Server welches open source ist und Builds von privaten Repositories unterstützt was andere Anbieter oft nicht können. Weiterhin werden *Astah Community* für UML-Diagramme, *Dropbox* als DMS, *Slack* als Kommunikationstool und die *Office Suite* für die Dokumentenerstellung benutzt.

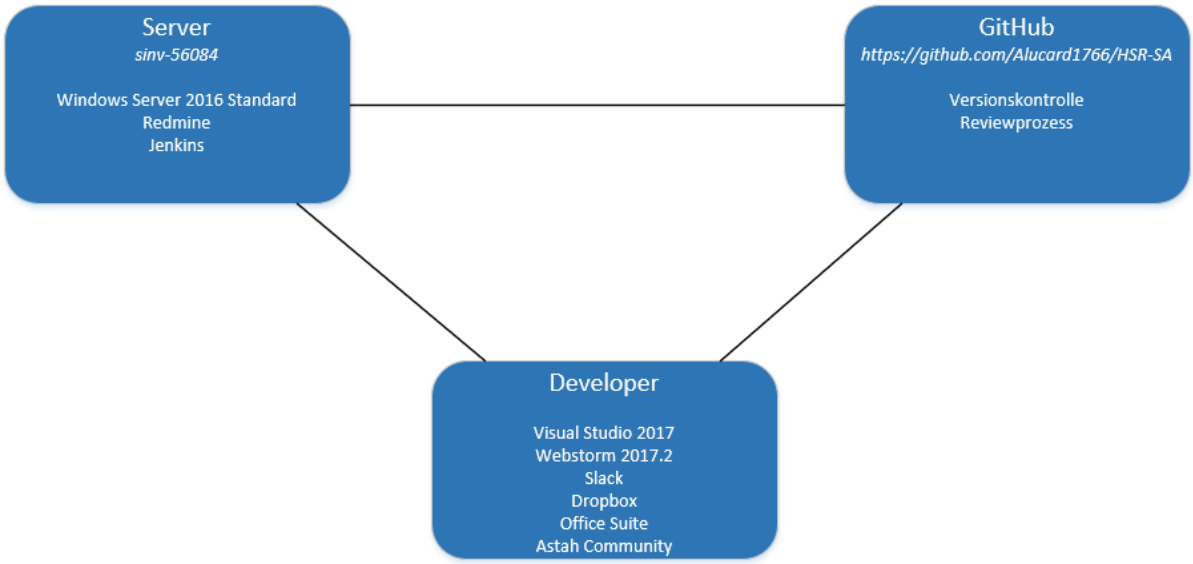

Abbildung 4 – Werkzeuge

3.4 API Beschreibung

Die API ist die Schnittstelle zwischen dem Backend und dem Frontend. Das Backend stellt die benötigten Funktionen zur Verfügung. Die folgenden Endpunkte sind im Backend definiert.

Endpunkt	Optionen	Format	Parameter bei GET/POST Requests
Customers	GET, POST	JSON	
Customers/{id}	GET, PUT	JSON	
Machines	GET, POST	JSON	Optional GET: CustomerId
Machines/{id}	GET, PUT	JSON	
Machines/{id}/Bom	GET	JSON	
ServiceCases	GET, POST	JSON	Optional GET: MachineId und CustomerId
ServiceCases/{id}	GET, PUT	JSON	
TaskRapports	GET, POST	JSON	Optional GET: ServiceCaseId
TaskRapports/{id}	GET, PUT	JSON	
MachineTypes	GET, POST	JSON	
MachineTypes/{id}	GET, PUT	JSON	
MachineTypes/{id}/Bom	GET PUT	JSON XML	
MachineTypes/{id}/PartCatalog	GET PUT	JSON XML	
Modifications	GET, POST	JSON	Optional GET: TaskRapportId
Modifications/{id}	GET	JSON	
Reporting/PartOptions	GET	JSON	
Reporting/PartAgePerPartType	GET	JSON	Optional GET: PartName

Tabelle 4 - API Endpoints

Es fällt auf, dass die Stücklisten jeweils beim Senden vom Client an das Backend als XML übertragen werden. Dies verursacht eine höhere Netzbelastung. Diese kann aber aufgrund der beschränkten Zugriffszahlen vernachlässigt werden. Ein Vorteil ist, dass das Dokument nicht vom Frontend in JSON umgewandelt werden muss um danach vom Backend wieder in XML geparst zu werden. Somit werden sowohl Client wie auch Server entlastet.

Eine vollständige API-Dokumentation wird jeweils immer durch Swagger generiert. Diese ist jeweils unter dem Endpunkt *http:///[BACKENDADRESSE]/swagger* zu finden. Ein Export der Dokumentation zu Projektende ist als HTML in der digitalen Abgabe zu finden.

3.5 XML-Format BOM

Der Aufbau der Stückliste besteht aus Baugruppen, welche weitere Baugruppen oder Einzelteile beinhalten.

Der zur Verfügung gestellte XML-Export besteht aus Part-Typen welche Baugruppen oder Teile sein können. Diese Part-Typen haben eine Beziehung zu einem weiteren Part-Typen. Diese Beziehung ist durch die *PartBom*-Typen definiert. Darauf ist spezifiziert wie viele Teile oder Baugruppen von diesem Typ zu der Beziehung gehören. Beim Export aus dem PLM werden automatisch viele zusätzliche Attribute ins XML übernommen. Diese werden für das Speichern auf ein Minimum reduziert. Benötigt werden lediglich ein Name und eine Artikelnummer.

Beim Laden der Stückliste in die entwickelte Applikation werden die Beziehungen durch eine UUID eindeutig im ganzen System identifiziert. Dies erlaubt es das Alter jedes einzelnen Teiles zu speichern und abzurufen. Dazu muss auch die Quantität aufgeschlüsselt werden und die Beziehungen so viele Male dupliziert werden bevor die UUID vergeben wird [6].

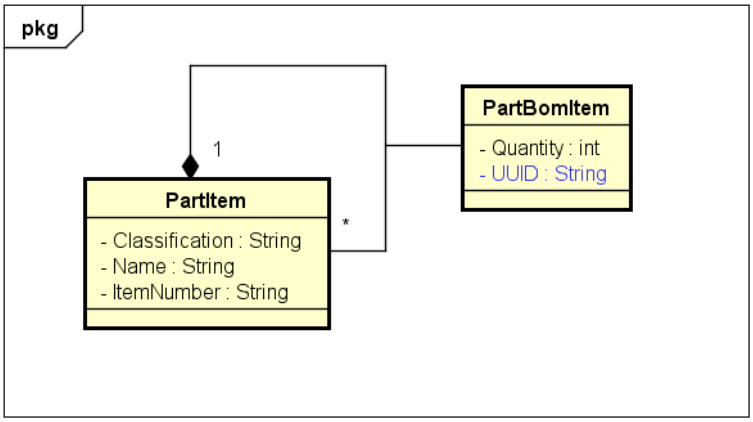

Abbildung 5 - Systematischer Aufbau XML-Export

Die XML-Datei wird direkt im XML-Format in der Datenbank gespeichert. Sollte die BOM abgerufen werden, wird diese geladen und als JSON zum Frontend gesendet.

Das Betriebsalter der Teile wird in der *Part*-Tabelle in der relationalen Datenbank gespeichert. Dort sind alle verbauten Teile erfasst. Dies erlaubt auch bei grossen Datenmengen eine sehr performante Auswertung dieser Daten.

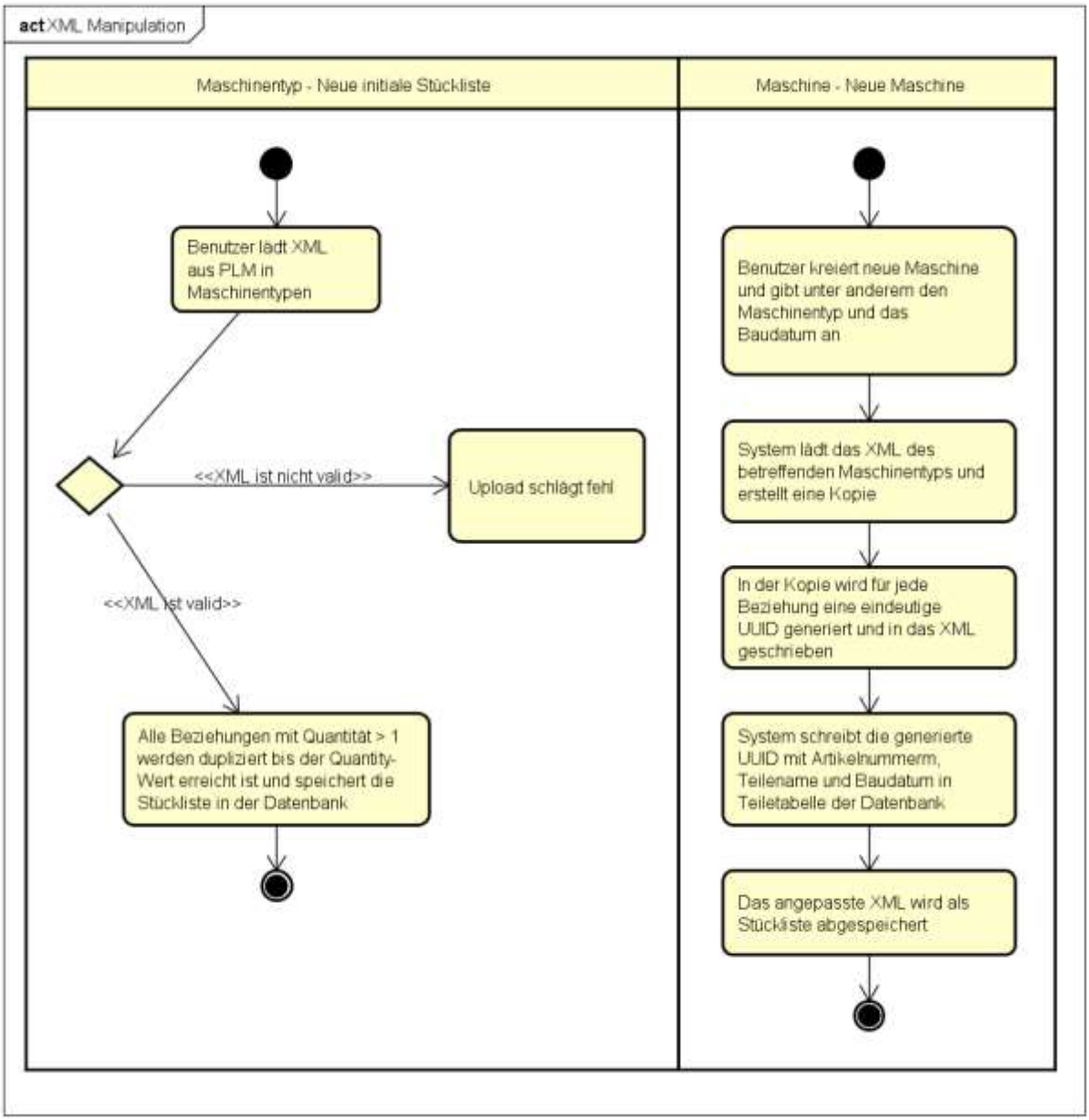

Abbildung 6 - Visualisierung XML Manipulation

Bei einer Manipulation einer Stückliste im Feld kann der Servicetechniker aus einer Baumstruktur die zu manipulierenden Teile wählen. Die Änderungen werden danach anhand der UUID in der Datenbank und im XML nachgeführt.

Die Validität der hochgeladenen XML-Dateien wird über ein XML-Schema (XSD) sichergestellt. XSD ist eine Datenstruktur welche unter anderem dazu entwickelt wurde, die Validität von XML Daten sicherzustellen [7]. Es ist sozusagen ein Bauplan für XML-Dokumente desselben Typs – hier von Stücklisten.

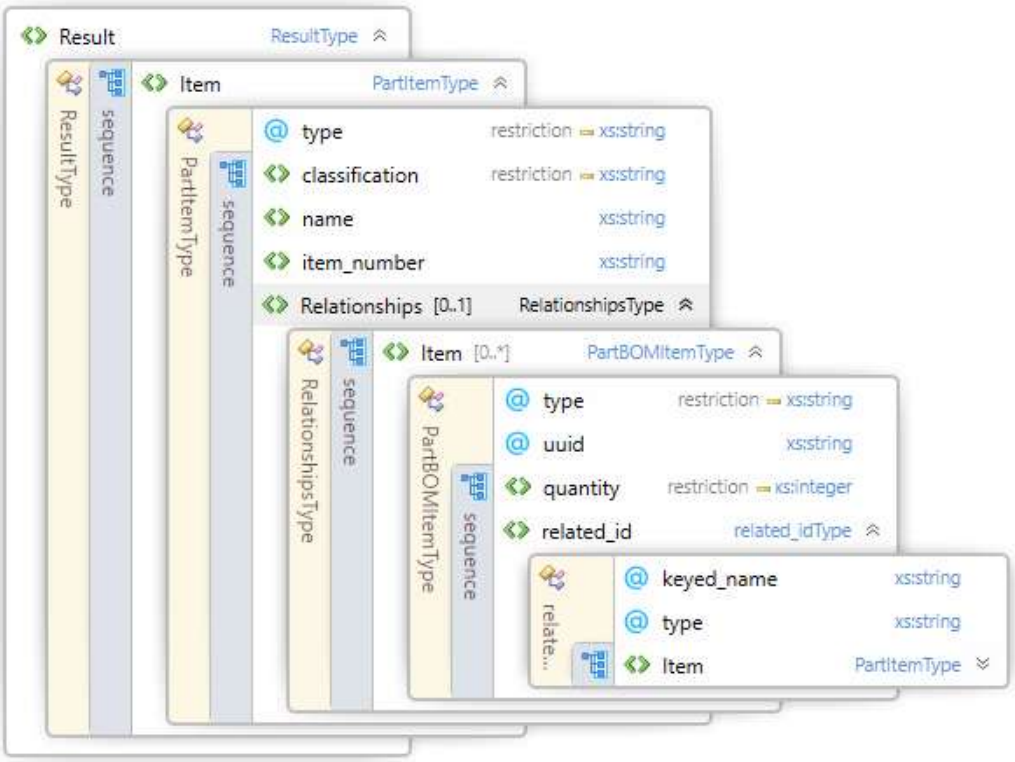

Abbildung 7 - Content Model View XML Schema

In dieser Abbildung ist das zu verwendende XML-Schema zu sehen. Eine Stückliste kann jederzeit im System mithilfe des Schemas validiert werden. Dies passiert immer, wenn eine Stückliste oder ein Ersatzteilkatalog aus der Datenbank geladen wird. So wird jederzeit der konsistente Zustand sichergestellt.

Für die Speicherung der XML-Dateien gibt es verschiedene Möglichkeiten. Da Entity Framework eine relationale Datenbank voraussetzt ist es nicht möglich, ausschliesslich eine NoSQL Datenbank zu verwenden. Dies wäre auch nicht praktisch, da so die Relationen verloren gehen. Die Möglichkeiten welche zur Verfügung stehen sind wie folgt.

- Ablegen einer XML-Datei in einem Ordnerverzeichnis und speichern des Orderpfades in der relationalen Datenbank.
- Speichern der XML-Datei als String in einer NoSQL Datenbank und Verweis auf deren ID in der relationalen Datenbank.
- Speichern der XML-Datei als String in der Relationalen Datenbank um diese jeweils auszulesen.
- Parsen der XML-Datei in eine Node-Leaf Baumstruktur und jedes Element als Entity in der relationalen Datenbank ablegen.

Die erste Variante scheidet aus, da sie keinen Mehrwert zu den anderen Varianten bietet. Bei einer Migration muss auch immer der XML-Ordner kopiert werden. Ein Umbenennen der Datei würde einen inkonsistenten Zustand herbeirufen und die Performance der Applikation leidet unter dem zusätzlichen Filezugriff. Ausserdem müssten Themen wie Berechtigungen und Datensicherheit auch auf das Filesystem übertragen werden. Neuere Microsoft SQL Server Versionen unterstützen den Filestream Typ. Dieser unterstützt das Speichern von Daten in einem NTFS Filesystem, bleibt aber trotzdem abfragbar und durchsuchbar [8]. Da die Applikation eventuell auf Open Source umgebaut werden soll, kommen keine Windowsspezifischen Lösungen in Frage.

Für die zweite Variante mit einer zusätzlichen Datenbank spricht, dass NoSQL Datenbanken auf solche Arten von Daten ausgelegt sind. Ausserdem sind sie leicht skalierbar. Auch dort stellt sich danach die Frage nach der Sicherheit und den Zugriffsrechten. Ausserdem müsste die Applikation nach der Einführung zwei verschiedene Datenbanken managen. Die Performance fällt auch bei dieser Lösung stärker ins Gewicht, da nicht von den Indexfunktionalitäten einer relationalen Datenbank profitieren werden kann.

Das Speichern der XML-Datei ohne Konvertierung in der relationalen Datenbank ist eine sehr pragmatische Lösung. Dies stellt jedoch nicht unbedingt ein Problem dar, da ein Prototyp entwickelt werden soll. Die Suche nach einzelnen Teilen ist, wie in den vorhergehenden Varianten auch, nicht optimal und erfordert einen hohen Aufwand bei der Suche. Beim Auslesen der gesamten Stückliste jedoch ist dies die performanteste Lösung. Die Struktur ist in der XML-Datei gespeichert. Die Eigenschaften der einzelnen Teile können in einer Erweiterung in einer eigenen Tabelle ohne Strukturinformationen gespeichert werden.

Die vierte Variante ist die sauberste, erfordert jedoch auch einen hohen Aufwand bei der Persistierung und dem Auslesen der gesamten Stückliste. Ein grosser Vorteil ist das Speichern von Eigenschaften von einzelnen Teilen direkt im entsprechenden Knoten. Das Verändern der Stückliste würde dann jedoch sehr komplex und fehleranfällig werden, da die Referenzen vertauscht und verändert werden müssen. Vor allem in einer relationalen Datenbank mit Fremdschlüsseleinschränkungen ist dies schwer zu implementieren. Nach Rücksprache mit Prof. Thomas Letsch, welcher die Module Algorithmen und Datenstrukturen sowie Software Engineering 3 doziert, ist die Speicherung von XML direkt anstelle einer geeigneten Datenstruktur in der Datenbank keineswegs „Bad Practice“, sondern bietet auch einzigartige Vorteile.

Der Fokus der Arbeit liegt auf dem Auslesen und Bearbeiten der gesamten Stückliste und nicht auf dem Erstellen von komplexen Statistiken. Aus diesem Grund fiel die Entscheidung auf die dritte Variante. Bei dieser werden XML-Daten direkt in der relationalen Datenbank und einer zusätzlichen Tabelle für die Eigenschaften der einzelnen Teile gespeichert. Anhand dieser können die wichtigsten Daten schnell ausgelesen werden. Einzelne DBMS-Anbieter haben teilweise einen XML-Typ in ihrer Datenbank implementiert [9]. Deren Vorteile überwiegen jedoch nicht den Nachteil der Portierbarkeit auf eine andere Technologie. Mit der Speicherung als String kann das System viel einfacher portiert werden. In einem Produktivsystem kann man sich dann Gedanken über eine Verwendung von einer herstellerspezifischen Lösung machen.

3.6 Testkonzept

Während der Entwicklungsphase wird kontinuierlich getestet. Dabei sollten während der Entwicklung eines Arbeitspakets jeweils Unit Tests für die einzelnen Methoden geschrieben werden. Nach dem Abschluss eines Arbeitspakets werden Integrationstests durchgeführt um zu prüfen ob die Funktionalität durch alle Layer des Frontends beziehungsweise Backends korrekt ist. Zu festgelegten Zeitpunkten werden Systemtests durchgeführt um zu überprüfen ob die spezifizierten Anforderungen erfüllt sind. Zusätzlich wird während der Entwicklung ein Usability-Test erstellt um die Benutzerfreundlichkeit zu verbessern.

Die einzelnen Module der Software werden mittels Unit Tests auf Korrektheit geprüft. Unit-Tests laufen automatisiert ab und sind daher nach jeder Änderung an der Software schnell und einfach durchzuführen. Sie testen das Verhalten einzelner Methoden und damit den feinsten Detaillierungsgrad. In der Applikation wurde das Backend sehr gründlich mittels Unit Tests getestet um die korrekte Funktionsweise der Controller und der API sicherzustellen. Dies ist wichtig, damit sich Fehler im System nicht fortpflanzen. Im Frontend machen Unit Tests nicht wirklich Sinn da diese nicht für den Test von User Interfaces gedacht sind. Es gäbe entsprechende Frameworks welche

Klicks emulieren können. Dieser Aufwand steht aber nicht in einem nützlichen Verhältnis zum Ertrag. Das User Interface wurde vor allem mittels Integrationstests kontrolliert.

Das Zusammenspiel der Module wird mit Hilfe von Integrationstests getestet. Backend sowie Frontend wurden während der Entwicklung bei jedem Review und vor jedem Pullrequest mittels Integrationstests getestet. So wurden zum Beispiel HTTP-Requests direkt an die API gesendet um zu prüfen ob die Antwort dem erwarteten Wert entspricht. Die Integrationstests wurden jeweils nicht dokumentiert da keine konkreten Testszenarien definiert wurden. Dies bietet den Vorteil, dass alle Entwickler eigene Szenarien ausdenken und somit die Testabdeckung besser ist als mit vordefinierten Szenarien. Fehlerfälle wurden im Review erfasst, behoben und erneut getestet.

Der Systemtest prüft im Gegensatz zum Integrationstest zusätzlich zu den funktionalen auch nicht-funktionale Anforderungen. Es handelt sich um einen Black-Box Test. Somit ist kein Wissen über das System nötig. Systemtests prüfen konkrete Anwendungsfälle der Applikation und decken Mängel in der Benutzerfreundlichkeit sowie in der Validation der Benutzereingabe auf. Die Szenarien sind im Kapitel 4 zu finden. Die Protokolle der Systemtests sind im Anhang abgelegt.

Ein Usability-Test wird eingesetzt, um die Benutzerfreundlichkeit des Systems zu testen. Im Gegensatz zum Systemtest wird der Test mit Benutzern erstellt welche das System nicht kennen. Dadurch kann herausgefunden werden wie diese auf das System ansprechen. Die Testfälle und deren Auswertung sind im Kapitel 4 einzusehen. Die Protokolle der Tests sind im Anhang zu finden.

3.7 Deployment Konzept

Für das Betreiben der Applikation gibt es mehrere Möglichkeiten welche evaluiert wurden. Die Applikation ist darauf ausgelegt in einer Microsoft Umgebung zu laufen. Dank der Plattformunabhängigkeit von ASP.NET Core und Entity Framework Core könnte aber auch UNIX verwendet werden. Es muss weiterhin ein Zugriff von aussen verfügbar sein damit die Servicetechniker auf die Dienste zugreifen können. Ausserdem sollte ein Webserver auf dem System laufen um die Applikation zu hosten. Auch sollten mindestens 20GB Speicherplatz vorhanden sein um die zu speichernden Daten abzulegen. Folgende Optionen wurden evaluiert.

- Hosting auf Microsoft Azure
- Hosting auf eigenem Server

Dadurch, dass der Auftraggeber keine Lizenz für Microsoft Azure besitzt und auch keine erwerben möchte blieb noch das Hosting auf einem eigenen Server. Azure oder allgemein eine Cloudlösung bietet eine hohe Skalierbarkeit. Dies ist jedoch für einen Prototyp nicht sehr relevant. Ein Hosting auf eigenen Servern bietet einige Vorteile gegenüber dem Cloudhosting. Da wären die geringeren Kosten und damit auch die Performance pro investiertem Franken. Auch die Datensicherheit ist deutlich besser da die Kontrolle über die Daten jederzeit beim Betreiber des Servers liegt.

Daher ist die Einschränkung kein einschneidender Punkt für das Projekt. Die Applikation wurde so gestaltet, dass jederzeit ohne zusätzlichen Entwicklungsaufwand das Projekt auf Azure bereitgestellt werden kann.

Bei der Implementierung auf einem eigenen Server sind jedoch noch einige Dinge zu beachten. Einerseits sind bei dem Server der HSR nur die Ports 80 und 40000-40010 offen was bei der Konfiguration beachtet werden muss. Auch müssen die Firewall Regeln der HSR beachtet werden [10].

3.8 Erweiterungen

Hier wird beschrieben, wie die Software um neue Funktionalität erweitert werden kann. Ein Beispiel könnte die Erfassung von Servicetechnikern darstellen. Dabei muss zuerst im Backend ein API Endpoint erstellt werden. Dazu muss ein Model für die Datenbank erstellt werden. Darin werden die Felder definiert und die Verknüpfung zum Service Case hergestellt. Anschliessend sollte ein Data Transfer Object für den Servicetechniker erstellt werden. Dieses wird jeweils von der API gesendet und empfangen. Ein DTO Konverter kann danach das Model in ein DTO und umgekehrt umwandeln und validieren.

Als nächstes muss ein Controller erstellt werden welcher das entsprechende Interface implementiert und die Funktionen für Erstellen, Updaten und Löschen bereitstellt. Nach der Definition der Unit Tests für den Controller und das erfolgreiche Ausführen jener ist die Arbeit im Backend erledigt.

Im Frontend muss eine Komponente ähnlich derer für die Maschinentypen erstellt werden. Dies umfasst das HTML, CSS und jeweils eine Komponente für die Übersicht über die Servicetechniker sowie eine Detailansicht für das Anzeigen und Bearbeiten der Daten. Anschliessend wird für die Service Case Erstellung ein neues Feld analog zu den bestehenden angelegt um den Servicetechniker auszuwählen. Dazu sollte das Model des Service Cases angepasst werden um die Verknüpfung abzubilden und korrekt an die API zu senden.

Eine Anpassung bestehender Felder oder das Hinzufügen von Neuen funktioniert ähnlich. Es müssen die Models in Frontend und Backend angepasst werden, das DTO erweitert und Controller sowie UI Komponenten geändert werden.

4 Umsetzung und Testing

4.1 Systemübersicht

Es wird der von Windows Server bereits mitgelieferte IIS als Webserver verwendet da damit das Frontend und Backend an den Client ausgeliefert werden können. ASP.NET Core 2.0 beinhaltet bereits intern einen Kestrel Webserver. Es wird aber immer noch die Verwendung von IIS als Reverse Proxy empfohlen weil damit die Berechtigungen verwaltet werden können sowie falls gewünscht ein SSL-Zertifikat eingerichtet werden kann [11]. Da das Frontend und das Backend auf der gleichen Domain betrieben werden, müssen verschiedene Ports für die Unterscheidung der Requests verwendet werden. Für das Frontend wird der Standard HTTP-Port 80 gewählt und für das Backend der Port 40007. In einem produktiven Umfeld empfiehlt es sich, das Backend auf einer eigenen Domain zu betreiben damit es keine Probleme mit geschlossenen Ports gibt. Dies kann im IIS konfiguriert werden.

Der SQL Server wird auf dem gleichen Server wie der Webserver betrieben. Dieser kann im produktiven Einsatz auf einen weiteren Server ausgelagert werden damit die Performance bei vielen Request erhöht werden kann.

Ausserdem kann das Backend mit ASP.NET Core und Microsoft SQL Server Express auch auf einer Linuxdistribution über einen Nginx Webserver ausgerollt werden [12] [13]. Ein Grobkonzept dazu ist im Kapitel 5 zu finden.

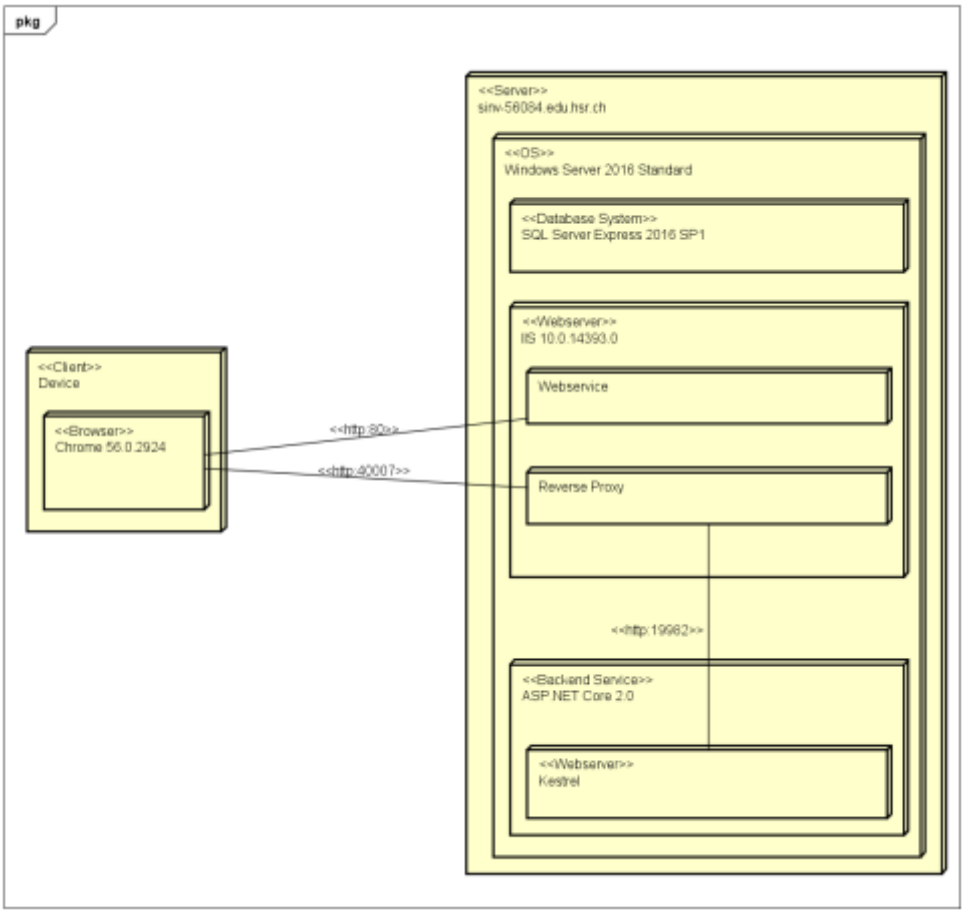

Abbildung 8 – Deployment-Diagramm

4.2 XML Schnittstelle

Das Auslesen der benötigten Daten aus der zur Verfügung gestellten XML-Stückliste ist eine grosse Herausforderung hinsichtlich Wartbarkeit und Unterstützung verschiedener Formate. Für die Arbeit wurde die Stückliste des Lego-Roboters der HSR bereitgestellt. Alle Operationen auf der XML-Struktur wurden in einer Worker-Klasse implementiert. Diese wurde gegen ein Interface programmiert. Sollen nun weitere XML-Formate unterstützt werden, können diese gegen dasselbe Interface implementiert werden und so ohne grosse Änderungen in den Controllern verwendet werden. Dies stellt die Kompatibilität mit verschiedenen, vielleicht erst in der Zukunft verfügbaren, XML-Formaten sicher.

Dabei wurde das Strategy Pattern eingesetzt. Es erlaubt das Auswählen einer Methode zur Datenverarbeitung zur Laufzeit.

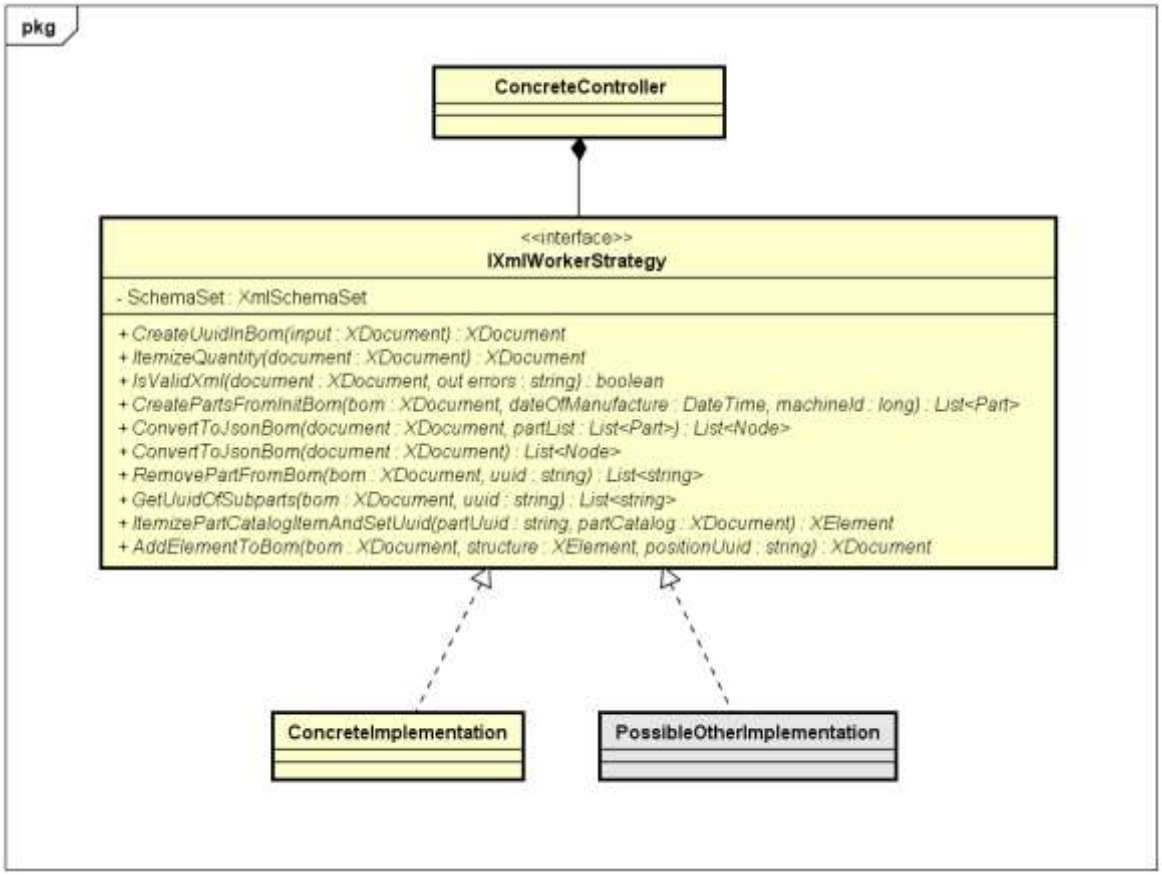

Abbildung 9 - Strategy Pattern des XML-Workers

4.3 Usability-Test

Ein Usability-Test wird mit einer unabhängigen, projektfremden Person durchgeführt um die Benutzerfreundlichkeit und Benutzbarkeit einer Software mit potenziellen Benutzern zu prüfen. Dabei werden verschiedene Szenarien durchgespielt. Die Versuchsperson wird zu lautem Denken aufgefordert. So erfährt der Beobachter, wonach die Person gerade sucht oder was sie erwartet.

Ausserdem wurde bei den regelmässigen Besprechungen mit dem Betreuer jeweils die aktuelle Version vorgeführt und Feedback wenn möglich direkt eingebaut.

4.3.1 Aufgaben

Die definierten Szenarien sollen so gut wie möglich die Nutzung der App im produktiven Umfeld simulieren. Die Datenbank enthält die benötigten Testdaten um die gestellten Aufgaben erfolgreich abzuschliessen. Zusätzlich benötigte Dateien werden vom Betreuer zur Verfügung gestellt.

Test Case 1: Task Rapport erstellen
Als Servicetechniker hast du beim Kunden „Photojam“ an der Maschine mit der Seriennummer „XC3180065“ vom Typ „Endeavor“ eine Software „MS2.0“ installiert. Erfasse diese Änderung im entsprechenden Service Case als Task Rapport.

Test Case 2: Maschinenteil tauschen
An der Maschine mit der Seriennummer „NK3592787“ des Typs „Express 2500“ vom Kunden „Omba“ muss aufgrund des Alters der Maschine das Teil „Lower Part For Turntable Z60“ ausgetauscht werden. Erfasse einen Service Case mit einem Problembeschreib und einen Task Rapport mit der entsprechenden Modifikation. Prüfe anschliessend in der Maschinenansicht ob das Alter des entsprechenden Maschinenteils zurückgesetzt wurde.

Test Case 3: Maschinenteil hinzufügen
An der Maschine mit der Seriennummer „NK3592787“ des Typs „Express 2500“ vom Kunden „Omba“ wird ein Upgrade durchgeführt. In die Baugruppe „CAD-LA000031“ soll ein „Technic 5M Beam“ eingebaut werden. Erfasse einen Service Case mit dem Problembeschreib „Upgrade“ und einen Task Rapport mit der entsprechenden Modifikation. Prüfe anschliessend in der Maschinenansicht ob das Teil korrekt angezeigt wird.

Test Case 4: Neuer Maschinentyp
Du bist nun ein Systemverantwortlicher. Deine Firma hat einen neuen Maschinentyp mit dem Modell „M42“ entwickelt. Dieser soll nun im System erfasst werden. Dafür wirst du zusätzliche Dateien benötigen.

Test Case 5: Neue Maschine
Im letzten Test Case hast du einen neuen Maschinentyp erfasst. Nun werden zwei Maschinen dieses Typs ausgeliefert. Erstelle diese im System. Die Maschinen sollen an den Kunden „Skippad“ geliefert werden und haben die Seriennummern „UO7148000“ bzw. „UO7148001“, wurden heute produziert und werden mit der Softwareversion V1 ausgeliefert.

4.3.2 Auswertung

Der Usability-Test wurde am 04.12.2017 durchgeführt. In der folgenden Tabelle sind die erkannten Schwächen aufgelistet und beschrieben welche Massnahmen zur Behebung durchgeführt wurden. Das vollständige Protokoll ist im Anhang zu finden.

Test Case	Beobachtungen	Massnahmen
TC 1	Button zur Eröffnung eines Task Rapports nicht sofort auffindbar.	Hervorheben des Buttons durch neues Farbkonzept.
TC 1	„Cause of error“ ist ein Pflichtfeld auch wenn der Task Rapport nicht aufgrund eines Fehlers erfasst wurde.	Re-evaluation der Entitäten und des Datenmodells im Zuge der folgenden Bachelorarbeit.
TC 1	UI suggeriert, dass die Erstellung eines Task Rapports abgeschlossen ist, wenn noch keine Modifikation erfasst wurde.	Anzeigen der Ansicht für die Erfassung einer Modifikation nach der Eingabe der Metadaten des Task Rapports.

TC 2	Erwartetes Verhalten nach dem Erstellen eines Service Cases ist, auf dessen Detailansicht zu gelangen. Anstelle bleibt man auf der Ansicht zur Erstellung eines neuen Service Cases.	Weiterleitung auf die Detailansicht nach dem Drücken des „Save“-Buttons.
TC 2	„Back“ Button wurde nicht wie geplant verwendet.	Verschiebung des „Back“ Buttons in die Titelleiste.
TC 3	Es wird erwartet, dass eine erfasste Modifikation wieder gelöscht werden kann.	Anforderung wird nicht unterstützt. Workaround über die Erfassung einer inversen Modifikation.
TC 4	Begriffe „Initial BOM“ und „Part Catalog“ sind nicht klar.	Bedeutung im Frontend mithilfe von „Form-Hints“ ergänzt.
TC 5	Funktion zum Erstellen einer Maschine könnte im Maschinentyp sowie im Kunden existieren.	Funktion wird ausschliesslich im Kunden angeboten um das System weniger komplex zu halten.

Tabelle 5 - Auswertung Usability-Test

Die Testperson lobte weiterhin die durchgehend konsistente Navigation durch die Applikation sowie die Bekanntheit durch das Material-Design. Die Felder für die Metainformationen waren intuitiv auszufüllen. Sie fand sich in der Software schnell zurecht.

4.4 Systemtest

Um das System Design zu testen wurden die in der nachfolgenden Tabelle aufgeführten Testszenarien entworfen. Diese dienen dazu, die verschiedenen Features der Software zu testen und auch Fehlerfälle zu berücksichtigen. Ein Systemtest wird am Ende der Entwicklungsphase durchgeführt um eventuelle Bugs aufzudecken. Diese Tests werden protokolliert und ausgewertet. Die Protokolle sind im Anhang zu finden.

Ein Systemtest wird auch vor der Abgabe durchgeführt um sicherzustellen, dass die beschriebenen Szenarien funktionieren.

Bereich	Testcases	Erwartetes Resultat
Machine Type	TC001: Versuch von Save bei nicht vollständigen Daten	Nicht möglich
Machine Type	TC002: Hochladen von invalidem XML in Part Catalog	Fehler, Part Catalog nicht gespeichert
Machine Type	TC003: Hochladen von invalidem XML in Init BOM	Fehler, BOM nicht gespeichert
Machine Type	TC004: URL Detailview mit falscher ID aufrufen	Fehlermeldung
Machine Type	TC005: Erstellung und Änderung eines validen Objekts	Erfolgsmeldung, Daten persistiert, BOM mit Quanitity 1, Part Catalog mit UUID versehen
Machine Type	TC006: Zeige Übersicht über Maschinentypen	Typen aufgelistet, sauber dargestellt
Customer	TC101: Versuch von Save bei nicht vollständigen Daten	Nicht möglich
Customer	TC102: Versuch von Save bei ZIP Code 123	Nicht möglich
Customer	TC103: URL Detailview mit falscher ID aufrufen	Fehlermeldung
Customer	TC104: Erstellung und Änderung eines validen Objekts	Erfolgsmeldung, Daten persistiert
Customer	TC105: Zeige Übersicht über Kunden	Kunden aufgelistet, sauber dargestellt

Machine	TC201: Versuch von Save bei nicht vollständigen Daten	Nicht möglich
Machine	TC202: URL Detailview mit falscher ID aufrufen	Fehlermeldung
Machine	TC203: Erstellung und Änderung eines validen Objekts	Erfolgsmeldung, Daten persistiert, BOM kopiert, BOM mit UUID versehen, Parts angelegt
Machine	TC204: Zeige Übersicht über Maschinen	Typen aufgelistet, sauber dargestellt
Service Case	TC301: Versuch von Save bei nicht vollständigen Daten	Nicht möglich
Service Case	TC302: URL Detailview mit falscher ID aufrufen	Fehlermeldung
Service Case	TC303: Erstellung und Änderung eines validen Objekts	Erfolgsmeldung, Daten persistiert
Service Case	TC304: Zeige Übersicht über Service Cases	Typen aufgelistet, sauber dargestellt
Task Rapport	TC401: Versuch von Save bei nicht vollständigen Daten	Nicht möglich
Task Rapport	TC402: URL Detailview mit falscher ID aufrufen	Fehlermeldung
Task Rapport	TC403: Erstellung und Änderung eines validen Objekts	Erfolgsmeldung, Daten persistiert
Task Rapport	TC304: Zeige Übersicht über Task Rapports	Typen aufgelistet, sauber dargestellt
Task Rapport	TC305: Erfassen einer Modifikation in abgeschlossenem Task Rapport	Nicht möglich
Task Rapport	TC306: Ändern eines Task Rapports während Modifikation erfasst wird	Nicht möglich
Task Rapport	TC307: Klicken auf weiter während nichts ausgewählt ist	Fehler bei Validierung
Task Rapport	TC308: Abbrechen einer Modifikation im letzten Schritt	Form zurückgesetzt, Daten nicht gespeichert
Task Rapport	TC309: Durchführen einer Add-Modifikation	Erfolgsmeldung, Daten in Maschine und BOM nachgetragen, Datum der neuen Komponente gesetzt
Task Rapport	TC310: Durchführen einer Remove-Modifikation	Erfolgsmeldung, Daten in Maschine und BOM nachgetragen
Task Rapport	TC311: Durchführen einer Replace-Modifikation	Erfolgsmeldung, Daten in Maschine und BOM nachgetragen, Datum auf heute gesetzt
Task Rapport	TC312: Durchführen einer Software Version-Modifikation	Erfolgsmeldung, Daten in Maschine nachgetragen
Task Rapport	TC313: Durchführen einer Location-Modifikation	Erfolgsmeldung, Daten in Maschine nachgetragen

Tabelle 6 - Systemtestszenarien

4.5 Code Metriken

Code Metriken erlauben es Entwicklern, problematische Stellen im Code zu finden um diese zu überarbeiten. Auch geben sie einen Überblick über die Codebasis. Als Tool für die Erstellung der Metriken wurde für das Backend NDepend evaluiert. Dabei ist jedoch nur eine Testversion gratis verfügbar. Andere Tools zur Erstellung der gewünschten Metriken unterstützen kein ASP.NET Core.

Metrik	Grösse
Physische Codezeilen	19‘856 LOC
Effektive Codezeilen	17‘517 LOC
Logische Codezeilen	1‘912 LOC
Maximale zyklomatische Komplexität	14 (XML Worker)
Durchschnittliche zyklomatische Komplexität	1.65
Anzahl Unit Tests	118 Unit Tests
Testabdeckung	83%

Tabelle 7 - Code Metriken Backend

Die physischen Codezeilen geben an, wie viele Zeichen physikalisch im Quellcode vorhanden sind. Dies kann einfach von der Anzahl Zeilenumbrüche im Projekt abgeleitet werden. Die effektiven Codezeilen entsprechen den Physischen minus aller Kommentarzeilen, Leerzeilen und Zeilen mit nur einer öffnenden oder schliessenden Klammer. Die logischen Codezeilen werden anders ausgewertet. Sie werden anhand der PDB Dateien von Visual Studio errechnet. Dabei werden alle Sequenzpunkte innerhalb dieser Datei zusammengezählt. Ein Sequenzpunkt wird benötigt um ein Punkt innerhalb der Intermediate Language zu referenzieren. Der Vorteil ist, dass der Programmierstil keinen Einfluss auf die Metrik hat und auch sprachunabhängig ist. Bei den logischen Codezeilen werden daher auch keine Interfaces, abstrakte Methoden, Enums, Namespaces, Typen, Eigenschaften oder Methodendeklarationen berücksichtigt [14].

Bei der zyklomatischen Komplexität besitzt die Methode *„ConvertToJsonBomRecursive“ im XML Worker* deutlich am meisten Verzweigungen. Diese konnten in einem Refactoring noch etwas verkleinert und aufgeteilt werden. Damit ist sie gerade noch so im wartbaren empfohlenen Bereich von 1-15. Der Durchschnitt von 1.65 hingegen ist sehr gut und deutet auf eine gut wartbare Codebasis hin.

Die Unit Tests wurden vor allem für die Controller geschrieben. Daher haben wir bei diesen auch fast eine 100% Abdeckung. Daraus ergaben sich für die effektiv genutzten Sourcefiles ohne den Datenbank-Initialisier mit den Testdaten und den automatisch generierten Migrationen die 83% Testabdeckung. Dies ist ein guter Wert, verspricht aber keineswegs fehlerfreie Software für die abgedeckten Teile.

Im Angular Frontend stehen keine Tools zur Verfügung um die zyklomatische Komplexität zu ermitteln. Aus diesem Grund wurde auf das Plugin Statistic von WebStorm zurückgegriffen, da dieses ermöglicht die Anzahl Codezeilen und die Dateigrössen nach Dateitypen zu ermitteln [15].

Dateitypen	Codezeilen	Anzahl Dateien
Typescript	2’039	61
HTML	747	29
CSS	294	9

Tabelle 8 - Codestatistik Angular Frontend

Die ermittelten Codezeilen entsprechen den effektiven Codezeilen. Bei dieser Zahl wurden schon alle Leerzeilen sowie Kommentarzeilen abgezogen. Gut zu sehen ist, dass durch Angular ein grosser Overhead für die Verwaltung der Komponenten und Module entstanden ist. Dies zeigt sich durch die grosse Anzahl an Typescript Dateien.

Der optimierte Build für das produktive Umfeld unseres Frontend umfasst eine Grösse von knapp zwei MB, die beim ersten Request des Clients übertragen werden. Bei den nächsten Requests werden nur noch die angeforderten Daten, die im kleinen kB Bereich liegen, vom Backend übertragen.

4.6 Showcases

Die folgenden Showcases sollen die Möglichkeiten des Prototyps anhand spezifischer Anwendungsfälle aufzeigen. Es wird nicht der gesamte Funktionsumfang demonstriert sondern lediglich die wichtigsten Funktionen.

4.6.1 Showcase „Defekte Pumpe?“

Der Servicetechniker bekommt einen Service Case der Firma „Robodog AG“ zugeteilt. Es geht um die Maschine mit der Seriennummer „Robodog1“. Er befindet sich derzeit im Lager seiner Firma und bereitet sich auf den Einsatz vor. Nun fragt er sich natürlich welche Ersatzteile er mitnehmen muss, sodass er nur einmal zu der Firma des Auftraggebers fahren muss.

Er wählt den entsprechenden Kunden in der Webapplikation auf seinem Smartphone und lässt sich dessen Service Cases darstellen. Darin findet er den ihm zugeteilten Service Case und liest zuerst die Problembeschreibung – „Druckverlust beim Pumpen von Kühlflüssigkeit“. Daraus schliesst er, dass die Ursache bei einer defekten Pumpe liegen könnte.

Nun hat er aber noch eine andere Möglichkeit, Informationen einzusehen. Er öffnet die zum Service Case gehörende Maschine und sieht sich die Standzeiten der mit der Pumpe interagierenden Teile an. Dabei sieht er, dass bei der Maschine welche vor acht Jahren ausgeliefert wurde die Pumpe vor einem Jahr bereits ausgetauscht wurde. Aus internen Unterlagen weiss er, dass die Betriebszeit einer Pumpe bei normalem Nutzungsgrad etwa 5 Jahre beträgt. Während einer Generalüberholung der Maschine vor einem Jahr wurden die meisten Verschleissteile erneuert. Jedoch fällt dem Servicetechniker auf, dass der Schlauch welcher für den Transport der Kühlflüssigkeit zuständig ist seit Auslieferung derselbe ist. Hier könnte sich also aus einem Grund ein kleines Leck gebildet haben was den Druckverlust zu erklären vermag.

Natürlich kann der Defekt auch an der Pumpe oder einem anderen Teil liegen. Also packt der Servicetechniker die Pumpe, einen Ersatzschlauch und seine übliche Ausrüstung ein und macht sich auf den Weg.

4.6.2 Showcase „Ein zusätzliches Problem“

Aufbauend auf dem vorhergehenden Showcase ist der Servicetechniker beim Kunden und arbeitet an der Maschine. Seine Vermutung war korrekt und er konnte den defekten Schlauch austauschen. Nun trägt er die Fehlerursache, seine Arbeit und das ausgewechselte Teil in einen neuen Task Rapport ein.

Kurz darauf erscheint der Abteilungsleiter und meldet, dass eine andere Maschine nicht mehr ganz rund läuft. Ob er sich dies doch ansehen könnte. Er erfasst zuerst einen Service Case und trägt da das Problem ein. Aus seiner Erfahrung schliesst der Servicetechniker, dass das Problem wohl an zu wenig Öl liegt. Nachdem er dieses nachgefüllt hat testet er die Maschine welche nun wieder wie erwartet funktioniert. Der Servicetechniker trägt nun wiederum die Fehlerursache und seine Arbeit in einen neuen Task Rapport ein und schliesst diesen ab.

Die gemachten Arbeiten können nun auf den Service Cases eingesehen und nachvollzogen werden. Der ausgetauschte Schlauch taucht nun als frisch verbaut in der Stückliste auf.

Zur Absicherung zeigt der Servicetechniker dem Auftraggeber den Service Rapport.

4.6.3 Showcase „Aus einem Hardware Upgrade wird ein Software Update“

Der Servicetechniker bekommt zwei Service Cases der Firma „Legoroboter AG“ zugeteilt. Darin geht es um das Upgrade eines Greifarms von zwei Maschinen. Er bereitet sich auf den Einsatz vor.

Zuerst schaut er sich die Maschine A aus dem ersten Service Case an. Diese ist momentan auf Softwareversion 2.0. Er weiss, dass diese Softwareversion den neuen Greifarm unterstützt. Also muss kein zusätzliches Update gemacht werden.

Bei der Maschine B aus dem zweiten Service Case ist es nicht ganz so einfach. Diese ist noch auf einer alten Version 1.7. Bei einem Upgrade des Greifarms ist die Version 2.0 erforderlich. Daher muss er die Software auf einem USB-Stick mitnehmen. Ausserdem packt er zwei Greifarme für die Hardware Upgrades ein.

Beim Kunden angekommen erledigt er die geplanten Schritte und hat dank der sauberen Vorbereitung bereits alle benötigten Teile zur Hand. Er erfasst das Upgrade jeweils in einem neuen Task Rapport in den entsprechenden Service Cases. Er wählt dazu im Task Rapport die Option ein neues Teil hinzuzufügen. Nachdem er den Einfüge-Ort in der Stückliste und das entsprechende Upgrade-Teil aus dem Ersatzteilkatalog gewählt hat, kann er die Modifikation bestätigen. Nun muss er noch das Softwareupdate im Task Rapport des zweiten Service Cases hinzufügen.

Die gemachten Arbeiten können nun auf den Service Cases eingesehen und nachvollzogen werden. Das hinzugefügte Upgrade-Paket taucht nun als frisch verbaut in der Stückliste auf. Zudem wird die Softwareversion bei der zweiten Maschine korrekt dargestellt.

4.6.4 Showcase „Neue Maschine“

Ein neuer Maschinentyp, der Robodog+, wird entwickelt. Die ersten beiden Maschinen dieses Typs werden heute ausgeliefert. Dieser Maschinentyp ist noch nicht im System erfasst. Um spätere Serviceeinsätze zu vereinfachen soll dieser nun erfasst werden.

Die systemverantwortliche Person erstellt am Desktop den neuen Maschinentyp im System, spezifiziert das Modell und importiert die aktuelle Stückliste mit welcher die Maschinen im Moment das Werk verlassen. Zusätzlich importiert sie auch den Ersatzteilkatalog.

Nun müssen die beiden Maschinen noch erfasst werden. Die systemverantwortliche Person trägt alle verfügbaren Metainformationen wie Seriennummer, Herstellungsdatum und Softwareversion der beiden Maschinen ein.

Das System erstellt aus der im Maschinentyp spezifizierten Stückliste die Maschinenspezifische Stückliste und speichert sie ab. Ab jetzt können die Maschinen unabhängig voneinander gewartet werden.

5 Ergebnis und Ausblick

5.1 Zielerreichung

Die Zielerreichung kann an den funktionalen und nicht-funktionalen Anforderungen gemessen werden. Bei den funktionalen Anforderungen wurden alle als Bronze und Silber eingestuften Use Cases implementiert. Darüber hinaus wurden vier der elf Anforderungen der Gold-Ausbaustufe implementiert. Diese waren das Auswechseln der initialen Stückliste und des Ersatzteilkatalogs während des Betriebs, das Reporting über das Alter eines gewissen Maschinenteils und die kurz vor Ende der Entwicklungsphase hinzugekommenen Anforderungen für das Aktualisieren des Standorts und der Softwareversion über eine Modifikation.

Use Case	Ausbaustufe	Implementiert
100 – Create machine type	Bronze	Ja
100a – Import initial bill of materials from file	Bronze	Ja
100b – Import part catalogue from file	Bronze	Ja
110 – Show machine type	Bronze	Ja
110a – Show initial bill of materials	Silber	Ja
110b – Show part catalogue	Silber	Ja
120 – Update machine type	Gold	Ja
200 – Create machine	Bronze	Ja
210 – Show machine properties	Bronze	Ja
210a – Show service cases of machine	Bronze	Ja
210b – Show history of parts	Gold	Nein
210c – Show bill of materials	Bronze	Ja
220 – Update machine properties	Silber	Ja
300 – Create customer	Bronze	Ja
310 – Show customer	Bronze	Ja
310a – Show machines	Bronze	Ja
310b – Show service case by customer	Bronze	Ja
320 – Update customer properties	Silber	Ja
400 – Create service case	Bronze	Ja
410 – Show service case with all task rapports	Bronze	Ja
420 – Update service case properties	Silber	Ja
430 – Change service case status	Bronze	Ja
500 – Create task rapport from service case	Bronze	Ja
500a – Add part	Silber	Ja
500b – Remove part	Silber	Ja
500c – Exchange part	Bronze	Ja
500d – Update software version	Gold	Ja
500e – Update location	Gold	Ja
510 – Read task rapport from service case view	Bronze	Ja
600 – Create contact person of customer	Gold	Nein
610 – Update contact person of customer	Gold	Nein
700 – Display service cases in reporting	Gold	Nein
700a – Filter service cases	Gold	Nein
710 – Get all parts of a specific type with their age	Gold	Ja
720 – Display service rapport	Bronze	Ja
800 – Create service technician	Gold	Nein
810 – Update service technician	Gold	Nein

Tabelle 9 - Erfüllung der funktionalen Anforderungen

Es wurden auch alle definierten User Stories berücksichtigt und implementiert. Diese sind auch in den definierten Showcases abgebildet. Die Showcases können vollständig präsentiert werden.

Die nicht-funktionalen Anforderungen für den Prototyp wurden erreicht. Die Erweiterbarkeit ist nicht objektiv messbar. Jedoch wurden während der Entwicklungsphase laufend neue Module hinzugefügt. Daher wurde dieser Punkt mit „erfüllt“ bewertet. Bei der Skalierbarkeit und dem Austausch der Datenquelle ist anzumerken, dass diese nicht durch ein anderes DBMS ersetzt werden kann. Jedoch ist das Auswechseln der Datenbank kein Problem.

Nicht-funktionale Anforderung	Erfüllt	Bemerkung
Kompatibilität	Ja	
Erweiterbarkeit	Ja	Nicht direkt messbar, subjektiv erfüllt
Skalierbarkeit	Ja	Mit der Einschränkung, dass Entity Framework als OR-Mapper verwendet werden kann.
Änderungssensivität	Ja	
Datenintegrität	Ja	
Bedienbarkeit	Ja	Gemessen mit dem durchgeführten Usability-Test

Tabelle 10 - Erfüllung der nicht-funktionalen Anforderungen

5.2 Ausblick und Erweiterungen

In den folgenden Abschnitten ist jeweils erläutert, wie die Applikation erweitert, verändert und verbessert werden kann. Es werden grobe Konzepte vorgestellt. Details sind dabei genauer zu evaluieren. Auch die Machbarkeit der Erweiterungen wurde nicht getestet sondern nur theoretisch erarbeitet.

5.2.1 Open Source

Die Verwendung von Open Source bietet, neben den nicht vorhandenen Kosten, Chancen im Bereich Erweiterbarkeit und Innovation sowie Sicherheit. Daher folgt hier ein Konzept zur Migration der Applikation auf Open Source. Einige Aspekte wurden im Bericht bereits an den geeigneten Stellen erwähnt. Diese werden hier nochmals aufgeführt und in einen Kontext gebracht.

In einer ersten Phase kann der Umstieg auf ein Unix-Betriebssystem realisiert werden. Auf Unix ist es nicht möglich ein IIS-Server zu starten. Daher sollte für den Reverse Proxy ein anderer Webserver benutzt werden. Die Literatur schlägt Nginx zur Nutzung vor [13]. Angular selbst kann auch auf demselben Webserver aufgespielt werden [16]. Somit sähe die Struktur wie folgt aus.

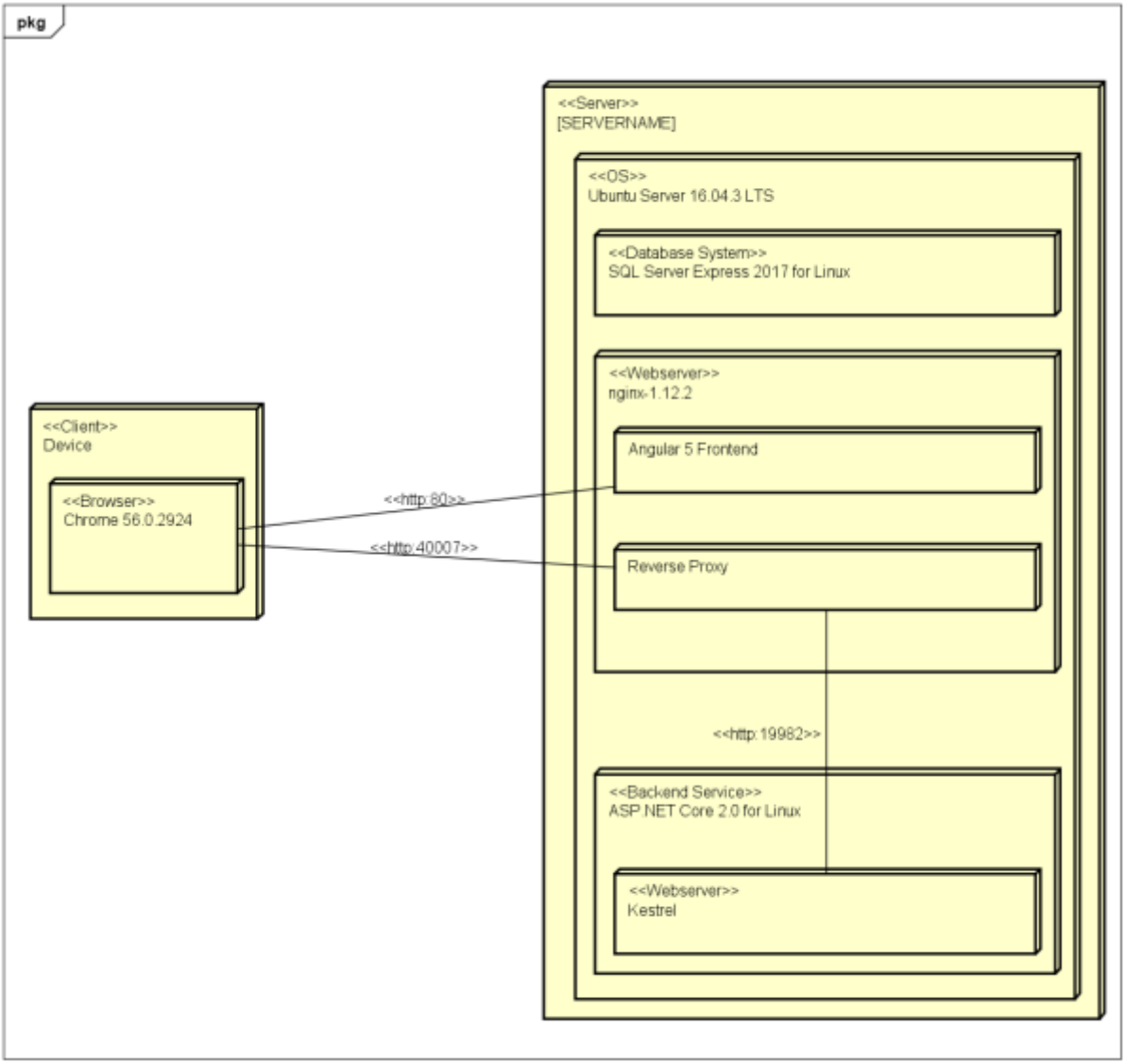

Abbildung 10 – Deploymentvorschlag Open Source

Es wird ersichtlich, dass sich an der generellen Struktur nichts ändert. Nginx ersetzt den IIS und es wurden anstelle der Windows Varianten von ASP.NET und dem Betriebssystem die frei verfügbaren Linux-Alternativen aufgezeichnet. Bei der Wahl des Betriebssystems ist man sehr frei. Ubuntu Server bietet einen guten Support und gehört zu den beliebtesten Serverdistributionen von Linux.

In einer zweiten Phase sollte der Microsoft SQL Server ausgetauscht werden. Dieser ist zwar gratis, wird aber nicht unter einer Open Source Lizenz vertrieben. Dafür muss ein neues Datenbankframework evaluiert werden. Je nach weiteren Anforderungen kann dies CouchDB, MySQL oder PostgreSQL sein. Ausserdem muss das Entity Framework als OR-Mapper ausgewechselt werden. Je nach DBMS gibt es verschiedene Lösungen welche sich aber in der Verwendung sehr ähneln. Dazu bietet sich die Einführung eines Service Layer in der Architektur an welcher dafür sorgt, dass das DBMS jederzeit getauscht werden könnte.

5.2.2 Offline Funktionalität

Eine grosse Herausforderung wird die Verfügbarkeit der Services wenn der Servicetechniker periodisch Netzwerkunterbrüche erlebt. Dies kann bei der Arbeit in Produktionshallen öfters der Fall sein. Ein vereinfachtes Konzept könnte wie folgt aussehen.

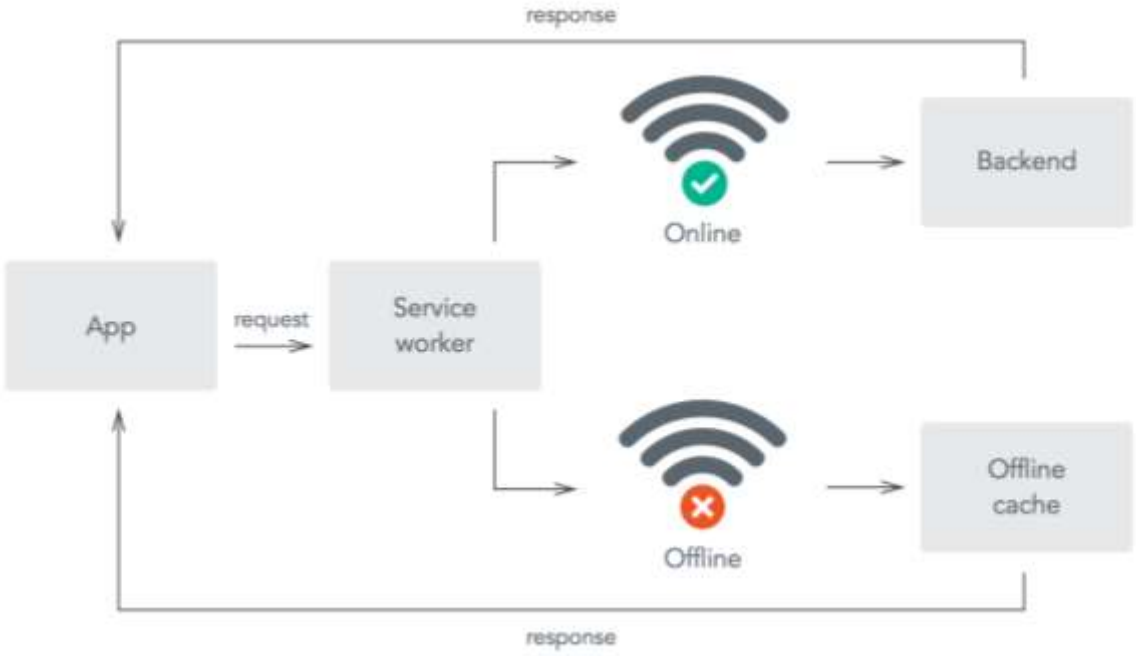

Abbildung 11 - Grobkonzept Offlineverfügbarkeit [17]

Dabei verwendet die Applikation weiterhin einen Service Provider. Dieser entscheidet dann aber je nach Netzwerkzustand, ob die Daten aus einem Cache geladen werden oder über einen Webrequest angezeigt werden. Diese Lösung ist bei nur lesenden Befehlen nicht sehr problematisch. Sollten auch schreibende Befehle dazu kommen, müssen bei einer Datenbank welche von mehreren Personen benutzt wird die Transaktionssicherheit speziell beachtet werden da die bisherige nicht mehr ausreichen wird. Eine Ausarbeitung eines solchen Konzepts für den kompletten Funktionsumfang ist überaus zeitaufwändig und bedarf der Erstellung mehrerer Machbarkeitsanalysen und Prototypen.

Ein Kompromiss wäre das Anbieten von allen Lesezugriffen sowie das Erstellen von neuen Objekten während der Offline-Phase. Damit wären nur die Update-Befehle nicht abgedeckt.

Somit wäre jedoch nur ein Teil zufriedenstellend gelöst. Ein weiterer ist die Implementation der Synchronisation. Denn ab einer gewissen Grösse der Datenbasis kann nicht mehr ohne weiteres die ganze Datenbank auf ein mobiles Gerät geklont werden. Dann muss ein Algorithmus eingesetzt werden welcher bestimmt, welche Daten nun auf das Gerät synchronisiert werden müssen und welche nicht. Da gibt es auch wieder verschiedene Ansätze. Zum Beispiel kann anhand des Nutzungsverhaltens evaluiert werden, welche Daten in nächster Zeit benötigt werden. Dies setzt einen Machine Learning Algorithmus voraus. Auch dieser ist schwierig korrekt zu implementieren. Ein weiterer Ansatz wäre das Laden von Daten welche vom jetzigen Punkt in der Software mit einer gewissen Anzahl Navigationsschritten erreicht werden können.

Eine Synchronisation an sich stellt die Anforderung einer Datenbank welche diese Art von Verfügbarkeit bereitstellt. Microsoft SQL Server kann dies nicht ohne weiteres. Branchenführende Anbieter sind zum Beispiel CouchDB oder PouchDB. Diese hätten das Wegfallen von Entity Framework als OR-Mapper zur Folge.

Im Zuge der Offline-Verfügbarkeit bietet sich die Entwicklung eines Android Apps an. SQLite kann die Anforderungen grösstenteils abdecken und bietet dabei auch noch bessere GPS-Unterstützung sowie eine eingebaute Datenbank. Dabei sollte sicherlich das Android Sync Adapter Framework evaluiert werden [18].

5.2.3 Rollenkonzept

Die Rollen wurden in der bestehenden Applikation nur rudimentär umgesetzt. Es gibt eine Ansicht für den Servicetechniker sowie eine für den Systemverantwortlichen. Ausserdem existieren gemeinsame Ansichten. Um ein Rollenkonzept einzuführen muss eine Anmeldefunktion eingebaut werden und gewisse Ansichten nur für die definierten Rollen sichtbar sein. Ein grobes Rollenkonzept könnte wie folgt aussehen.

Rollenname	Berechtigungen
Systemtechniker	- Personalisiertes Dashboard - Hinzufügen und Bearbeiten von Task Rapports - Hinzufügen von Modifikationen - Hinzufügen und Bearbeiten von Service Cases - Ansehen von Kunden, Maschinentypen, Maschinen, Service Cases, Task Rapports und Servicerapports - Erstellen von Servicerapports - Workflow Tagesplanung
Systemverantwortlicher	- Personalisiertes Dashboard - Hinzufügen und Bearbeiten von Kunden, Maschinentypen und Maschinen - Ansehen von Kunden, Maschinentypen, Maschinen, Service Cases, Task Rapports und Servicerapports - Erstellen von Auswertungen - Definition von Auswertungen
Operator	- Personalisiertes Dashboard - Hinzufügen und Bearbeiten von Service Cases und Kunden - Ansehen von Kunden, Maschinentypen, Maschinen, Service Cases, Task Rapports und Servicerapports - Erstellen von Auswertungen - Erstellen von Tagesplanungen

Tabelle 11 – Rollenkonzept

Dies erlaubt es, verschiedene Workflows zu definieren. Zum Beispiel eine Tagesplanung welche von einem Operator erstellt wird. Diese wird dann von einem Servicetechniker abgearbeitet und in Task Rapports dokumentiert. Durch die Strukturierung wird die Applikation für einen Anwender schlanker und einfacher zu bedienen.

Es wäre dann auch möglich, die Rollen Systemverantwortlicher und Operator nur auf einem Desktop anzubieten und für den Servicetechniker eine native Mobile-App zu entwickeln welche auf dessen Bedürfnisse zugeschnitten ist.

6 Literaturverzeichnis

[1] Microsoft, «Wahl zwischen .NET Core und .NET Framework für Server-Apps,» 13 10 2017. [Online]. Available: https://docs.microsoft.com/de-de/dotnet/standard/choosing-core-framework-server.

[2] Python Software Foundation, «Python. Executive Summary,» 06 Oktober 2017. [Online]. Available: https://www.python.org/doc/essays/blurb/.

[3] Google, «FEATURES & BENEFITS,» Angular, [Online]. Available: https://angular.io/features. [Zugriff am 11 Oktober 2017].

[4] Google, «Angular Material,» Angular, [Online]. Available: https://material.angular.io/. [Zugriff am 11 Oktober 2017].

[5] JAX Editorial Team, «Angular 5 is here,» JAXenter, 1 Dezember 2017. [Online]. Available: https://jaxenter.com/road-to-angular-5-133253.html. [Zugriff am 11 Dezember 2017].

[6] IETF, «RFC 4122 - A Universally Unique IDentifier,» Juli 2005. [Online]. Available: https://tools.ietf.org/pdf/rfc4122.pdf. [Zugriff am 9 November 2017].

[7] E. Kotsakis und K. Böhm, «XML Schema Directory: a data structure for XML data processing,» 6 August 2002. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=882376. [Zugriff am 23 November 2017].

[8] Microsoft, «Designing and Implementing FILESTREAM Storage,» 2010. [Online]. Available: https://technet.microsoft.com/en-us/library/bb933993(v=sql.105).aspx. [Zugriff am 23 November 2017].

[9] I. Dayen, «Storing XML in Relational Databases,» O'Reilly Media, Inc., 20 Juni 2001. [Online]. Available: https://www.xml.com/pub/a/2001/06/20/databases.html. [Zugriff am 28 November 2017].

[10] C. Spielmann, «Virtual Infrastructure Guide,» Rapperswil, 2017.

[11] Microsoft, «Introduction to Kestrel web server implementation in ASP.NET Core,» 8 Februar 2017. [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/servers/kestrel?tabs=aspnetcore2x. [Zugriff am 9 September 2017].

[12] Microsoft, «Microsoft SQL Server 2017 Whitepaper,» 2017. [Online]. Available: http://info.microsoft.com/rs/157-GQE-382/images/EN-US-CNTNT-Whitepaper-DBMC-SQLServerOnLinuxQuickStart_Updated.pdf. [Zugriff am 9 November 2017].

[13] Microsoft, «Set up a hosting environment for ASP.NET Core on Linux with Nginx,» 8 August 2017. [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/publishing/linuxproduction?tabs=aspnetcore2x. [Zugriff am 9 November 2017].

[14] ZEN PROGRAM LTD, «Code Metrics Definitions,» NDepend, [Online]. Available: https://www.ndepend.com/docs/code-metrics#NbLinesOfCode. [Zugriff am 8 Dezember 2017].

[15] T. Topinka, «Statistic,» JET BRAINS, 17 November 2017. [Online]. Available: https://plugins.jetbrains.com/plugin/4509-statistic. [Zugriff am 19 Dezember 2017].

[16] A. Matei, «How to configure Nginx in production to serve an Angular app,» CodingpediaOrg, 21 Mai 2017. [Online]. Available: http://www.codingpedia.org/ama/how-to-configure-nginx-in-production-to-serve-angular-app-and-reverse-proxy-nodejs. [Zugriff am 12 Dezember 2017].

[17] R. Chenkie, «Creating Offline-First Web Apps with Service Workers,» Auth0, 30 Oktober 2015. [Online]. Available: https://auth0.com/blog/creating-offline-first-web-apps-with-service-workers/. [Zugriff am 11 Dezember 2017].

[18] A. Emani, «Handling Offline Capability and Data Sync In An Android* App,» Intel, 9 Dezember 2014. [Online]. Available: https://software.intel.com/en-us/articles/handling-offline-capability-and-data-sync-in-an-android-app-part-1. [Zugriff am 12 Dezember 2017].

[19] C. Larman, Agile and iterative development, 2004.

7 Abbildungsverzeichnis

8 Tabellenverzeichnis

Anhang

Anhang A – Use Case Diagramm

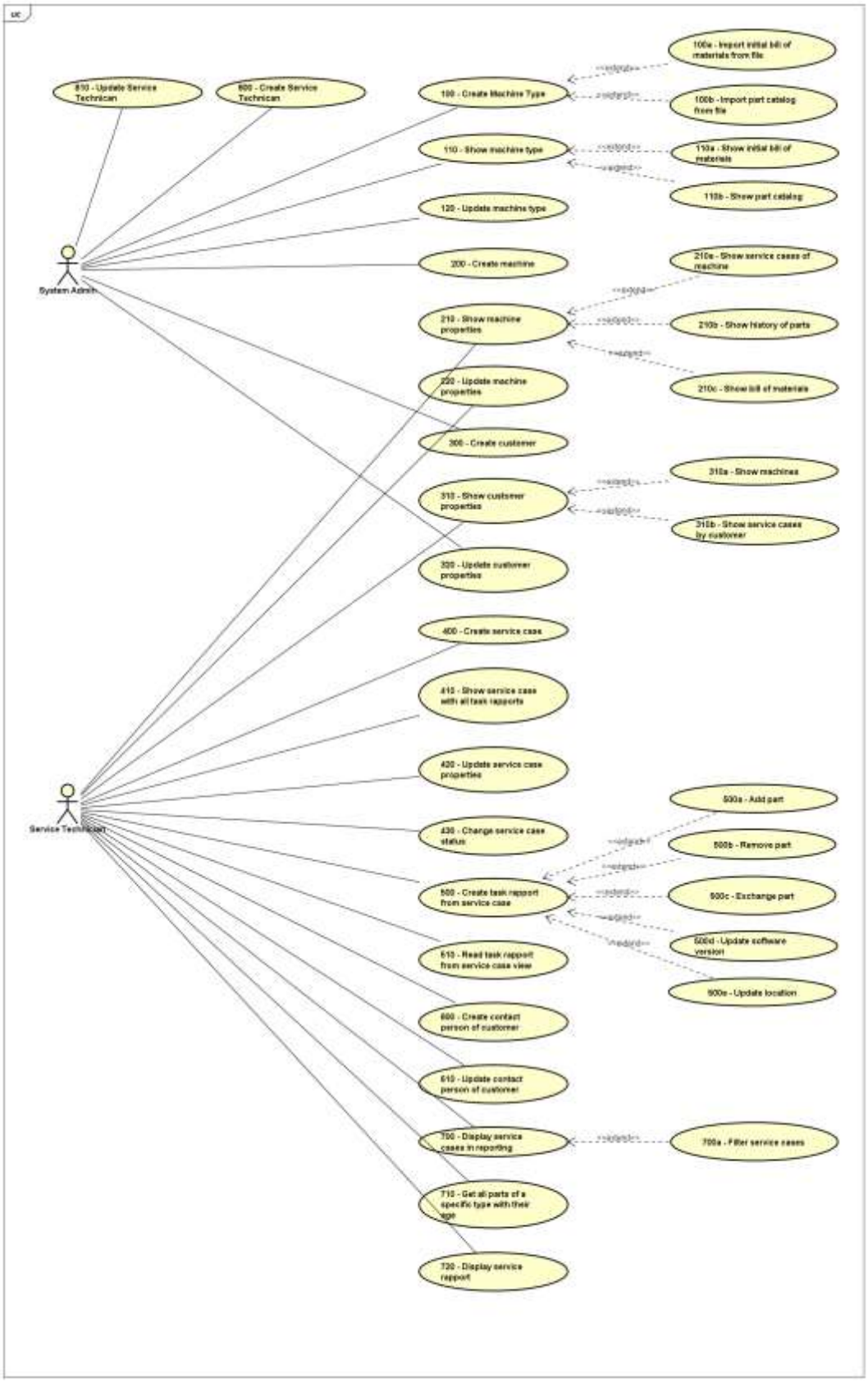

Abbildung 12 - Use Case Diagramm

Anhang B – Test Protokolle

Systemtest 04.12.2017

Zu testende Version: Build #56, Commit c5ca620
Testumgebung: Windows 10, Datenbank initialisiert
Tester: Luca Salzani

Testcases	Erwartetes Resultat	Bestanden
TC001	Nicht möglich	Ja
TC002	Fehler, Änderungen nicht gespeichert	Ja
TC003	Fehler, Änderungen nicht gespeichert	Ja
TC004	Fehlermeldung	Ja
TC005	Erfolgsmeldung, Daten persistiert, BOM mit Quanitity 1, Part Catalog mit UUID versehen	Ja
TC006	Typen aufgelistet, sauber dargestellt	Ja
TC101	Nicht möglich	Ja
TC102	Nicht möglich	Ja
TC103	Fehlermeldung	Ja
TC104	Erfolgsmeldung, Daten persistiert	Ja
TC105	Kunden aufgelistet, sauber dargestellt	Ja
TC201	Nicht möglich	Ja
TC202	Fehlermeldung	Ja
TC203	Erfolgsmeldung, Daten persistiert, BOM kopiert, BOM mit UUID versehen, Parts angelegt	Ja
TC204	Typen aufgelistet, sauber dargestellt	Ja
TC301	Nicht möglich	Ja
TC302	Fehlermeldung	Ja
TC303	Erfolgsmeldung, Daten persistiert	Ja
TC304	Typen aufgelistet, sauber dargestellt	Ja
TC401	Nicht möglich	Ja
TC402	Fehlermeldung	Ja
TC403	Erfolgsmeldung, Daten persistiert	Ja
TC304	Typen aufgelistet, sauber dargestellt	Ja
TC305	Nicht möglich	Ja
TC306	Nicht möglich	Ja
TC307	Fehler Validierung	Ja
TC308	Form zurückgesetzt, Daten nicht gespeichert	Ja
TC309	Erfolgsmeldung, Daten in Maschine und BOM nachgetragen, Datum der neuen Komponente gesetzt	Ja
TC310	Erfolgsmeldung, Daten in Maschine und BOM nachgetragen	Ja
TC311	Erfolgsmeldung, Daten in Maschine und BOM nachgetragen, Datum auf heute gesetzt	Ja
TC312	Erfolgsmeldung, Daten in Maschine nachgetragen	Ja
TC313	Erfolgsmeldung, Daten in Maschine nachgetragen	Ja

Tabelle 12 - Systemtestprotokoll 04.12.2017

Verbesserungen

Testcase	Beschreibung	GitHub Issue
TC00X	Feldvalidation wird bei Klick auf Upload ausgeführt	#31
TC103	Inkorrekte Fehlermeldung bei Eingabe von falscher URL	#32

Tabelle 13 - Verbesserungen aus Systemtest 04.12.2017

Systemtest 14.12.2017

Zu testende Version: Build #64, Commit 0d58cb5
Testumgebung: Windows 10, Datenbank initialisiert
Tester: Luca Salzani

Testcases	Erwartetes Resultat	Bestanden
TC001	Nicht möglich	Ja
TC002	Fehler, Änderungen nicht gespeichert	Ja
TC003	Fehler, Änderungen nicht gespeichert	Ja
TC004	Fehlermeldung	Ja
TC005	Erfolgsmeldung, Daten persistiert, BOM mit Quanitity 1, Part Catalog mit UUID versehen	Ja
TC006	Typen aufgelistet, sauber dargestellt	Ja
TC101	Nicht möglich	Ja
TC102	Nicht möglich	Ja
TC103	Fehlermeldung	Ja
TC104	Erfolgsmeldung, Daten persistiert	Ja
TC105	Kunden aufgelistet, sauber dargestellt	Ja
TC201	Nicht möglich	Ja
TC202	Fehlermeldung	Ja
TC203	Erfolgsmeldung, Daten persistiert, BOM kopiert, BOM mit UUID versehen, Parts angelegt	Ja
TC204	Typen aufgelistet, sauber dargestellt	Ja
TC301	Nicht möglich	Ja
TC302	Fehlermeldung	Ja
TC303	Erfolgsmeldung, Daten persistiert	Ja
TC304	Typen aufgelistet, sauber dargestellt	Ja
TC401	Nicht möglich	Ja
TC402	Fehlermeldung	Ja
TC403	Erfolgsmeldung, Daten persistiert	Ja
TC304	Typen aufgelistet, sauber dargestellt	Ja
TC305	Nicht möglich	Ja
TC306	Nicht möglich	Ja
TC307	Fehler Validierung	Ja
TC308	Form zurückgesetzt, Daten nicht gespeichert	Ja
TC309	Erfolgsmeldung, Daten in Maschine und BOM nachgetragen, Datum der neuen Komponente gesetzt	Nein
TC310	Erfolgsmeldung, Daten in Maschine und BOM nachgetragen	Ja
TC311	Erfolgsmeldung, Daten in Maschine und BOM nachgetragen, Datum auf heute gesetzt	Nein
TC312	Erfolgsmeldung, Daten in Maschine nachgetragen	Ja
TC313	Erfolgsmeldung, Daten in Maschine nachgetragen	Ja

Tabelle 14 - Systemtestprotokoll 14.12.2017

Verbesserungen

Testcase	Beschreibung	GitHub Issue
TC309, TC311	Datum nicht korrekt angezeigt. Wurde gefixt in Commit 157fbcf	-

Tabelle 15 - Verbesserungen Systemtest 14.12.2017

Usability-Test 04.12.2017

Angaben zur Durchführung

Usability-Test Objekt:	#56
Gerät:	Desktop
Testperson (TP):	Student HSR Informatik
Betreuer:	Luca Salzani
Beobachter:	Fabian Gübeli
Datum:	04.12.2017
Methodik:	Lautes Denken

Einführung

Herzlich Willkommen zu unserem Usability-Test oder für nicht Informatiker einfach Benutzerfreundlichkeitstest, wir danken dir schon im Voraus für die aufgebrachte Zeit. Durch diesen Test wollen wir auf keinen Fall dein Können oder sogar dich persönlich testen - Du kannst absolut nichts falsch machen.

Wir wollen mit diesem Test unsere Software unter die Lupe nehmen und allfällige Schwierigkeiten in der Benutzerführung beziehungsweise dem Design aufdecken.

Testablauf

Der Test wird wie folgt funktionieren:

- Es wird mit einer kurzen Einführung über die Software und ihre Funktionen begonnen um den Testpersonen einen Überblick und etwas Kontext zu geben.
- Der Testleiter gibt bekannt welcher Test Case durchgeführt wird.
- In jedem Test Case findest du eine Aufgabe, die du **selbstständig** in unserer Software lösen sollst.
- Während dem lösen der Aufgabe bitten wir dich, immer alle Gedankengänge laut zu denken, damit unser Beobachter Notizen machen kann.
- Hast du den Test abgeschlossen, kannst du dem Betreuer rufen und er wird sich das Ergebnis anschauen und Notizen machen, dies geschieht ohne dir eine Bewertung abzugeben.
- Bitte jetzt Warten bis alle Benutzer soweit sind, es wiederholt sich dann der ganze Ablauf mit einem neuen Test Case.

Durchführung

Aufgabe
Aufgabenstellung Als Servicetechniker hast du beim Kunden „Photojam“ an der Maschine mit der Seriennummer „XC3180065“ vom Typ „Endeavor“ eine Software „MS2.0“ installiert. Erfasse diese Änderung im entsprechenden Service Case als Task Rapport.
Notizen Beobachter - Sofortiges Auffinden der korrekten Maschine und des Service Cases - TP war nicht sofort klar, wo ein neuer Task Rapport eröffnet werden kann - Felder des Task Rapports wurden intuitiv ausgefüllt - Cause of error ist ein Pflichtfeld obwohl kein Fehler an der Maschine vorliegt, daher wurde es mit „keine“ ausgefüllt - TP dachte er wäre mit der Erfassung fertig bevor eine Modifikation erfasst wurde. Erst nach dem Speichern wurde es durch die TP bemerkt. - Erfassen der Modifikation ohne Probleme

Tabelle 16 - Test Case 1 Usabilitytestprotokoll

Aufgabe
Aufgabenstellung An der Maschine mit der Seriennummer „NK3592787“ des Typs „Express 2500“ vom Kunden „Omba“ muss aufgrund des Alters der Maschine das Teil „Lower Part For Turntable Z60“ ausgetauscht werden. Erfasse einen Service Case mit einem Problembeschreib und einen Task Rapport mit der entsprechenden Modifikation. Prüfe anschliessend in der Maschinenansicht ob das Alter des entsprechenden Maschinenteils zurückgesetzt wurde.
Notizen Beobachter - Sofortiges Auffinden der Maschine und der Möglichkeit einen Service Case zu erfassen da aus Test Case 1 bekannt - Felder des Service Cases sofort korrekt ausgefüllt - Nach Speicherung des Service Cases erwartet TP, dass er auf die vorhergehende Seite umgeleitet wird. TP ging über das Menü zur Maschine zurück anstatt über den Zurück-Button - Task Rapport gut ausgefüllt da aus Test Case 1 bekannt - Modifikation wurde korrekt erfasst, Navigation im BOM intuitiv - Alter des Teiles korrekt geprüft. Diesmal über den Zurück-Button

Tabelle 17 - Test Case 2 Usabilitytestprotokoll

Aufgabe
Aufgabenstellung An der Maschine mit der Seriennummer „NK3592787“ des Typs „Express 2500“ vom Kunden „Omba“ wird ein Upgrade durchgeführt. In die Baugruppe „CAD-LA000031“ soll ein „Technic 5M Beam“ eingebaut werden. Erfasse einen Service Case mit dem Problembeschreib „Upgrade“ und einen Task Rapport mit der entsprechenden Modifikation. Prüfe anschliessend in der Maschinenansicht ob das Teil korrekt angezeigt wird.
Notizen Beobachter - Service Case gut erfasst da aus Test Case 2 bekannt - Task Rapport korrekt ausgefüllt - Bei der Modifikation war aus der Aufgabenstellung nicht herauszulesen wo im BOM das Teil eingefügt werden muss. Daher wurde es im Root hinzugefügt. Nach Nachfragen beim Betreuer konnte die TP das Teil mittels Modifikation wieder entfernen und am korrekten Ort einfügen - Alter des Teils korrekt geprüft

Tabelle 18 - Test Case 3 Usabilitytestprotokoll

Aufgabe
Aufgabenstellung Du bist nun ein Systemverantwortlicher. Deine Firma hat einen neuen Maschinentyp mit dem Modell „M42“ entwickelt. Dieser soll nun im System erfasst werden. Dafür wirst du zusätzliche Dateien benötigen. Diese bekommst du vom Betreuer.
Notizen Beobachter - Durch Menü auf korrekte Darstellung navigiert - Ausfüllen der Felder intuitiv durchgeführt - Verwirrung bei den Begriffen „Initial BOM“ und „Part Catalog“. Dies ist darauf zurückzuführen, dass die TP keinen Bezug zur Branche hat. - Korrektes Hochladen der Dateien

Tabelle 19 - Test Case 4 Usabilitytestprotokoll

Aufgabe
Aufgabenstellung Im letzten Test Case hast du einen neuen Maschinentyp erfasst. Nun werden zwei Maschinen dieses Typs ausgeliefert. Erstelle diese im System. Die Maschinen sollen an den Kunden „Skippad“ geliefert werden und haben die Seriennummern „UO7148000“ bzw. „UO7148001“, wurden heute Produziert und werden mit der Softwareversion V1 ausgeliefert.
Notizen Beobachter - TP suchte zuerst im Maschinentyp nach der Funktion eine Maschine hinzuzufügen. - Erst als Zweites wurde im Kunden gesucht. - Erfassen der Maschinen intuitiv - TP überprüfte selbständig die BOM der erstellten Maschinen

Tabelle 20 - Test Case 5 Usabilitytestprotokoll

Anhang C – Installationsanleitung

Diese Installationsanleitung ist nicht für Benutzer der Software gedacht sondern für Entwickler welche die Applikation weiterentwickeln oder Betreuer welche die Arbeit bewerten und daher debuggen/starten werden. Am Ende der Anleitung wird noch erläutert wie die Applikation lokal gestartet werden kann.

Vorbedingungen Client

- Visual Studio 2017 inkl. folgender Workloads
  - .NET Desktopentwicklung
  - ASP.NET und Webentwicklung
  - .NET Framework 4.6.1 SDK
  - Plattformübergreifende .NET Core-Entwicklung
- Google Chrome ab Version 56
- Angular IDE nach Wahl – Entwickelt wurde mit WebStorm 2017

Vorbedingungen Server

- MS SQL Express Server ab Version 13
  - SQL Server Management Studio
- Internet Information Services (IIS) Version 10.x

Installation

1. Erstellen eines neuen Benutzers im SQL Server Management Studio mit den Rollen „public“ und „dbcreator“.

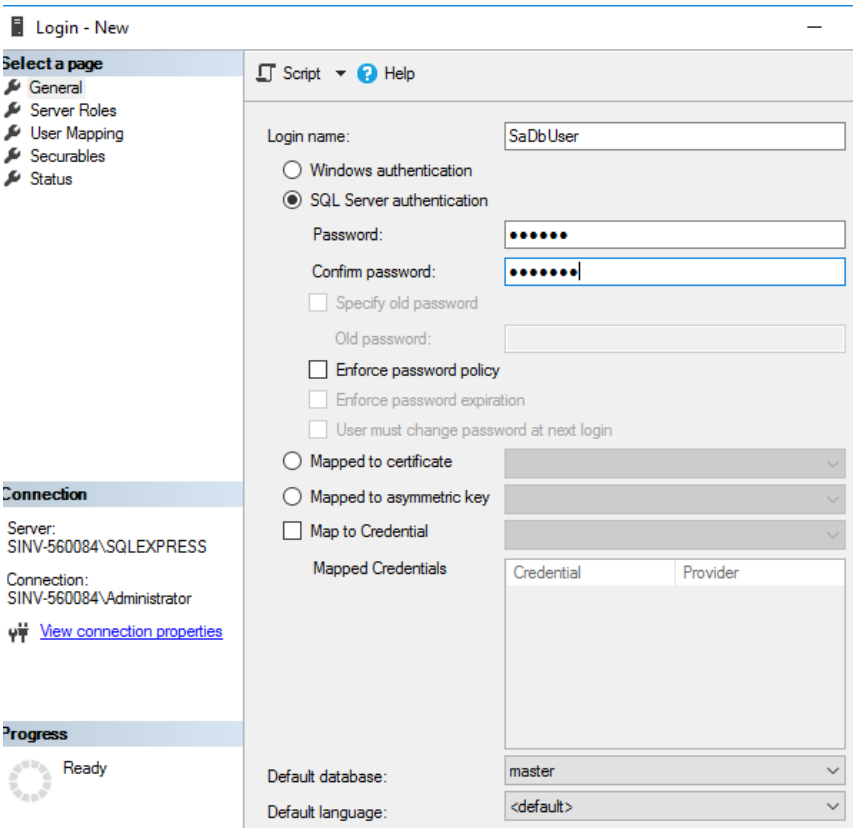

Abbildung 13 - Microsoft SQL Server Benutzererstellung

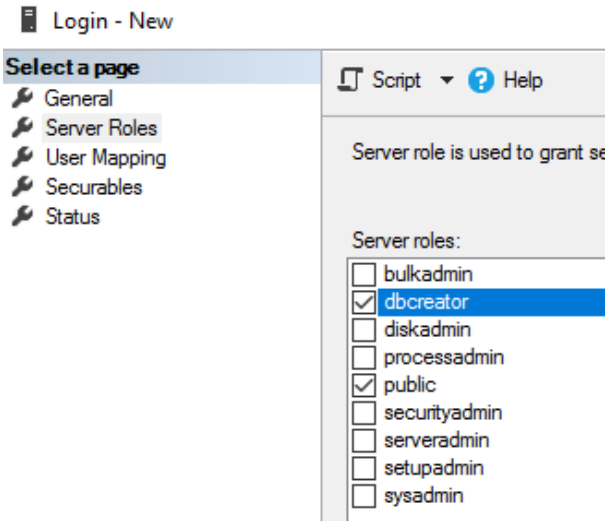

Abbildung 14 - Microsoft SQL Server Benutzerrollen

2. Hinzufügen folgender Pakete im IIS über den Server Manager.

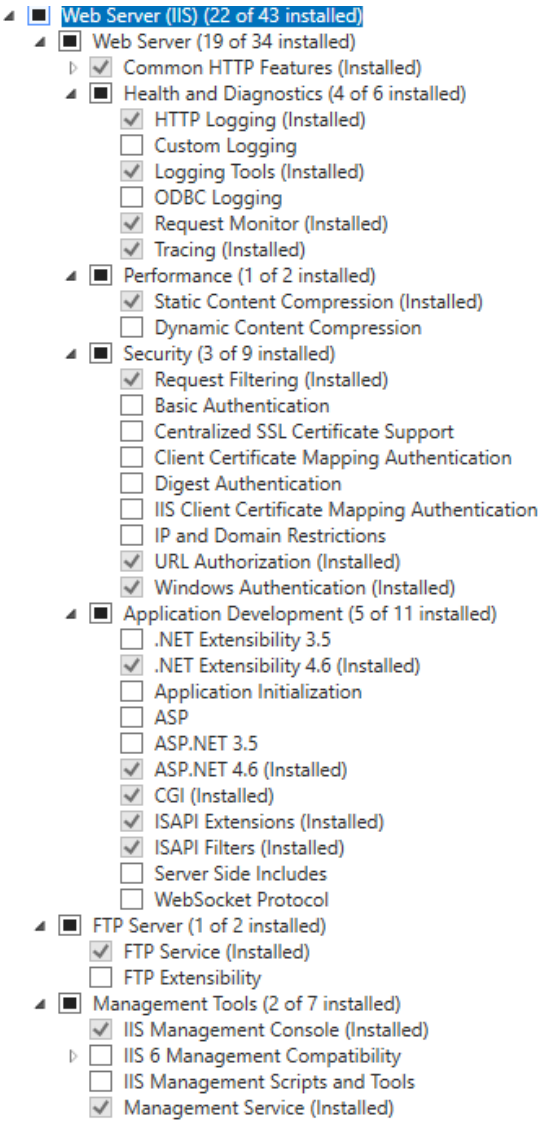

Abbildung 15 - IIS Pakete

3. Erstellen der Application Pools für Backend und Frontend.

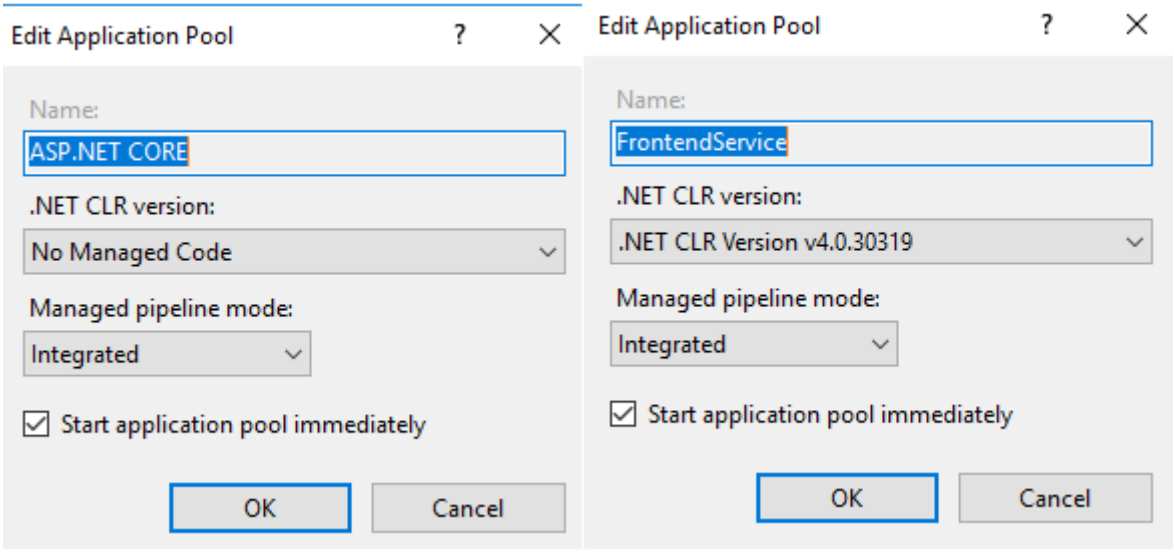

Abbildung 16 - IIS Application Pools

4. Erstellen der Sites für Frontend und Backend. Dazu sollten Pfade im System für das Deployment definiert werden. Hier auf dem Desktop. Es sind die Application Pools für Backend (ASP.NET CORE) und Frontend (FrontendService) auszuwählen.

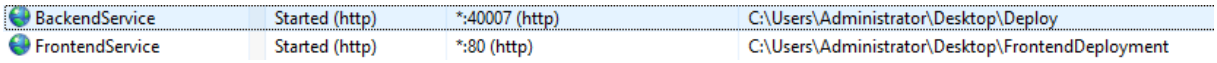

Abbildung 17 - IIS Site Configuration

5. Erstellen eines Shares, damit die Nutzer «Everyone» Lesezugriff auf das Frontend erhalten.

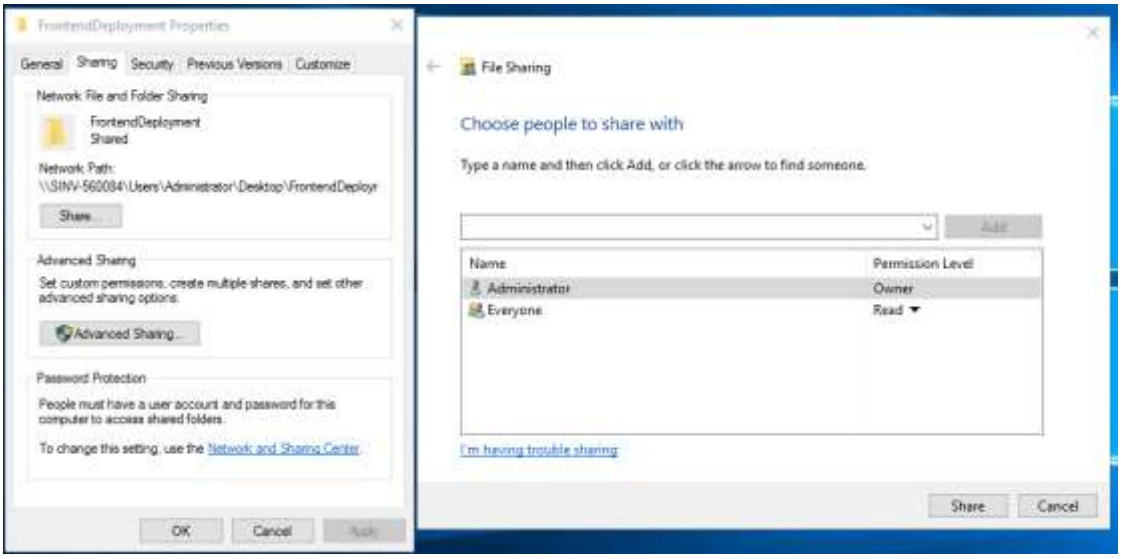

Abbildung 18 - Folder Sharing Options

6. Anpassen des Connection-Strings in „BackendService/appsettings.json/appsettings.Production.json“ auf den korrekten Server, User und Passwort. Der Name der Datenbank kann frei gewählt werden.

Abbildung 19 - Anpassung des Connection Strings im Backend

7. Build des Backends. Über die Kommandozeile im Order in dem sich die Solution befindet.

```
dotnet publish BackendService -o <GEWÜNSCHTER PFAD>
```

8. Kopieren des Inhalts aus dem Outputfolder in den Ordner auf dem Server des Backend Service.

9. Festlegen der apiURL in der Datei «Frontend/src/environments/environment.prod.ts» im WebStorm. Dazu muss im WebStorm der Frontend als Root Ordner geöffnet werden.

```
import {HttpHeaders} from '@angular/common/http';

export const environment = {
  production: true,
  apiUrl: 'http://sinv-56084.edu.hsr.ch:40007/api',
  JsonHeader: new HttpHeaders().set('Content-Type', 'application/json'),
  XmlHeader: new HttpHeaders().set('Content-Type', 'application/xml')
};
```

Abbildung 20 - Konfiguration der API URL im Frontend

10. Build des Frontends. Im Terminal von Webstorm.

```
npm install -g @angular/cli@latest
```

```
npm install
```

```
ng build --prod --aot
```

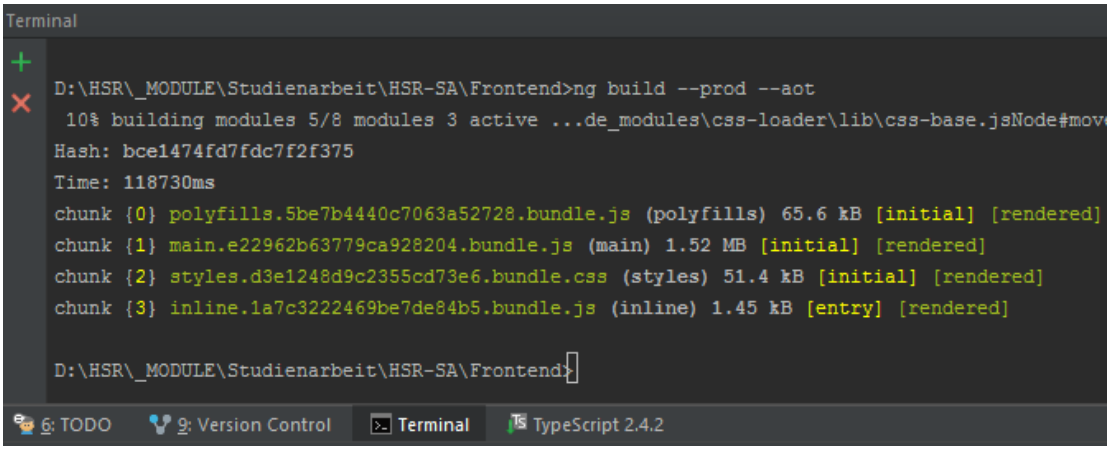

Abbildung 21 - Build des Frontends

11. Kopieren des Inhalts aus dem Ordner „dist“ im Frontendverzeichnis in den Ordner auf dem Server des Frontend Services.

12. Neustart des IIS Servers. Nun wird die DB initialisiert und das Frontend kann über Port 80 erreicht werden.

Lokale Ausführung

1. Öffnen der Backend Solution in Visual Studio. Falls die Applikation nur lokal ausgeführt werden soll kann durch einen Klick auf „IIS Express“ lokal gestartet werden. Dabei wird die lokale MSSQL Datenbank verwendet. Diese wird initialisiert. Ein Chrome Fenster mit Swagger startet nun. Das Backend ist betriebsbereit.

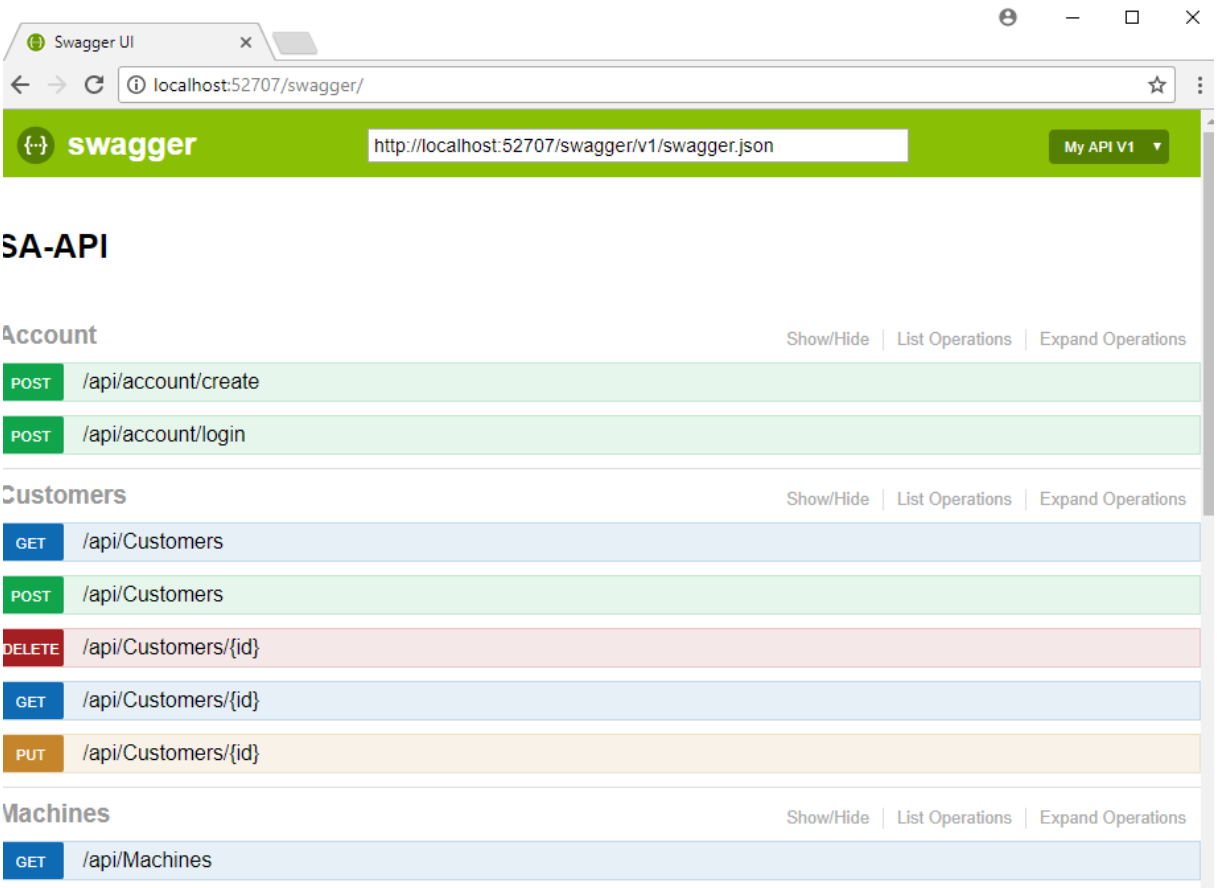

Abbildung 22 - Backendübersicht Swagger

2. Öffnen des Frontend in WebStorm. Ändern der apiURL in der Datei «Frontend/src/environments/environment.ts». Der Port muss bei Swagger in der URL nachgeschaut werden.

```
6   import {HttpHeaders} from '@angular/common/http';
7
8   export const environment = {
9     production: false,
10    apiUrl: 'http://sinv-56084.edu.hsr.ch:40007/api',
11    JsonHeader: new HttpHeaders().set('Content-Type', 'application/json'),
12    XmlHeader: new HttpHeaders().set('Content-Type', 'application/xml')
13  };
```

Abbildung 23 - Konfiguration der API URL im Frontend für lokale Benutzung

3. Start der Applikation über das Terminal. Das Frontend ist nun über localhost:4200 erreichbar.

```
npm start
```

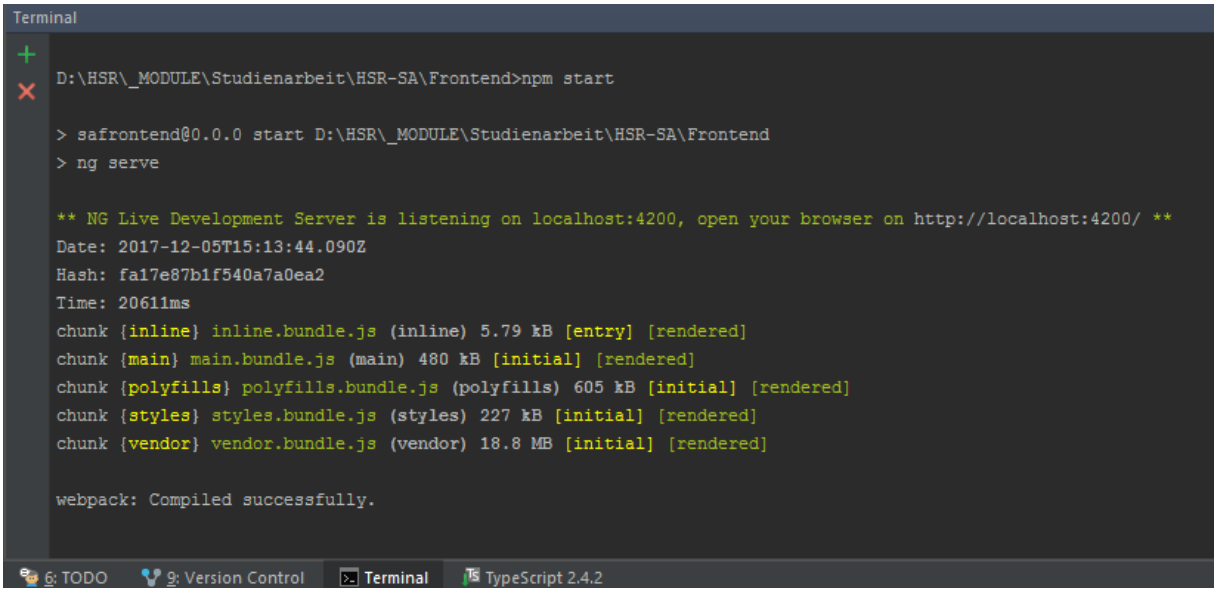

Abbildung 24 - Starten des Frontends in WebStorm