Friedrich-Alexander-Universität Erlangen-Nürnberg

Technische Fakultät, Department Informatik

THI YEN LE

MASTER THESIS

ANWENDUNG VON NATURAL LANGUAGE PROCESSING ZUR EVALUIERUNG DER ANFORDERUNGSBESCHREIBUNGEN

Eingereicht am 15. Juli 2021

Betreuer: Prof. Dr. Dirk Riehle, M.B.A.

Julia Krause, M.Sc.

Professur für Open-Source-Software

Department Informatik, Technische Fakultät

Friedrich-Alexander University Erlangen-Nürnberg

Versicherung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

_____

Nürnberg, 15. Juli 021

License

_____

Nuremberg, 15. Juli 2021

Abstract

The main reasons for software project failure are the lack of user involvement, incomplete requirements, unrealistic expectations, and poor quality of requirements specification. A requirements template is a blueprint for the syntactic structure of individual requirements. The application of requirements templates is an effective and simple approach for reducing the ambiguity of natural language requirements and thus increases the precision of the requirements and the quality of the requirements specification. The requirements templates define the syntax for requirements description and are applied for evaluating, whether the requirements follow this syntax. But the manual evaluating of the conformance is a challenge when dealing with a huge amount of requirements. Manual evaluation is time-consuming and expensive. This thesis presents an approach for automated evaluating the compliance of requirement descriptions using natural language processing techniques and requirements templates. This increases the quality of the requirements and project success.

Zusammenfassung

Die Hauptgründe für das Scheitern bei der Umsetzung der Softwareprojekte sind beispielweise unvollständige Anforderungen, Mehrdeutigkeit der Anforderungsbeschreibungen sowie schlechte Anforderungsqualität. Die Verwendung einer Anforderungsschablone ist ein effektiver Ansatz zur Verbesserung der Qualität von den natursprachlichen Anforderungen, um die Mehrdeutigkeit bei dem Interpreter zu reduzieren und dadurch die Qualität der Anforderungen zu verbessern. Die Anforderungsschablone definiert hierbei die Syntax für die Anforderungen und wird zur Überprüfung angewendet, ob die Anforderungen dieser Syntax folgen. Bei einer großen Menge der Anforderungen ist es eine Herausforderung, wenn diese Überprüfung manuell durchgeführt wird. Die Nutzer müssen diese Aufgabe mehrmals wiederholen und es führt zu einem Zeit- und Kostenverlust. Diese Arbeit präsentiert einen Ansatz zur automatischen Überprüfung nach Konformität der Anforderungsbeschreibungen mit der Anwendung der Natural Language Processing Techniken und Anforderungsschablonen. Dieser Ansatz hilft, die Qualität der Anforderungen zu erhöhen und bringt einen Mehrwert für Softwareprojekte.

Inhaltsverzeichnis

Abkürzungsverzeichnis

CRUD	Create, Read, Update, Delete
DOM	Document Object Model
EMS	Enhanced Voice Mail System
EARS	Easy Approach to Requirements Syntax
GCP	Google Cloud Platform
HTTP	Hypertext Transfer Protocol
IOB	Inside-Outside-Beginning
IE	Information Extraction
KI	Künstliche Intelligenz
MASTeR	Mustergültige Anforderungen- die SOPHIST Templates für Requirerments
NL	Natural Language
NLP	Natural Language Processing
POS	Part-Of-Speech
RE	Requirements Engineering
REST	Representational State Transfer
SDLC	Software Development Life Cycle
URI	Uniform Ressource Identifiers

Abbildungsverzeichnis

Tabellenverzeichnis

Quelltextverzeichnis

1 Einleitung

1.1 Motivation

Requirements Engineering (RE) ist ein wichtiger und initialer Bestandteil für jedes Softwareprojekt. RE hat zur Aufgabe, die Anforderungen des zu entwickelnden Produkts auf Basis der Kundenwünsche festzuhalten (Kof, 2005). Wenn Fehler in der RE Phase auftreten, kann das möglicherweise zu einem Projektmisserfolg führen oder müssen die in späteren Phasen korrigiert werden (Kof, 2005). Dies kann zu einer Kostenerhöhung und zu einem Zeitverlust im Projekt führen.

Unvollständige Anforderungen, mangelnde Benutzerbeteiligung, übermäßige Änderung bei den Anforderungen und schlechte Anforderungsqualität sind die Indikatoren in der RE Phase, die meistens auf das Scheitern eines Softwareprojekts einwirken (M. . Singh & Vyas, 2012). In einer online-Umfrage von 151 Software-Unternehmen haben Mich et al. (Mich et al., 2004) erfasst, dass 95% der Anforderungen in den Softwareprojekten in irgendeiner Form von natürlicher Sprache (*eng. Natural Language; NL*) wie Text mit Schlüsselwörtern oder Tabellen formuliert werden. Natürliche Sprache ist flexibel, weit verbreitet, universell, leicht anwendbar und interpretierbar. Damit können alle Stakeholder aus dem Projekt ohne Probleme die Anforderungen an die Software und das System verstehen. Aber diese in natürliche Sprache formulierten Anforderungen können unvollständig, inkonsistent und mehrdeutig sein (Kamsties & Paech, 2000). Außerdem ist es schwer, einen Überblick bei einer großen Sammlung von Anforderungen zu behalten, um Inkonsistenzen, Redundanzen oder fehlenden Anforderungen zu finden (Dalpiaz et al., 2018).
Anforderungsschablonen sind einfach erlernbar und leicht einsetzbarer Ansatz zur Reduzierung der sprachlichen Effekten bei der Formulierung der Anforderungen, (Pohl & Rupp, 2015a). Durch die Anwendung der Anforderungsschablone kann die syntaktische Eindeutigkeit der Anforderung erreicht werden (Pohl & Rupp, 2015a).

1.2 Problemstellung

Die Anforderungsschablone definiert hierbei die Syntax einer Anforderung und wird zur Erstellung und Überprüfung angewendet, ob die Anforderungen dieser Syntax folgen. Die Überprüfung nach Konformität der Anforderungen und Anforderungsschablone ist eine

Herausforderung, wenn es manuell bei einer großen Sammlung von Anforderungen durchgeführt wird. Mit Anwendung von natürlicher Sprachverarbeitung Techniken (*engl. Natural Language Processing; NLP)* wird dieser Prozess automatisiert. Außerdem hilft die natürliche Sprachverarbeitung bei der Identifizierung von den Qualitätsmängeln, Mehrdeutigkeiten von den Anforderungen und der Extraktion von Schlüsselwörtern (Dalpiaz et al., 2018).

In dieser Arbeit wird die Frage beantwortet, wie NLP Techniken und Anforderungsschablone angewendet werden, um die Anforderungsbeschreibungen der Softwareprojekten automatisiert zu kontrollieren und die Probleme bei der Formulierung der Anforderungen zu mildern.

1.3 Aufbau der Arbeit

Die vorliegende Arbeit ist in neun Kapitel gegliedert. Begonnen wird mit den Grundlagen der Anforderungen und des NLPs in RE. Weiterhin wird auf den Stand der Wissenschaft zu den verschiedenen Anforderungsschablonen eingegangen. Die Anforderungsschablonen werden miteinander verglichen, um eine passende Anforderungsschablone für die spätere Implementierung zu identifizieren. Das vierte Kapitel beschäftigt sich mit den funktionalen und nicht-funktionalen Anforderungen in dieser Arbeit. Die Architektur und das Design der Anwendung werden im fünften Kapitel erläutert. Das sechste Kapitel beschäftigt sich mit der Implementierung basierend auf der Architektur der Anwendung. Im siebten Kapitel wird sich mit der Evaluierung befasst, bei der die Anwendung nach den im vierten Kapitel beschriebenen Anforderungen bewertet werden. Im achten Kapitel werden die Stärken bzw. Schwächen zu der entwickelten Anwendung diskutiert. Schließlich wird im neunen Kapitel eine Zusammenfassung der gesamten Arbeit gezogen.

2 Grundlagen

Im Folgenden wird auf die Themen Anforderung (*engl. Requirement*) und NLP eingegangen. Im Kapitel 2.1 wird der Begriff *Anforderung* und die wesentlichen Qualitätskriterien für eine Anforderung erläutert. Im Kapitel 2.2 wird in NLP und die NLP-Techniken eingeführt, welche die Grundlage zur Entwicklung der Anwendung in dieser wissenschaftlichen Arbeit darstellt.

2.1 Grundlage zu Anforderungen

Anforderungen spielen eine essenzielle Rolle im RE und für die Systementwicklung. Darüber hinaus haben sie einen Einfluss beim Entwurf oder im Test direkt oder indirekt auf das Projektgeschehen (Pohl & Rupp, 2015a).

Nach IEEE Standard Glossary of Software Engineering Terminology (IEEE Standards Board, 1990) wird eine Anforderung wie folgt definiert:

> " (*1) A condition or capability needed by a user to solve a problem or achieve an objective.*
>
> *(2) A condition or capability that must be met or possessed by a systems or system component to satisfy a contract, standard, specification, or other formally imposed documents.*
>
> *(3) A documented representation of a condition or capability as in (1) and (2)."*

Die Anforderungen werden in zwei Kategorien unterteilt, funktionale und nicht-funktionale Anforderungen.

- *Funktionale Anforderungen* beschreiben die Funktionalitäten, die das System oder die Systemkomponente leisten müssen (IEEE Standards Board, 1990). Die Anforderungen können in funktionalen Anforderungen, Verhaltensanforderungen und Datenanforderungen zugeordnet werden (Pohl & Rupp, 2015b).
- *Nicht-funktionale Anforderungen* legen fest, welche Qualitätseigenschaften das System haben soll und welche Randbedingungen an das System gestellt werden (Grande, 2014).

Die Qualität einer Anforderung muss nach bestimmten Kriterien überprüft und abgesichert werden. Das nächste Kapitel stellt hierzu die Qualitätskriterien zur Überprüfung der funktionalen und nicht-funktionalen Anforderungen dar.

2.1.1 Qualitätskriterien

Chris Rupp und Klaus Pohl (Pohl & Rupp, 2015a) haben anhand dem Standard ISO/IEC/IEEE 29148:2011 (ISO/IEC/IEEE 29148: 2011, 2011) die Qualitätskriterien für die System- und Softwareanforderungen definiert. Die Tabelle 1 veranschaulicht die Qualitätskriterien für jede einzelne Anforderungen und deren Beschreibungen.

Qualitätskriterien	Beschreibungen
eindeutig *(engl. unambiguous)*	Jede Anforderung soll so formuliert werden, damit diese nur auf eine Art und Weise interpretiert werden kann.
notwendig (*engl. necessary*)	Jede Anforderung muss die Gegebenheiten im Systemkontext widerspiegeln, dass diese Anforderung hinsichtlich der aktuellen Gegebenheiten im Systemkontext gültig ist. Die Gegebenheiten können sich beispielweise auf die Schnittstellen der externen Systeme oder auf die relevanten Standards beziehen.
vollständig *(engl. complete)*	Jede Anforderung muss die geforderte und zu erfüllende Funktionalität vollständig beschreiben. Keine weitere Information ist notwendig, um die Anforderung zu verstehen.
atomar *(engl. singular)*	Jede Anforderung beschreibt eine einzelne Fähigkeit, Bedingung, Qualitätsfaktor oder Charakter. Daher ist es nicht möglich, die in weiteren Anforderungen zu zerlegen.
realisierbar (*engl. feasible)*	Jede Anforderung muss innerhalb von Systembeschränkungen beispielweise Kosten, Plan, Techniken mit akzeptablen Risiken umgesetzt werden.
prüfbar *(engl. verifiable)*	Jede Anforderung muss so formuliert werden, dass sie prüfbar ist.
konsistent *(engl. consistent)*	Jede Anforderung muss gegenüber anderen Anforderungen widerspruchsfrei sein.
verfolgbar *(engl. traceable)*	Jede Anforderung muss von dem Ursprung bis zur Umsetzung und deren Beziehung zu anderen Dokumenten eindeutig identifizierbar sein.

Tabelle 1 Übersicht über Qualitätskriterien für einzelne Anforderungen (Pohl & Rupp, 2015a)

Die Anforderungen jeder Software- und Systemprojekten sollen nach sogenannten Qualitätskriterien folgen.

2.2 Grundlage zu Natural Language Processing im Requirements Engineering

Natürliche Sprachverarbeitung (*engl. Natural Language Processing; NLP)* ist ein wichtiges Teilgebiet der künstlichen Intelligenz (KI) und Linguistik. Mit Hilfe von NLP können wir mit dem Computer durch natürliche Sprache interagieren und diesen steuern. Das bekannteste Beispiel für die Anwendung von NLP im Alltag sind Sprachassistenten, mit denen Computer, Smartphone oder Tablet die Informationen durch Spracherkennung extrahieren und darauf die entsprechenden Befehle ausführen. Weitere Beispiele sind Übersetzungsprogramme wie Google Translate [1] oder Spam-Filter [2] usw.

Liddy (Liddy, 2001) hat den Begriff NLP wie folgt definiert:

> „*Natural Language Processing is a theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human-like language processing for a range of tasks or applications.“*

Darunter kann man verstehen, dass NLP die Interaktion des Computers mit der menschlichen Sprache ist, um eine menschenähnliche Sprachverarbeitung für die Aufgaben oder Anwendungen zu erzielen. Die Interaktion umfasst Lernen, Analyse, Verstehen, Erkennen und die Generierung der menschlichen Sprache (Bauer & Warschat, 2020).

NLP werden in unterschiedlichen Phasen eines Softwareentwicklung-Lebenszyklus (*engl. Software Development Life Cycle; SDLC*) angewendet (Sharma, 2018). Insbesondere in der Requirements Engineering Phase spielt NLP eine wichtige Rolle, da in diese Phase die Anforderungsdokumente an die zu entwickelnde Software und Systeme definiert werden. NLP-Techniken, Tools und Ressourcen können auf definierten Anforderungen angewendet werden, um die linguistischen Analyseaufgaben zu unterstützen (Zhao et al., 2020). Zu den Analyseaufgaben zählen beispielweise die Erkennung von Sprachproblemen, Identifizierung der Schlüsseldomänenkonzepten oder Rückverfolgbarkeit von den Anforderungen, damit die Probleme bei der Formulierung der Anforderungen frühzeitig entdeckt und rechtzeitig behoben

[1] https://translate.google.com/
[2] https://de.wikipedia.org/wiki/Spamfilter

werden können (Zhao et al., 2020). Es gibt drei Arten von Sprachverarbeitungstechnologien, *NLP-Ressource, NLP-Tool* und *NLP-Technik,* die im Folgenden beschrieben werden (Zhao et al., 2020):

- Eine *NLP-Ressource* ist eine linguistische Datenressource zur Unterstützung der NLP-Technik und NLP-Tool
- Eine *NLP-Technik* ist eine praktische Methode, Ansatz, Prozess oder Prozedur zur Ausführung einer NLP-Aufgabe.
- Ein *NLP-Tool* ist eine Softwaresystem oder eine Softwarebibliothek wie CoreNLP[3], OpenNLP[4] oder NLTK[5], die eine oder mehrere NLP-Techniken unterstützt.

Das nächste Kapitel beschreibt die NLP-Techniken, welche zur Entwicklung dieser Arbeit wesentlich sind.

2.2.1 Natural Language Processing Techniken

Die Beschreibungen von Anforderungsdokumenten werden als unstrukturierte Daten bezeichnet. Diese unstrukturierten Daten müssen vorerst zu strukturierten Daten konvertiert werden, um die essentiellen Bedeutungen aus den Texten zu extrahieren und diese anschließend nach konformen und nicht-konformen Anforderungen zu klassifizieren. Dieser Prozess ist Aufgabe der Informationsextraktion *(engl. Information Extraction; IE)* (Steinkamp, 2019). Die Informationsextraktion verwendet die Methode zur Identifizierung der relevanten Informationen in Dokumente oder Textdateien, um diese nach bestimmten Kriterien für verschiedene Zwecke darzustellen (S. Singh, 2010). IE wird in *traditionelle IE* und *Open IE* wie folgt unterteilt (Villavicencio et al., 2018).

- *Traditionelle IE* versucht, automatisch die nützlichen Muster in Dokumente herauszufinden und fokussiert sich auf kleine, homoge Domänen.
- *Open IE* erzielt, die semantischen Informationen in Texte von verschiedenen Domänen in Form von Phrasen zu extrahieren.

[3] https://nlp.stanford.edu/software/
[4] https://opennlp.apache.org
[5] https://www.nltk.org

Glauber und Barreiro Claro (Glauber & Barreiro Claro, 2018) haben eine Open IE Pipeline (siehe Abbildung 1) vorgestellt, welche die wesentlichen Schritte und Techniken für die Informationsextraktion definiert.

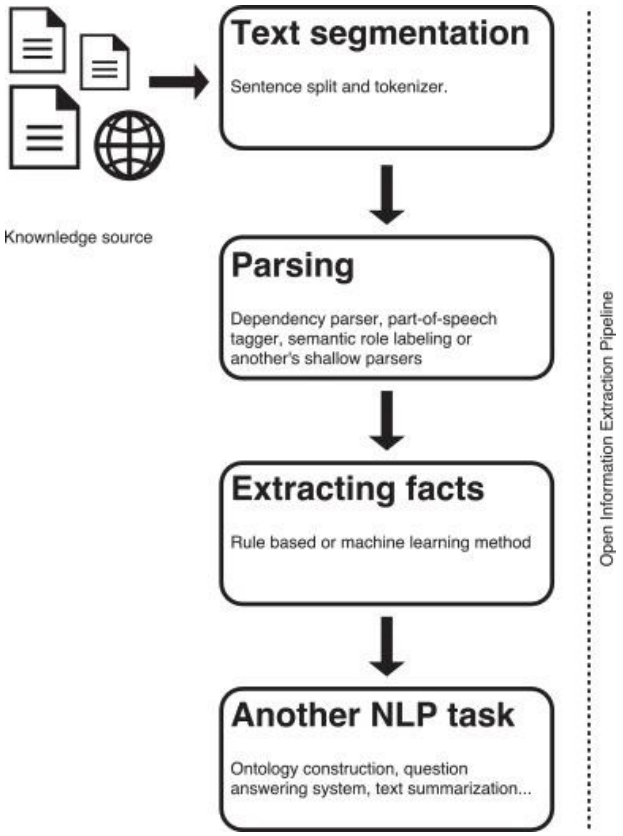


Abbildung 1 Übersicht über Open Information Extraction Pipeline (Glauber & Barreiro Claro, 2018)

Anhand dieser Pipeline wird der Informationsextraktionsprozess für die Anforderungsanalyse dieser Arbeit entwickelt. In *Text Segmentation* Phase wird jedes Anforderungsdokument auf einzelne Sätze aufgetrennt. Danach wird jeder Satz auf einzelne Wörter segmentiert. In der *Parsing* Phase werden verschiedene Techniken wie Part-Of-Speech (POS) Tagging, Parse und Chunking verwendet, um die Eingaben für *Extracting Facts* Phase zu erzeugen. In *Extracting Facts* Phase werden die Eingaben überprüft, ob diese mit den Regeln bzw. regulären Ausdrücken der ausgewählten Anforderungsschablone konform ist. Dadurch kann die Konformität der Anforderungen bestimmt werden.

Um ein besseres Verständnis über die Funktionalität der sogenannten NLP-Techniken zu schaffen, beschreiben die nächsten Kapitel die Techniken genauer.

2.2.1.1 Tokenizing

Ein **Token** ist eine Bezeichnung für eine Sequenz von Zeichen und ist möglicherweise ein Wort, eine Zahl oder ein Sonderzeichen (Bird et al., 2009).

Tokenisierung (*engl. Tokenizing*) ist die Zerlegung eines Satzes oder eines Dokuments in einzelne Token (Bauer & Warschat, 2020). Die Zerlegung durch Leerzeichen ist das einfachste Tokenisierungsverfahren, wobei der Text an den Leerzeichnen aufgetrennt wird. Zur Verdeutlichung der Aufgabe von Tokenisierung wird folgender Satz als Beispiel gewählt (Bird et al., 2009):

„The Project Gutenberg EBook of Crime and Punishment, by Fyodor Dostoevsky“

Nach Tokenisierung entsteht eine Liste von Tokens:

[‚*The’, ‘Project’, ‘Gutenberg’, ’EBook’, ’of’, ‘Crime’, ‘and’, ‘Punishment’, ‘,’, ‘by’, ‘Fyodor’, ‘Dostoevsky’*]

2.2.1.2 Part-Of-Speech Tagging

Jedes Token nach Tokenisierung erhält eine Markierung der Wortarten (*engl. Part-Of-Speech; POS*). Es gibt insgesamt acht Wortarten wie Normen, Verben, Adjektive, Adverben, Pronomen, Präpositionen, Konjunktionen und Interjektionen (Bauer & Warschat, 2020). Abhängig von der Sprache wurden verschiedenen Sets von Markierung (*engl. tag sets*) zur Auszeichnung der Wortarten entwickelt. In der deutschen Sprache wird STTS-Tag-Set[6] und Penn Treebank-Tag-Set[7] in der englischen Sprache verwendet. In Rahmen dieser Arbeit wird das Penn Treebank-Tag-Set (siehe Abbildung 2) zur Implementierung der Anwendung verwendet.

[6] https://www.linguistik.hu-berlin.de/de/institut/professuren/korpuslinguistik/mitarbeiter-innen/hagen/STTS_Tagset_Tiger
[7] https://www.ling.upenn.edu/courses/Fall_2007/ling001/penn_treebank_pos.html

Tag	Description	Example
CC	coordinating conjunction	and, but, or
CD	cardinal number	one, two
DT	determiner	a, the
EX	existential ,, there"	There
FW	foreign word	Mea
IN	preposition/subordin - conjunction	of, in, by
JJ	adjective	Yellow
JJR	comparative adj.	Biggerst
JJS	superlative adj.	Bigger
LS	list item marker	1, 2, one
MD	modal	Can, should
NN	singular or mass noun	Llama
NNS	noun, plural	Llamas
NNP	proper noun. Singular	IBM
NNPS	proper noun plural	Carolinas

Tag	Description	Example
PDT	predeterminer	all, both
POS	possessive ending	`s
PRP	personal pronoun	I, you, he
PRP $	possess. pronoun	your,
RB	adverb	quickly
RBR	comparative adverb	Faster
RBS	superlatv. adverb	fastest
RP	particle	up, off
SYM	symbol	+, %, &
TO	,,to"	to
UH	interjection	ah, oops
VB	verb base form	eat
VBD	verb past tense	ate
VBG	verb gerund	eating
VBN	verb past part.	Eaten

Tag	Description	Example
VBP	verb, non- 3sg present	eat
VBZ	verb3sg p res	eats
WDT	wh-determ.	which, that
WP	wh-pronoun	what, who
WP$	wh-possess.	whose
WRB	wh-adverb	how, where
$	dollar sign	$
#	pound sign	#
„	left quote	„or"
„	right quote	„or"
(	left paren	[, (, <
)	right paren	],), >
,	comma	,
.	sent-end punc	., !, ?
:	sent-mid punc	„ ., ... , -

Abbildung 2 Penn Treebank POS-Tags (Bauer & Warschat, 2020)

Wortarterkennung (*engl. Part-Of-Speech Tagging; POS Tagging*) ist ein Prozess zur Zuordnung von Tokens (Worte, Satzzeichen) zu den entsprechenden Wortarten, um die Mehrdeutigkeiten von den Worten aufzulösen (Jurafsky & Martin, 2020). Das Ergebnis der Wortarterkennung wird in eine Liste von POS-Tags dargestellt. Die Abbildung 3 zeigt das Ergebnis nach POS Tagging des im Abschnitt 2.2.2.1 beschriebenen Beispielsatzes:

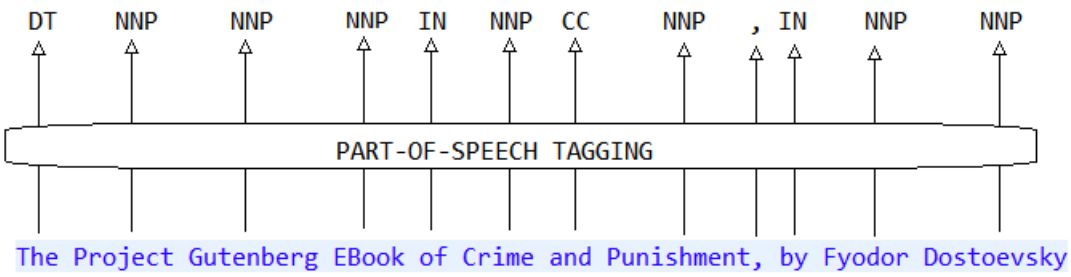


Abbildung 3 Ergebnis nach POS-Tagging

2.2.1.3 Parsing und Parse Trees

Neben Tokenisierung und Wortarterkennung wird Parsing häufig in der natürlichen Sprachverarbeitung eingesetzt. Mit Parsing ist es möglich, die Abhängigkeit von den Worten untereinander darzustellen und dadurch die syntaktische Struktur des Satzes zu ermitteln (Bauer & Warschat, 2020). Ein Zerteiler (*engl. Parser*) ist ein Computerprogramm, das die eingegebenen Sätze in natürliche Sprache zu einer baumartigen Struktur zuordnet (Hellwig, 1989). Diese Struktur wird als Syntaxbaum (*engl. Parse Tree*) bezeichnet.

Für den im Abschnitt 2.2.1.1 beschriebenen Beispielsatz wird ein Syntaxbaum (siehe Abbildung 4) erzeugt, wobei *S* für den eingegebenen Satz, *NP* für Normen Phrase und *PP* für Präposition Phrase und die anderen für in Abbildung 2 dargestellten POS-Tags stehen.

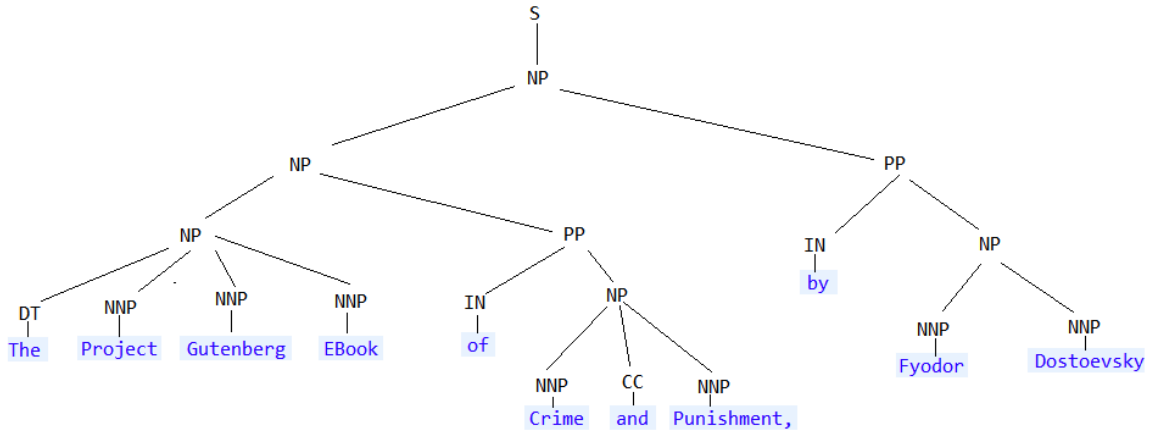


Abbildung 4 Der Syntaxbaum des Beispielsatzes

2.2.1.4 Chunking

Phrasen (*engl. Chunks*) sind die flachen, nicht überlappenden Segmente im Satz und werden in vier Arten wie Nominalphrase, Verbalphrase, Adjektivphrase und Präpositionalphrase unterteilt (Jurafsky & Martin, 2020). Die Worte, die in einer Phrase eingeteilt wurden, spielen gemeinsam eine Rolle im Satz.

Phrasenerkennung (*engl. Chunking*) ist ein Prozess zum Identifizieren und Klassifizieren der Phrasen (Jurafsky & Martin, 2020). Das Ergebnis der Phrasenerkennung wird entweder durch eine Markierung oder einer baumartigen Struktur repräsentiert (Bird et al., 2009). IOB **(***engl. Inside-Outside-Beginning)* Markierung wird häufig zur Repräsentation des Ergebnisses von der Phrasenerkennung verwendet. Ein Token wird mit *B*-Suffix markiert, wenn dieser am Anfang einer Phrase ist. Die weiteren Tokens derselben Phrase werden mit *I*-Suffix markiert. Die Tokens werden mit *O*-Suffix bezeichnet, wenn die zu keiner Phrase zugeordnet werden. Der

Beispielssatz aus dem Abschnitt 2.2.1.1 wird nach Phrasenerkennung mit IOB Markierung in Abbildung 5 umgewandelt.

B-NP	I-NP	I-NP	I-NP	B-PP	B-NP	I-NP	I-NP	O	B-PP	B-NP	I-NP
The	Project	Gutenberg	EBook	of	Crime	and	Punishment	,	by	Fyodor	Dostoevsky

Abbildung 5 Ergebnis nach Phrasenerkennung des Beispielsatzes

3 Stand der Wissenschaft zu Anforderungs-schablonen

„A requirements template is a blueprint for the syntactic structure of individual requirements“ (Pohl & Rupp, 2015b).

Laut Chris Rupp und Rainer Joppich ist eine Anforderungsschablone ein Bauplan, der die Struktur jeder einzelnen Anforderung festlegt. Die Anforderungsschablonen bieten einen einfachen Ansatz zur Reduzierung von Spracheffekten bei der Dokumentation der Anforderungen (Pohl & Rupp, 2015a).

Dieses Kapital beschreibt den aktuellen Stand der Wissenschaft zu unterschiedlichen Anforderungsschablonen. Im letzten Kapitel wird ein Vergleich zur Auswahl der Anforderungsschablonen durchgeführt, welche die Grundlage zur Entwicklung der Anwendung in dieser Arbeit darstellt.

3.1 MASTeR Anforderungsschablone

Chris Rupp und Rainer Joppich (Rupp & Joppich, 2014) bieten eine Vielzahl der unterschiedlichen Anforderungsschablonen für die natürlichsprachlichen Anforderungen an, von Anforderungsschablone für die funktionalen Anforderungen bis hin zu Anforderungsschablone für Bedingungssätze. Die gesamten Anforderungsschablonen werden unter dem Begriff MASTeR (Mustergültige Anforderungen- die SOPHIST Templates für Requirerments) Schablone bezeichnet. Die Abbildung 6 zeigt eine vollständige Übersicht über MASTeR Anforderungsschablone.

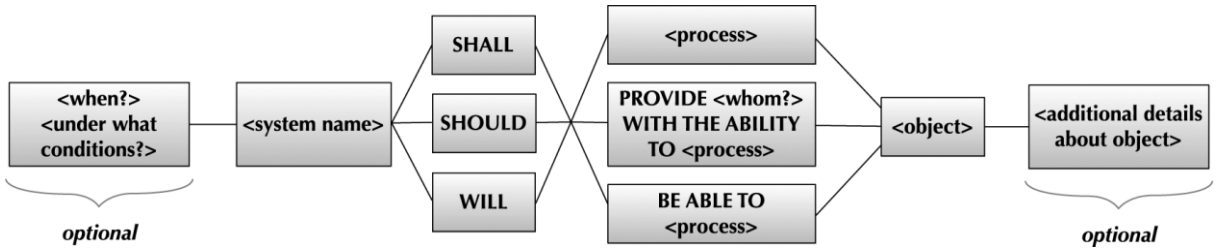


Abbildung 6 MASTeR- Anforderungsschablone (Pohl & Rupp, 2015b)

Das Wort <*process*> steht in der MASTeR Anforderungsschablone im Mittelpunkt, mit dem die geforderten Funktionalitäten der Anforderungen beschrieben werden sollen. Im

Zusammenhang mit dem Prozesswort spielt die Systemaktivität eine wichtige Rolle bei dem Aufbau dieser Anforderungsschablone und wird in drei Arten unterteilt (Pohl & Rupp, 2015a):

- *Selbsttätige Systemaktivität (engl. Autonomous system activity):* Das System führt den Prozess selbstständig durch.
- *Benutzerinteraktion (engl. User interaction):* Das System stellt dem Nutzer eine Funktionalität zur Verfügung, die der Nutzer startet oder es mit ihm in Interaktion steht.
- *Schnittstellenanforderung (engl. Interface requirement):* Das System führt einen Prozess in Abhängigkeit von einem Dritten beispielweise von einem Fremdsystem aus. In diesem Fall ist das System passiv und wartet auf ein externes Ereignis.

Neben Prozesswort und Systemaktivität existieren noch in der MASTeR Anforderungsschablone Modalverben, Objekte und Bedingungen. Für jede Art der Systemaktivität haben Rupp und Joppich ein entsprechendes Beispiel vorgestellt. Im Folgenden wird ein Beispiel zur Verdeutlichung der Schnittstellenanforderung dargelegt (Rupp & Joppich, 2014):

If the library system is in use, the library system shall be able to receive data for a software update from a central administration server.

Die MASTeR Anforderungsschablone hilft den Nutzern, die Aufwände beim Ermitteln und Schreiben von Anforderungen zu minimieren und die Qualität der Anforderungen in einem optimalen Zeit- und Kostenrahmen zu erhöhen (Pohl & Rupp, 2015b). Trotz der Vorteile kann diese Schablone die Mehrdeutigkeit der Worte nicht verhindern. Ein möglicher Grund dafür ist, dass die Projektbeteiligten die Anforderungen aufgrund unterschiedlichen Wissens, Prägung und Erfahrung unterschiedlich formulieren (Pohl & Rupp, 2015a).

3.2 Easy Approach to Requirements Syntax (EARS) Anforderungsschablone

Die EARS Anforderungsschablone wurde entwickelt, um die Probleme der in natürliche Sprache beschriebenen Anforderungsdokumente zu vermindern. Die Probleme sind Mehrdeutigkeit, Unbestimmtheit (Mangel an Präzision, Struktur), Komplexität, Vervielfältigung (Wiederholung der Anforderungen, die gleiche Funktionalität definieren) (Mavin et al., 2009). Diese Anforderungsschablone wird aus vier Bausteinen wie Bedingung, Systemaktivität, Modalverb und Systemverhalten aufgebaut. Die Abbildung 7 zeigt ein Gesamtbild der EARS Anforderungsschablone.

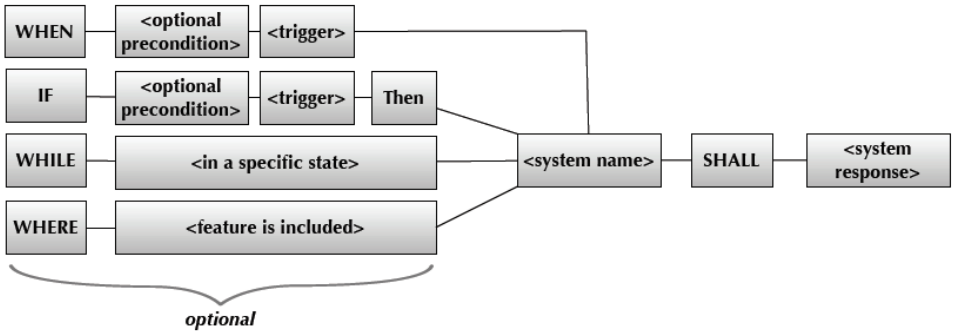


Abbildung 7 Übersicht der EARS Anforderungsschablone (Arora et al., 2014)

Die Bedingung ist optional und abhängig von den sechs folgenden Anforderungstypen (Mavin et al., 2009):

- *Uniquitous requirements* haben keine Vorbedingungen und werden für die Anforderungen verwendet, die immer aktiv sind.
- *Event-driven requirements* werden mit **WHEN** gestartet und initiiert, nur wenn ein Triggerereignis (*engl. trigger event*) an der Systemgrenze erkannt wird.
- *Unwanted behavior requirements* werden mit **IF-THEN** Struktur bezeichnet und können die ungewünschten Situationen abdecken, die beispielweise Ausfälle, Störungen, Abweichungen von gewünschten Benutzerverhalten und unerwartetes Verhalten der Interaktionssysteme sind.
- *State-driven requirements* werden mit Schlüsselwort **WHILE** gestartet und für die Anforderungen verwendet, wenn das System sich in einem definierten Zustand befindet.
- *Optional feature requirements* werden mit Schlüsselwort **WHERE** gekennzeichnet und werden nur für die Systeme verwendet, die eine bestimmte Funktion enthalten.
- *Complex requirements* sind eine Kombination der oben sogenannten Anforderungen.

Mavin et al. (Mavin et al., 2009) haben für jeden Anwendungsfall der Anforderungen ein Beispielsatz vorgestellt, um die Syntax der Schablone deutlicher zu erläutern. Folgend ist ein Beispiel für die komplexe Anforderung, die aus der Kombination der State-Driven Anforderung und Event-Driven Anforderung besteht (Mavin et al., 2009):

„***While*** *the aircraft is on-ground,* ***when*** *reverse thrust is commanded, the control system shall enable deployment of thrust reverser* “

Die EARS Anforderungsschablone bietet eine automatische Unterstützung bei der Ableitung der Anwendungsfallmodelle aus den funktionalen Anforderungen (Majumdar et al., 2011). Mit der Anwendung der Anforderungsschablone können die Probleme wie Komplexität,

Wiederholung von Anforderungen, Unprüfbarkeit oder unangemessene Implementierung aufgelöst werden (Mavin et al., 2009). Trotzdem kann diese Anforderungsschablone die Mehrdeutigkeit, Unbestimmtheit und Worthaftigkeit der Anforderungen nur reduzieren, aber nicht eliminieren (Mavin et al., 2009).

3.3 Anforderungsschablone nach Toro et al.

Toro et al. (Toro et al., 1999) stellt eine Anforderungsschablone vor, die Ingenieure und Benutzern bei der Formulierung der Anforderungen für die Informationssysteme in natürlicher Sprache unterstützen. Diese Anforderungsschablone unterteilt sich in zwei Patterns, *linguistic patterns (L-Patterns)* und *requirement patterns (R-Patterns)*. Während die L-Patterns häufig in den Anforderungsbeschreibungen verwendet werden, kommen R-Patterns in den Anforderungserhebungsprozessen vor. Die L- und R-Patterns werden in der Form einer Tabelle präsentiert und in unterschiedlichen Anwendungsfällen angewendet, die unter drei Kategorien unterteilt wird (Toro et al., 1999):

- *Information Storage Requirements* helfen den Benutzern herauszufinden, welche relevanten Informationen für die Geschäftsziele im System gespeichert werden sollen. Die Informationen können die Daten über Kunden, Produkte, Bestellungen usw. sein. Die Abbildung 8 zeigt ein Beispiel für die Anwendung dieser Anforderungsschablone für ein System zum Ausleihen von Videobändern.

RI–01	**Information about movies**
Version	1.0 (Feb, 17, 1999)
Author	A. Durán (University of Seville)
Source	R. Corchuelo (Super Video Shop)
Purpose	To know availability of movies at any moment and to be able to help customers to select a movie using different criteria
Description	The system shall store the information corresponding to movies in the video store. More precisely:
Specific data	• Title of the movie • Number of tapes of the movie rented at any moment • Number of tapes of the movie ready to rent at any moment • Type of the movie: children, action, science–fiction or adults • Time of the movie, in hours and minutes • Main actors of the movie • Director of the movie • Producer of the movie • Year of production of the movie
Time interval	past and present
Importance	vital
Urgency	immediately
Comments	none

Abbildung 8 Beispiel für die Anwendung der Schablone für ein System zum Ausleihen von Videosbändern (Toro et al., 1999)

- *Functional Requirements* beschreiben die Anwendungsfälle und unterstützen den Benutzer und Kunden, um herauszufinden, wie die gespeicherten Informationen im System angestellt werden sollen, damit die Geschäftszielen erreicht werden. Darüber hinaus umfassen *R-Patterns* noch vier Patterns, die häufig in der Entwicklung der Informationssysteme verwendet und als CRUD R-Patterns (*Create, Read, Update, Delete*) benannt werden.
- *Non-Functional Requirements* unterstützen den Benutzer bei der Anforderungsbeschreibung über Datenschutz, Zuverlässigkeit usw.

Diese Anforderungsschablone und Patterns wurden in mehr als 40 akademischen Praxen angewendet und in zwei Entwicklungen von Sadiel S.A., ein Software-Unternehmen in Spanien, eingesetzt (Toro et al., 1999). Das Resultat zeigt, dass die Schablone bei der Ermittlung der Anforderungen und beim Ausdrücken in der natürlichen Sprache hilft. Dies hat zur Folge, dass die Benutzer und Kunden die Anforderungen besser verstehen können.

3.4 Anforderungsschablone nach Denger et al.

Die häufigen Probleme bei den Anforderungsbeschreibungen in natürlicher Sprache sind inhärente Ungenauigkeit, Mehrdeutigkeit oder Unvollständigkeit (Denger et al., 2003). Um diese Probleme bei der Formulierung der Anforderungen in natürliche Sprache zu reduzieren, haben Denger et al. (Denger et al., 2003) verschiedene natürlichsprachliche Muster für eingebettete Systeme vorgestellt. Zusätzlich haben die Autoren ein Metamodell entwickelt, das als Grundlage für die Entwicklung von Mustern von den Anforderungsbeschreibungen für eingebettete Systeme dient (Denger et al., 2003). Die Arten der Anforderungsangaben können mit der Syntax der Muster beschrieben werden. Die Abbildung 9 zeigt einen Überblick über alle von Denger et al. vorgestellten Muster und die Syntax der Event Patterns wird als ein Beispiel dargelegt.

Übersicht der Muster

Pattern Content	Pattern Name
Functional Requirement Sentence Patterns	SPD, SPC
Event Patterns	EP1, EP2, EP3, EP4, EP5, EP6
Reaction Patterns	CRP, RP1, RP2
Computation Pattern	CCP
Condition Patterns	ABCP, BCP, SCP, TCP1, TCP2, TCP3, TCP4, TCP5, TCP6
Relationships Patterns	PP, OP1, OP2, OP3, OP4, OP5, OP6
Exception Patterns	EXP1, EXP2, ERP1, ERP2, ERP3
Patterns for Special Aspects	RoRP, RoCP, CoRP, ADP
Nonfunctional Requirement Sentence Pattern	NFRP

Abkürzung:

SP : sentence pattern
EP : event pattern
CRP: conditioned reaction pattern
RP : reaction pattern
CCP: continuous computation pattern
ABCP: abstract boolean condition pattern
BCP: boolean condition pattern
SCP: state condition pattern
TCP: time condition pattern

PP: priority pattern
OP: order pattern
EXP: exception pattern
ERP: exception reaction pattern
RoRP: realization of reaction pattern
RoCP: realization of computation pattern
CoRP: completion of reaction pattern
ADP : assumption duration pattern
NFRP: nonfunctional requirements pattern

Event Patterns

EP1: <**When** | **If**> (conjunction)
noun phrase (VARIABLE) *verb* (VALUE CHANGE)
{*numeral adjective* (VARIABLE VALUE) | *noun phrase* (VARIABLE)}

EP3: <**When** | **If**> (conjunction)
noun phrase (ACTOR | RECEIVER) *verb* (ACTION | COMMUNICATION) *noun phrase* (ACTUATOR | OBJECT)

EP5: <**When** | **If**> (conjunction)
noun phrase (ACTOR | VARIABLE | STATE OF)
within *time* (variable of type Time)

Abbildung 9 Übersicht der Muster und Event Patterns (Denger et al., 2003)

Zur Verdeutlichung der Anwendung dieser Anforderungsschablone, wird folgendes Beispiel für Event Patterns *EP5* gewählt (Denger et al., 2003):

***If** the system does not receive the signal 'temp' **within** 4 ms, …*

Die Autoren haben die vorhandenen Anforderungen von einem Kommunikationssystem, Enhanced Voice Mail System (EMS), umgeschrieben, um die vorgestellte Anforderungsschablone zu evaluieren. Das Resultat dieser Umschreibung zeigt, dass die Anforderungen besser erklärt wurden und dadurch die Interpretation bei den Lesern erleichtert. Darüber hinaus konnten die möglichen Ursachen für Mehrdeutigkeit, Unvollständigkeit und Ungenauigkeit beseitigt werden (Denger et al., 2003). Neben den genannten Vorteilen hat diese Anforderungsschablone auch Nachteile. Zum einen sind einige umgeschriebenen Anforderungen länger als die Originale, da die originalen Anforderungen um die nötigen Informationen erweitert werden müssen. Zum anderen ist diese Anforderungsschablone domänenabhängig und spezifisch für eingebetteten Systeme.

3.5 Anforderungsschablone nach Eckhardt et al.

Eckhardt et al. (Eckhardt et al., 2016) stellen die Unvollständigkeit bei der Anforderungsbeschreibungen in den Vordergrund. Das Problem der Unvollständigkeit betrifft sowohl funktionale als auch nicht-funktionale Anforderungen wie beispielweise Performance-Anforderungen (Eckhardt et al., 2016). Eine Performance-Anforderung ist eine Anforderung an einem Systemverhalten wie Geschwindigkeit, Genauigkeit usw. und legt die Bedingungen für eine funktionale Anforderung fest (IEEE Standards Board, 1990). Um die Unvollständigkeit der Performance-Anforderungen zu verhindern, haben die Autoren ein Framework für die Spezifikation dieser Anforderungen definiert. Das Framework besteht aus einem einheitlichen Modell der Performance-Anforderungen, einem Inhaltsmodell**,** einer Definition für Vollständigkeit und einer Operationalisierung durch Satzmuster, die folgend beschrieben werden (Eckhardt et al., 2016):

- Bei dem *einheitlichen Modell* werden die Anforderungen in vier verschiedenen Typen unterschieden, Zeitverhaltensanforderungen (*engl. Time behavior requirements*), Durchsatzanforderungen (*engl. Throughput requirements*), Kapazitätsanforderungen *(engl. Capacity requirements*) und Querschnittsanforderungen (*engl. Cross-cutting requirements)*. Folgende Abbildung 10 zeigt einen Überblick über die Zuordnung der Anforderungen und die zugehörigen Performance-Aspekte.

Performance			
Time Behavior	**Throughput**	**Capacity**	**Cross-cutting**
- Points in time - Response time - Reaction time - Turnaround time - Time intervals - Latency - Time constraints	- Rate of transactions - Data volume per unit of time - Reaction speed - Processing speed - Operating speed	- Maximum limits - Concurrent users - Communication bandwidth - Size of database or storage	- Measurement location - Measurement period - Load - Platform - Scope of measurement - Measurement assumption

Abbildung 10 Einheitliches Modell von Performance-Anforderungen mit zugehörigen Aspekten (Eckhardt et al., 2016)

- Das *Inhaltsmodell* (siehe Abbildung 11) erfasst die relevanten Inhaltselemente, die sich auf unterschiedliche Performance-Aspekte vom einheitlichen Modell der Performance-Anforderungen beziehen. Während Part 1 in Abbildung 11 die allgemeinen Inhaltselemente wie Modularität, Umfang usw. bezeichnet, stehen die Inhaltselemente im Part 2 im Zusammenhang mit drei Typen (Zeitverhalten, Durchsatz, Kapazität) von Performance-Anforderungen. Part 3 zeigt die Inhaltselemente für Querschnittsaspekte.

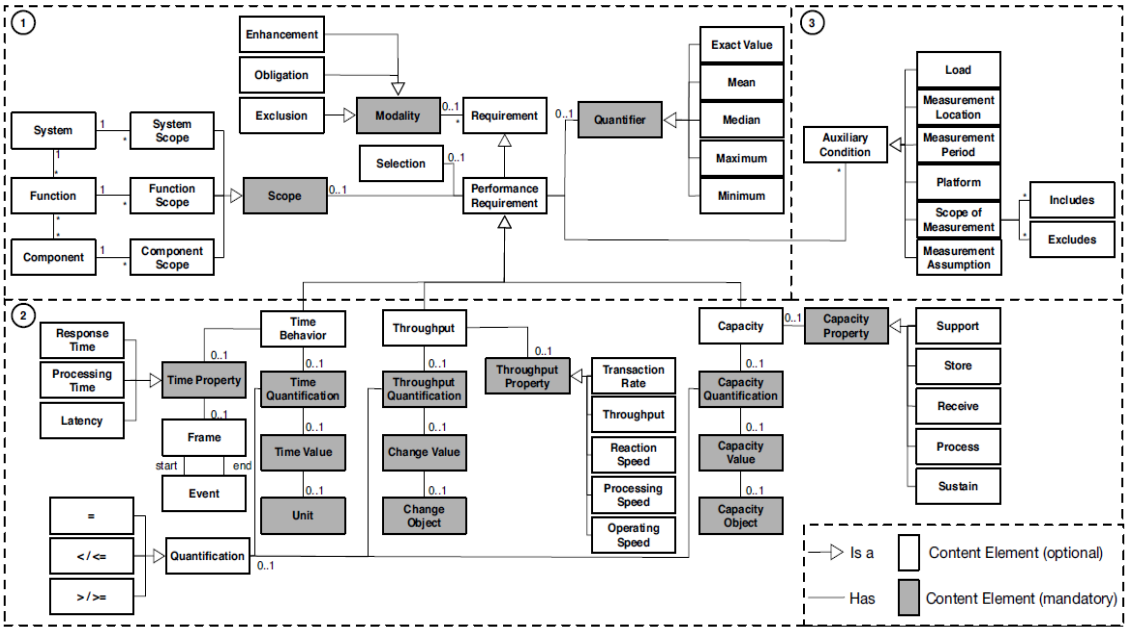
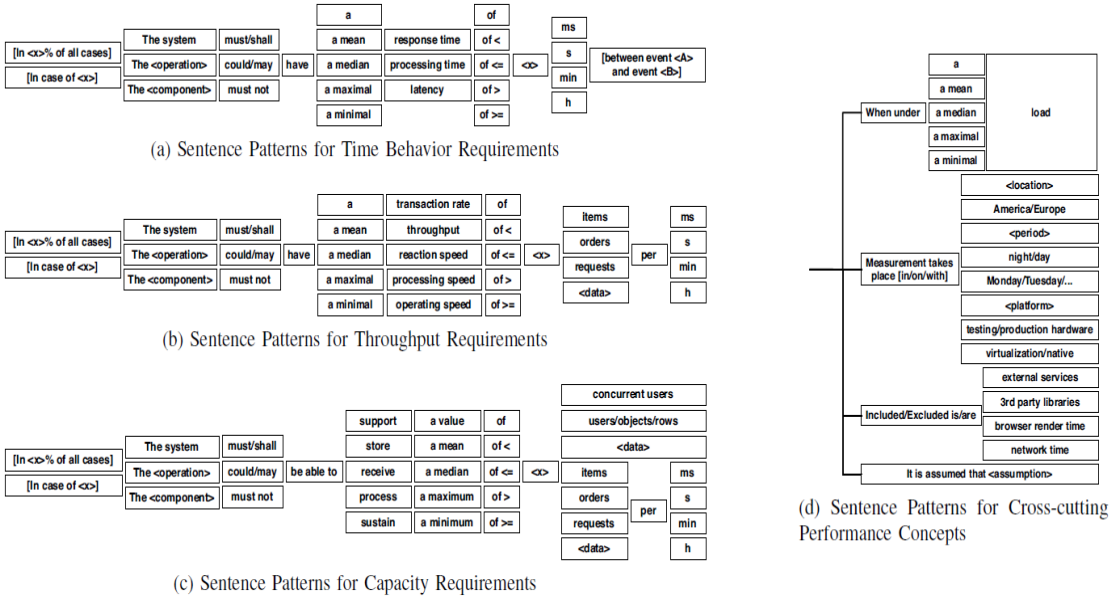


Abbildung 11 Inhaltsmodell von Performance-Anforderungen (Eckhardt et al., 2016)

- *Eine Definition der Vollständigkeit* für die Performance-Anforderungen wurde basierend auf den Entwicklungsaktivitäten erstellt. Diese Definition beschreibt, welche Inhaltelemente für die Beschreibung einer Performance-Anforderung nötig sind, um vollständig bezeichnet zu werden. Diese Definition hilft bei der Erkennung und Prüfung nach der Unvollständigkeit der Anforderungen.
- Basierend auf das Inhaltsmodell und den vier Typen von Performance-Anforderungen werden die *Satzmuster* abgeleitet. Die Abbildung 12 veranschaulicht den Aufbau der Satzmuster für jeden Typ der Performance-Anforderungen, wobei die Syntax und Schlüsselworte der Satzmuster dargestellt werden.



(a) Sentence Patterns for Time Behavior Requirements

(b) Sentence Patterns for Throughput Requirements

(c) Sentence Patterns for Capacity Requirements

(d) Sentence Patterns for Cross-cutting Performance Concepts

Abbildung 12 Übersicht der Satzmuster für die Performance-Anforderungen (Eckhardt et al., 2016)

Folgend ist ein Beispiel für die Kapazitätsanforderung anhand der vorgestellten Satzmuster (Eckhardt et al., 2016):

The system must be able to process a maximum of 10 000 requests per s.

Für die Evaluierung dieses Frameworks haben die Autoren insgesamt 530 nicht-funktionalen Anforderungen aus 5 Unternehmen für unterschiedliche Applikationsdomänen gesammelt. Anhand der Qualitätskriterien an jede einzelne Anforderung nach ISO/IEC 9126 wurden die gesamten Anforderungen auf 58 Anforderungen aussortiert. Aufgrund der fehlenden und ungenaueren Informationen wurden schließlich nur 50 Anforderungen mit dem vorgestellten Framework angewendet. Das Ergebnis nach der Evaluierung zeigt, dass man die Unvollständigkeit der Anforderungen erkennen und mit den Satzmustern des Frameworks umsetzen kann. Der Nachteil des Frameworks ist, dass die Fokussierung auf die Unvollständigkeit der Anforderungen liegt, obwohl eine Anforderung noch weitere Qualitätsmerkmale, die in Abschnitt 2.1.1 vorgestellt wurden, erfüllen muss.

3.6 Mazo & Jaramillo Anforderungsschablone

Mazo et al. (Mazo et al., 2020) argumentieren, dass die im Kapital 3.1 vorgestellte MASTeR Anforderungsschablone Probleme mit Mehrdeutigkeit und Inkonsistenz bei der Anforderungsbeschreibungen hat. Aus diesen Gründen und basierend auf die MASTeR-Anforderungsschablone haben die Autoren eine erweiterte Anforderungsschablone entwickelt, um die Formulierung der funktionalen und nicht-funktionalen Anforderungen in unterschiedlichen Domänen wie Anwendungssysteme, Softwareproduktlinien, Cyber-Physical Systeme zu unterstützen (Mazo et al., 2020). Außerdem wurden die im Abschnitt 3.2 beschriebene EARS Schablone und der Standard ISO/IEC/IEEE 29148:2011(ISO/IEC/IEEE 29148: 2011, 2011) auch in dem ersten Zyklus der Entwicklungsphase berücksichtigt (Mazo et al., 2020). Die Abbildung 13 zeigt eine Übersicht über die Mazo & Jaramillo Anforderungsschablone.

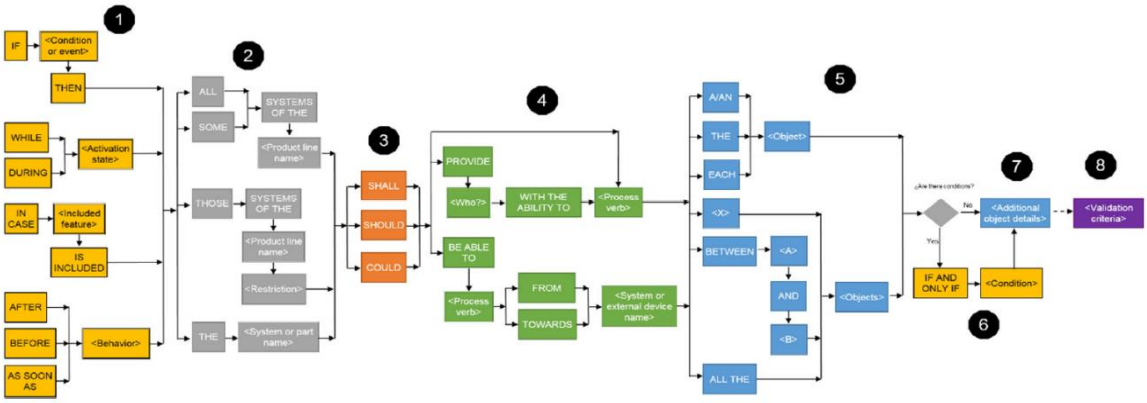


Abbildung 13 Mazo & Jaramillo Anforderungsschablone (Mazo et al., 2020)

Jede Farbe der Anforderungsschablone hat eine unterschiedliche Bedeutung (Mazo et al., 2020):

- Die **gelbe** Farbe präsentiert die Bedingungen.
- Die **graue** Farbe wird benutzt, um die Gruppe von Systemen, das System oder ein Teil des Systems darzustellen.
- Die **orange** Farbe präsentiert die Modalverben.
- Die **grüne** Farbe ist für die Aktivitäten, die das System charakterisieren.
- Die **blaue** Farbe stellt die Objekte mit den entsprechenden Mengen und Ergänzung dar.
- Die **violette** Farbe beschreibt die messbaren Kriterien zur Anforderungsüberprüfung.

Folgend ist ein Beispiel für diese Anforderungsschablone, um die Syntax bei der Formulierung der Anforderung zu veranschaulichen (Mazo et al., 2020).

If a motion sensor is activated, then the Oktupus system should send an instant image to the home owner's email

Die Mazo & Jaramillo Anforderungsschablone wurde durch zwei aktive Forschungszyklen aufgebaut und evaluiert. Außerdem wurden unterschiedliche Phasen in jedem Zyklus durchgeführt, um die Qualität der vorgestellten Anforderungsschablone zu sichern. Zwei Gruppen aus den Business Analysts und technischen Anforderungsprüfern wurden für jeden Forschungszyklus gegründet, um die Umsetzung dieser Anforderungsschablone auf die 46 Anforderungen an dem PeopleQA System von SQA S.A Unternehmen zu überprüfen (Mazo et al., 2020). Die Probleme wie Unvollständigkeit, mangelnde Überprüfbarkeit bei nicht-funktionalen Anforderungen werden durch Anwendung dieser Anforderungsschablone behoben. Darüber hinaus können Lücken wie Mehrdeutigkeit der MASTeR-Anforderungsschablone abgedeckt werden. Trotzdem wurde diese Anforderungsschablone nur bei wenigen industriellen Fällen getestet.

3.7 Vergleich der Anforderungsschablonen

Zur Bewertung der vorgestellten Anforderungsschablonen werden die im Abschnitt 2.1.1 beschriebenen Kriterien an die Qualität der Anforderungen und Fokus auf die funktionalen und/oder nicht-funktionalen Anforderungen verwendet. Die Tabelle 2 gibt eine Übersicht über die Bewertung der Anforderungsschablonen. Punkt (+) bedeutet, dass das Kriterium zur Bewertung der Anforderungsschablone berücksichtigt wird. Die Anforderung hat kein (+), wenn dieses Kriterium nicht betrachtet wird.

Anforderungsschablone		MASTeR	EARS	nach Toro et al.	nach Denger et al.	nach Eckhardt et al.	Mazo & Jaramillo
Qualitätskriterien	Eindeutigkeit	+	+		+		+
	Notwendigkeit						
	Realisierbarkeit						
	Atomarität						
	Vollständigkeit	+	+		+	+	+
	Verfolgbarkeit						
	Konsistenz						+
	Prüfbarkeit	+	+				+
Fokus auf	Funktionale Anforderungen	+	+	+	+	+	+
	Nicht-funktionale Anforderungen	+		+	+	+	+

Tabelle 2 Übersicht über die Bewertung der Schablone (in Anlehnung (Rupp & Joppich, 2014), (Mavin et al., 2009), (Toro et al., 1999), (Eckhardt et al., 2016), (Denger et al., 2003), (Mazo et al., 2020)) aus vorherigen Abschnitten

Aus der Bewertung der vorgestellten Anforderungsschablonen unterstützt die Mazo & Jaramillo Anforderungsschablone die meisten Qualitätskriterien und die Formulierung der funktionalen und nicht-funktionalen Anforderungen. Darüber hinaus wurde die Mazo & Jaramillo Anforderungsschablone basierend auf den MASTeR, EARS Anforderungsschablonen und den Standard ISO/IEC/IEEE 29148:2011 entwickelt, die häufig in der Praxis für die Anforderungsdokumente der Softwareprojekten angewendet wurden. Aus diesen Gründen wird die Mazo & Jaramillo Anforderungsschablone in der geplanten Anwendung implementiert.

4 Anforderungen an die Arbeit

Im Folgenden wird auf die Anforderungen dieser Arbeit eingegangen. Im Kapital 4.1 wird der Zweck der Arbeit veranschaulicht. Als nächstes erfolgt im Kapital 4.2 die Anforderungsbeschreibungen für die Arbeit.

4.1 Zweck der Arbeit

Das Ziel dieser Arbeit ist es, die Anforderungsbeschreibungen von Softwaresystemen mit Fokus auf einer Anforderungsschablone zu überprüfen. Dies ist nötig, um eine Anwendung zur automatischen Überprüfung der Anforderungsbeschreibungen durch die Anwendung von Natural Language Processing und Anforderungsschablone zu implementieren.

4.2 Anforderungen

Jede Anforderung hat unterschiedlich rechtliche Verbindlichkeit, die durch ein Schlüsselwort bezeichnet wird. Christine Rupp und Rainer Joppich (Rupp & Joppich, 2014) haben drei Schlüsselwörter *muss, sollte* und *wird* definiert, die in Tabelle 3 beschrieben werden. Anhand dieser Schlüsselwörter werden die Anforderungen für diese Arbeit formuliert.

Schlüsselwörter	Semantische Definition
muss	Diese Anforderung ist verpflichtend und muss erfüllt werden, damit die Funktionalitäten der Applikation erfüllt werden.
sollte	Die Anforderung ist wichtig, aber nicht verpflichtend. Es ist keine notwendige Bedingung für die Erfüllung der Funktionalitäten der Applikation.
wird	Die Anforderung ist nicht notwendig, aber gewünscht. Diese Anforderung hilft in der aktuellen Lösung, mit den künftigen Anforderungen optimal zu integrieren.

Tabelle 3 Definition der Schlüsselwörter (Rupp & Joppich, 2014)

Die Anforderungen für diese Arbeit werden in zwei Kategorien, funktionale und nicht-funktionale Anforderungen, unterteilt.

4.2.1 Funktionale Anforderungen

REQ-01: Die Anwendung muss dem Benutzer die Möglichkeit bieten, die Anforderungen zu beschreiben und zu überprüfen.

> **REQ-01.1**: Die Anwendung muss ein Textfeld beinhalten, um die Anforderungsbeschreibungen einzugeben.
>
> **REQ-01.2**: Die Anwendung muss ein Button beinhalten, um die Anfragen zu senden und die Antworte von Server zu empfangen.

REQ-02: Die Anwendung muss auf Google Cloud Plattform[8] bereitgestellt werden.

REQ-03: Die Anwendung muss dem Benutzer die Möglichkeit bieten, die konformen und nicht konformen Anforderungen zu zeigen.

> **REQ-03.1**: Die Anwendung sollte die Gründe für die nicht konformen Anforderungen zeigen.

4.2.2 Nicht-funktionale Anforderungen

ISO/IEC 25010:2011 (ISO/IEC 25010:2011, 2011) hat das Produktqualitätsmodell definiert, um die Qualität eines Produkts zu sichern. Das Produktqualitätsmodell unterteilt die Produktqualitätseigenschaften in insgesamt acht Qualitätsmerkmale, die funktionale Eignung, Zuverlässigkeit, Effizienz, Benutzbarkeit, Sicherheit, Kompatibilität, Wartbarkeit und Portabilität sind (ISO/IEC 25010:2011, 2011). Um die Anforderungen an der Qualität der zu entwickelnden Anwendung zu bestimmen, werden die sogenannten Qualitätsmerkmale betrachtet. Die Tabelle 4 beschreibt die Bedeutung der für die nicht-funktionalen Anforderungen dieser Arbeit benötigten Qualitätsmerkmale.

[8] https://cloud.google.com/

Charakteristiken	Sub-Charakteristiken	Definition
Portability		Grad, Charakteristiken mit dem die Software effizient in eine Umgebung auf eine andere übertragen werden kann
Maintainability	Modularity	Grad der Modularität einer Software, sodass die Veränderung einzelner Module auf die anderen Module minimal auswirkt.
	Modifiability	Grad, in dem die Software effizient angepasst oder erweitert werden kann
	Reusability	Grad, in dem Komponenten der Software in mehreren Komponenten verwendet werden können.
	Testability	Grad der Wirksamkeit und Effizienz, mit dem Testkriterien für eine Komponente der Software festgelegt und Tests durchgeführt werden können, um festzustellen, ob diese Kriterien erfüllt wurden.
Performance efficiency	Time Behaviour	Grad, in dem die Antwortzeiten der Software den Anforderungen entspricht.

Tabelle 4 Charakteristiken und Sub-Charakteristiken für Softwarequalität (ISO/IEC 25010:2011, 2011)

– **Portability**

REQ-05: Die Anwendung sollte gestaltet sein, dass die in verschiedenen Browsern wie Google Chrome, Firefox usw. ausführbar sein.

– **Maintainability**

- **Reusability**

REQ-06: Die Anwendung sollte gestaltet sein, dass gemeinsam nutzbarer Code von den existierten Komponenten bei neuen Komponenten wiederverwendet werden kann und nicht mehrfach definiert wird.

- **Modifiability**

REQ-07: Die Anwendung muss gestaltet sein, dass sie einfach erweitert werden kann.

- **Modularity**

REQ-08: Die Anwendung muss gestaltet sein, dass sie vom bestehenden QDAcity[9] entkoppelt bzw. verbunden werden kann.

- **Testability**

REQ-09: Alle Methode der Anwendung müssen mit Unit-Tests abgedeckt werden.

REQ-10: Die gesamte Testabdeckung der Anwendung sollte bei mindestens 75% liegen.

– **Performance**

- **Time Behaviour**

REQ-11: Die Anwendung muss gestaltet sein, dass die Ausführungszeit eines Diensts unter eine Minute begrenzen muss.

[99] https://qdacity.com/

5 Architektur und Design

Die Architektur dieser Arbeit wird in diesem Kapitel beschrieben. Zuerst wird im Kapitel 5.1 auf die Schichten und ihre jeweiligen einzelnen Komponenten eingegangen. Anschließend wird im Kapitel 5.2 die dynamische Sicht der Funktionen der Anwendung beschrieben.

5.1 Schichten

Ausgehend von den im Kapitel 4 beschriebenen Anforderungen *REQ-07* und *REQ-08* soll die Anwendung mit der bestehenden Anwendung QDAcity[10] verbunden bzw. entkoppelt werden und erweiterbar sein. Aus diesen Gründen, eignet sich die Microservice Architektur als grundlegender Architekturstil dieser Anwendung, da die Dienste entkoppelt strukturiert werden (microservices.io, n.d.). Jeder Dienst ist für eine konkrete Verantwortlichkeit zuständig und technologisch von anderen Diensten unabhängig, damit können die Anforderungen *REQ-07* und *REQ-08* erfüllt werden.

Es gibt viele Microservice Technologien wie Azure Service Fabric[11], Lagom[12], MicroProfile[13], Spring Boot[14]. Laut Larrucea et al. (Larrucea et al., 2021) bietet Azure Service Fabric mehr Vorteile als anderen Technologie. Zum Beispiel unterstützt diese Technologie C#, .NET und Java Programmiersprache, während andere Technologien nur Java und Scala (Larrucea et al., 2021). Außerdem wird Azure Service Fabric häufig in industriellen Projekten verwendet. Nachteil dieser Technologie ist, dass diese kostenpflichtig ist. Aus diesem Grund wird Spring Boot ausgewählt, da diese den zweiten Platz bei der Analyse erhalten hat und kostenlos ist. Die vorliegende Arbeit verwendet die Microservice Architektur basierend auf der Spring Boot Architektur.

Spring Boot bietet einen effektiven Weg zur Entwicklung eines Microservices und hat eine Schichtenarchitektur, in der die Schichten miteinander kommunizieren. Es gibt insgesamt vier Schichten in Spring Boot, die *Presentation Layer, Business Layer, Persistence Layer* und *Database Layer* sind (Rakshith & Swamy, 2020). Die Abbildung 14 stellt eine Übersicht über

[10] https://qdacity.com/
[11] https://azure.microsoft.com/de-de/services/service-fabric/
[12] https://www.lagomframework.com/
[13] https://microprofile.io/
[14] https://spring.io/microservices

die Architektur der Anwendung basierend auf der Spring Boot Architektur dar. Hierbei wird das System in zwei Schichten Frontend und Backend unterteilt.

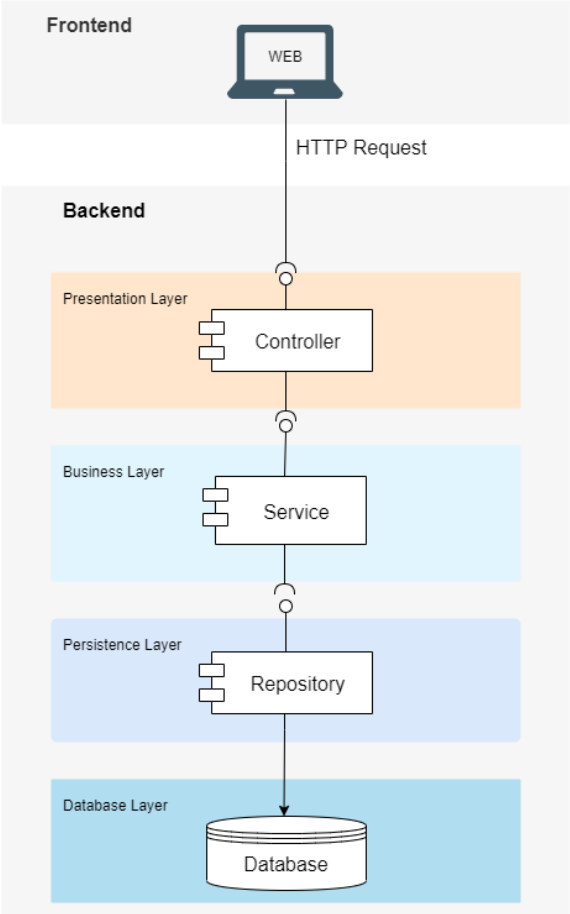


Abbildung 14 Übersicht über Architektur der Anwendung

In den nächsten Kapiteln wird genauer auf die einzelnen Schichten eingegangen.

5.1.1 Frontend

Das Frontend beschreibt die Weboberfläche, bei der einerseits die Funktionalitäten der Anwendung visuell dargestellt und anderseits die Backend Services gesteuert werden. Die Frontend Schicht bietet dem Benutzer die Möglichkeiten, die Anforderungsbeschreibungen einzugeben und als Ergebnis der Überprüfung (konformen bzw. nicht-konformen Anforderungen) anzuzeigen. Darüber hinaus können die Benutzer neue Regeln für die Anforderungsschablone hinzufügen.

5.1.2 Backend

Das Backend wird in vier Schichten unterteilt und wird wie folgt beschrieben:

- *Presentation Layer* beinhaltet die *Controller*-Klasse mit RESTful-API. Einerseits werden die HTTP-Anfragen aus der *Frontend* Schicht in dieser Schicht empfangen und an der *Business Layer* weitergeleitet. Anderseits werden die Ergebnisse nach Verarbeitung aus der *Business Layer* an die *Frontend* Schicht weitergeleitet und dort angezeigt.
- *Business Layer* beinhaltet die *Service*-Klassen und repräsentiert alle Geschäftslogiken dieser Anwendung. Die *Service*-Klassen dieser Schicht integrieren mit der *Persistence Layer* und *Presentation Layer.*
- *Persistence Layer* ist für das Lesen und Speichern der Daten in der Datenbank zuständig. Darüber hinaus umfasst diese Schicht die Entitäten, auf die im Kapitel 5.1.3 eingegangen wird.
- *Database Layer* beschreibt die Datenbanken, der die CRUD Operationen ausgeführt werden. Die Regeln für Anforderungsschablone werden hier gespeichert und für die Überprüfungsaufgabe der Anforderungsbeschreibungen abgefragt.

5.1.3 Entitäten

Die im Kapitel 3.6 beschriebene Mazo & Jamamillo Anforderungsschablone wird in verschiedenen Farben aufgeteilt, dabei stellt jede Farbe eine eigene Bedeutung dar. Um die Überprüfung der Anforderungsbeschreibungen zu vereinfachen und für das Hinzufügen der neuen Regeln zu ermöglichen, wird der reguläre Ausdruck für jede Farbe auf Entitäten konvertiert. Somit können die regulären Ausdrücke als unterschiedlichen Tabellen in der relationalen Datenbank gespeichert werden. Die Abbildung 15 zeigt eine Übersicht über entsprechenden Tabellen mit den zugehörigen Attributen, wobei *key_name* für Schlüsselwort steht und *regex* für regulären Ausdruck des entsprechenden Teils.

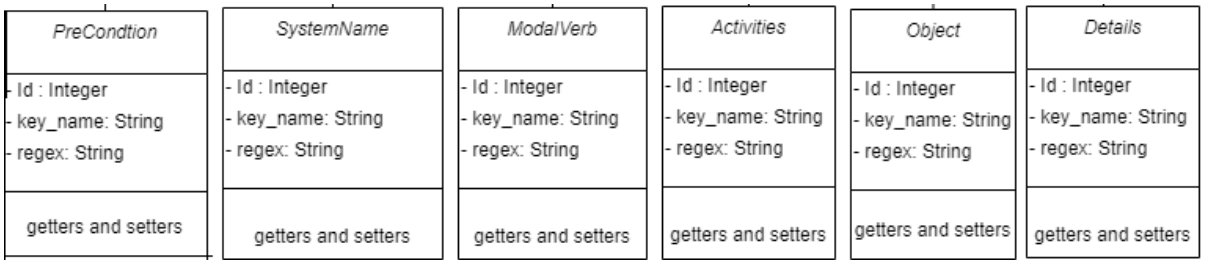


Abbildung 15 Übersicht über die Entitäten

5.1.4 Kommunikation zwischen Frontend und Backend

Die Kommunikation zwischen Weboberfläche und Microservice wird durch REST (Representational State Transfer) ermöglicht. REST ist eine Schnittstelle zur Unterstützung der Client-Server Kommunikation in Netzwerken. Diese Schnittstelle basiert auf der Uniform Ressource Identifiers (URI) für Ressourcenerkennung und auf Hypertext Transfer Protocol (HTTP) für Nachrichtübertragung (Neumann et al., 2018). REST verwendet die Standard-HTTP Anfragenmethode, um Ressourcen vom Server anzufordern (*GET*-Methode) oder die neuen Ressourcen an Server zu schreiben (*POST*-Methode) sowie die bestehenden Ressourcen zu ändern (*PUT*-Methode) oder löschen (*DELETE*-Methode).

5.2 Dynamische Sicht

Im Kapitel 5.1 wurden die statischen Aspekte der Anwendung berücksichtigt. In diesem Kapitel wird die dynamische Sicht der Anwendung beschrieben. Hierbei werden die Vorgänge des Einfügens der neuen Regeln und der Überprüfung der Anforderungsbeschreibungen betrachtet. Das UML-Sequenzdiagramm wird verwendet, um die Interaktionen zwischen Komponenten darzustellen und die Vorgänge zu veranschaulichen.

5.2.1 Einfügen der neuen Regeln

Die Abbildung 16 gibt eine Übersicht über den Ablauf des Einfügens der neuen Regeln für die Anforderungsschablone.

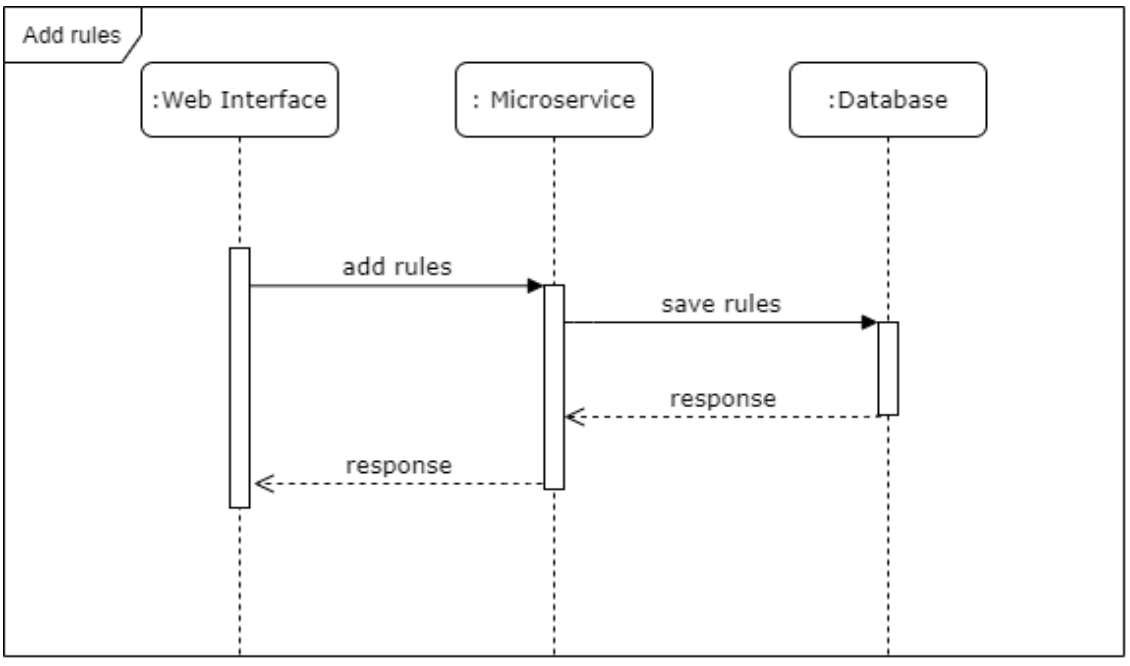


Abbildung 16 Sequenzdiagramm für Hinzufügen der neuen Regeln für die Anforderungsschablone

Nachdem der Nutzer auf dem Button „*Add Rules*“ klickt, wird ein Fenster angezeigt, in dem der Nutzer die neuen Regeln in der Form der regulären Ausdrücke für die Anforderungsschablone eingeben kann. Daraufhin wird eine Anfrage an das Microservice gesendet, um die eingegebenen Daten in die Datenbank zu speichern.

5.2.2 Überprüfung der Anforderungsbeschreibungen

Um die konformen und nicht-konformen Anforderungen zu identifizieren, müssen diese zunächst geprüft werden. Einen Überblick über den Ablauf der Überprüfung zeigt das Sequenzdiagramm in Abbildung 17.

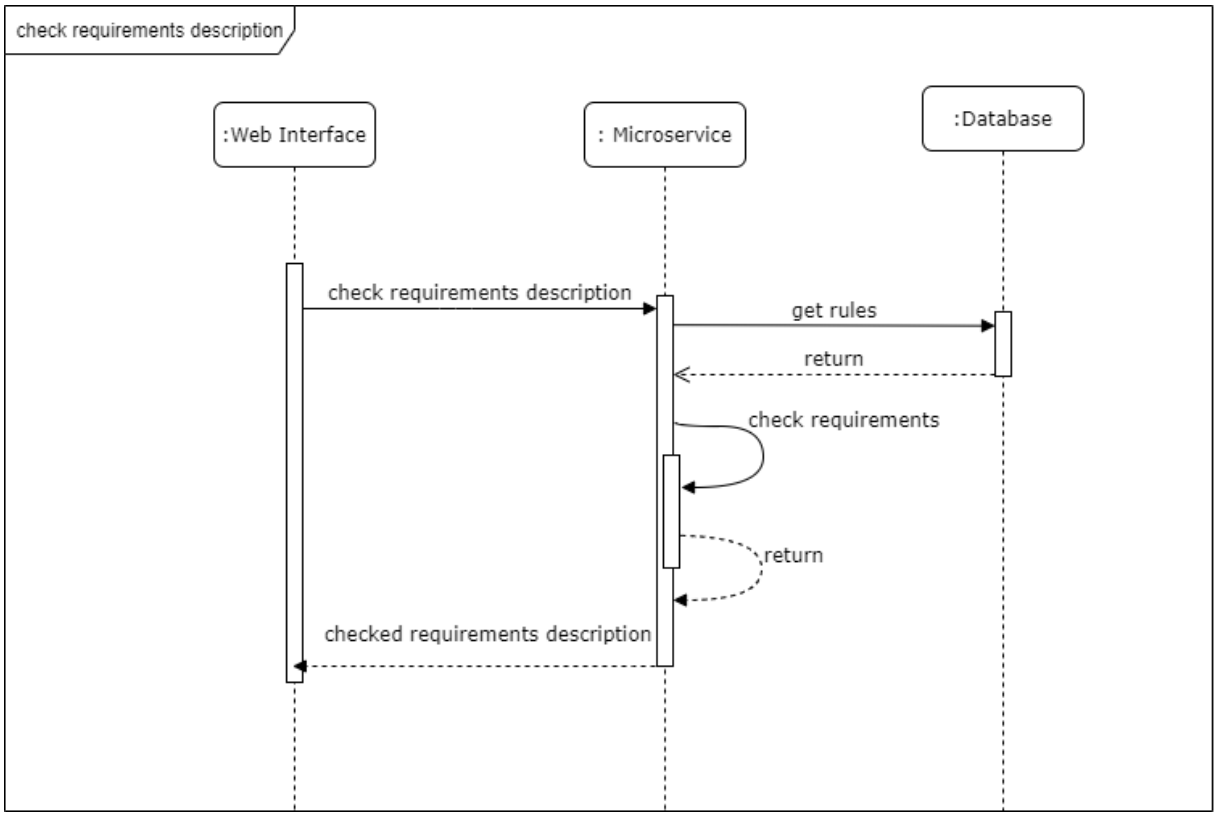


Abbildung 17 Sequenzdiagramm für Überprüfung der Anforderungsbeschreibungen

Die Weboberfläche bietet dem Nutzer die Möglichkeit, die Anforderungsbeschreibungen einzugeben. Nachdem der Microservice die Anfrage von der Weboberfläche bekommt, wird eine Anfrage an die Datenbank gesendet, um die aktuellen Regeln der Anforderungsschablone abzufragen. Nach Erhalt der Regeln wird das Microservice die wesentlichen Funktionen für die Überprüfung ausführen. Schließlich wird eine Antwort mit konformen und nicht-konformen Anforderungen an der Weboberfläche zurückgesendet und dort visualisiert.

Basierend auf diese Architektur wird die Weboberfläche und das Microservice der Anwendung implementiert. In dem nächsten Kapitel wird die Implementierung der Anwendung detaillierter betrachtet.

6 Implementierung

In diesem Kapitel wird die Implementierung der aus dem Kapitel 5 beschriebenen Softwarearchitektur dargestellt. Zunächst wird in den Kapiteln 6.1 und 6.2 auf die Implementierung der Frontend und Backend Schichten eingegangen. Das Kapitel 6.3 gibt eine Gesamtübersicht zur Kommunikation zwischen Frontend und Backend. Schließlich wird die Vorgehensweise beim Testen in dem Kapitel 6.4 vorgestellt.

6.1 Frontend

Um die Weboberfläche für die Anwendung zu implementieren, werden verschiedene webbasierte Open-Source-Frameworks wie Angular[15], React.js[16], Vue.js[17] betrachtet (siehe Abbildung 18).

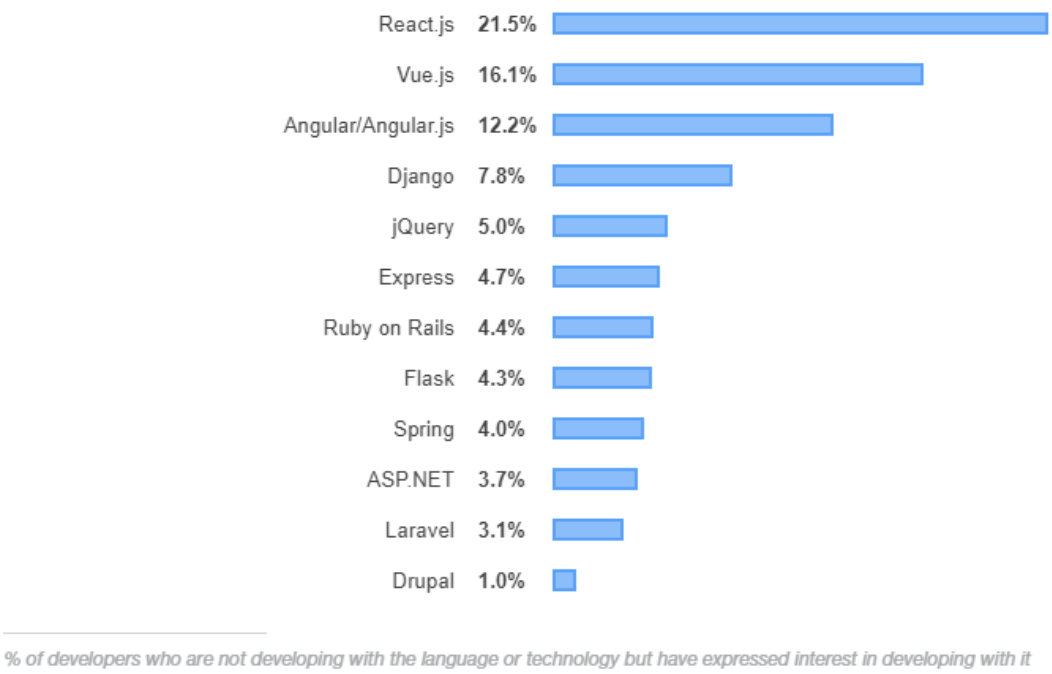


Abbildung 18 Statistik über Verwendung der Frontend Frameworks im 2019 (Existek.com, n.d.)

Angular ist komplex und schwierig beim Erlernen, da die Entwickler das Bündeln wie Komponenten, Modulen, Dependency Injection usw. lernen müssen (Existek.com, n.d.). Das Vue.js Framework ist neu und daher fehlt die Praxistauglichkeit im Vergleich zu Angular und

[15] https://angular.io/
[16] https://reactjs.org/
[17] https://vuejs.org/

React.js. React.js hat ein virtuelles DOM (*Domain On Memory*), die eine abstrakte Kopie vom realen Speicher ist und hilft, bei jeder Aktualisierung in der Weboberfläche schnell zu laufen (Reactjs.org, n.d.). Bei der Anforderung *REQ-08* muss dieses Microservice mit der QDAcity-Website integriert werden, die mit React.js implementiert wurde. Aus diesem Grund fällt die Auswahl des Frameworks zur Entwicklung der Weboberfläche auf React.js.

Die Abbildung 19 zeigt einen Überblick über die Komponenten der Frontend Schicht, die mit React.js implementiert wurden.

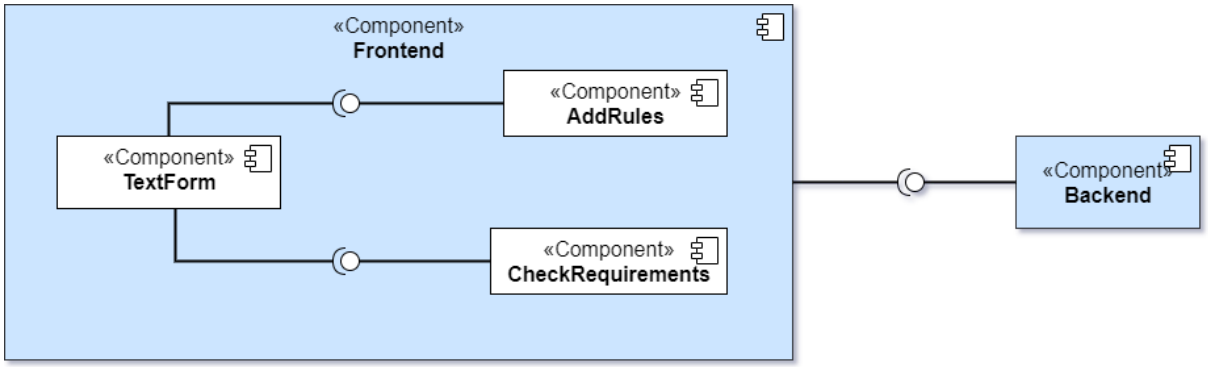


Abbildung 19 Komponentendiagram der Frontend Schicht

Die Startseite der Weboberfläche wird durch die Komponente *TextForm.js* repräsentiert, welche die Komponente *AddRules* und *CheckRequirements* durch die Variablen *popUp* and *addRules* steuert.

Nach Eingabe der Anforderungen und dem anschließenden Klick auf den *Check*-Button wird die Methode *checkText()* aufgerufen, die in Quelltext 1 veranschaulicht wird. Zunächst wird die Eingabe überprüft (Codezeile 3), ob die Nutzer die Anforderungen eingegeben haben. Falls keine Anforderung vorhanden ist, wird eine Fehlermeldung angezeigt (Codezeile 4), dass die Nutzer das Textfeld befüllen sollen. Falls die Eingabe nicht leer ist, wird der Anfrageinhalt in die Variable *desc* gekapselt (Codezeile 7) und an dem Backend über eine Anfrage gesendet (Codezeile 8). Nachdem das Backend die Antwort zurückgesendet hat, werden die Daten der Antwort überprüft. Wenn der Inhalt null ist, wird ein Log auf dem Console mit dem entsprechenden Status angezeigt (Codezeile 45). Im anderen Fall werden die Daten der Antwort bearbeitet, um die Oberfläche für *CheckRequirements* Komponente vorzubereiten (Codezeile 11-38). Schließlich wird die Variable *popUp* auf *true* umgesetzt, um die *CheckRequirements* Komponente anzuzeigen (Codezeile 43).

```
checkText=()=>{
 // check input
    if(this.state.description === null || this.state.description === "") {
            this.showAlert(true);
            return;
    }
  const desc = {description: this.state.description}
  axios.put("http://localhost:8080/description/check", desc)
 .then(response => {
     if(response.data != null) {
       var resullt = response.data[0];
       var conform_list = response.data[1];
       var logs_list = response.data[2];
       var tmp = '';
     Object.keys(resullt).forEach(function(key) {
       // conform
       if(conform_list[key] === '0'){
         tmp = [tmp, <Card style={{ backgroundColor: '#90ee90' }}>{resullt[key]}</Card>];
        }
        // not conform
        else if(conform_list[key] === '1'){
          tmp = [tmp,
               <Accordion  defaultActiveKey="0" >
                     <Accordion.Toggle as={Card.Header}
                           variant="link" eventKey="1"
                            style={{ backgroundColor: ' #ff5050' }}>
                       {resullt[key]}
                     </Accordion.Toggle>
                     <Accordion.Collapse eventKey="1"
                                         style={{ backgroundColor: '#ffb2b2' }}>
                          <Card.Body>
                               <label>Logs:</label>
                               <div className="logs">
                                         {logs_list[key]}
                               </div>
                          </Card.Body>
                      </Accordion.Collapse>
                  </Accordion>
                 ];
             }
       });
         this.setResult(tmp);
         this.setPopUp(true);
    } else {
        console.log(response.status)
    }
 });
}
```

Quelltext 1 Methode checkText() in TextForm-Klasse

Die Abbildung 20 ist ein Beispiel für die Anzeige der Resultate nach Überprüfung der Anforderungen.

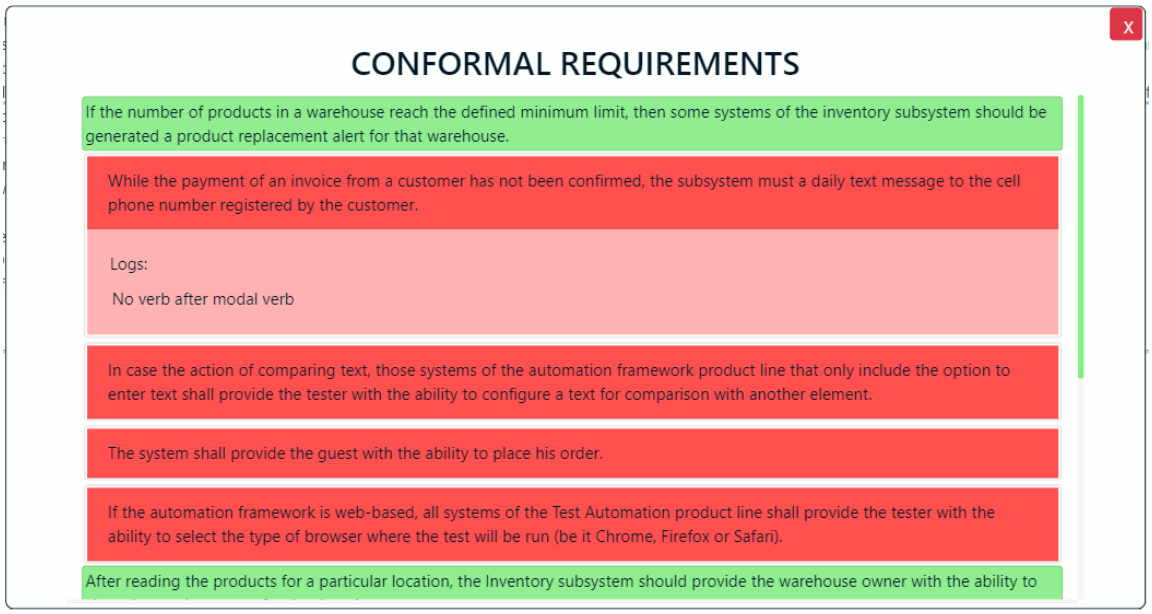


Abbildung 20 Conformal requirements-Fenster

Hierbei beschreibt die grüne Farbe die konformen Anforderungen und die rote Farbe die nicht-konformen Anforderungen. Darüber hinaus kann der Nutzer die Gründe für die nicht-konformen Anforderungen sehen.

Der Klick auf den *Add Rules*- Button ermöglicht dem Nutzer, neue Regeln für die Anforderungsschablone hinzufügen. Der Wert der *addRules* Variable wird auf *true* nach dem Klick auf *Add Rules*-Button gesetzt, um die Oberfläche der *AddRules*- Klasse anzuzeigen (siehe Abbildung 21).

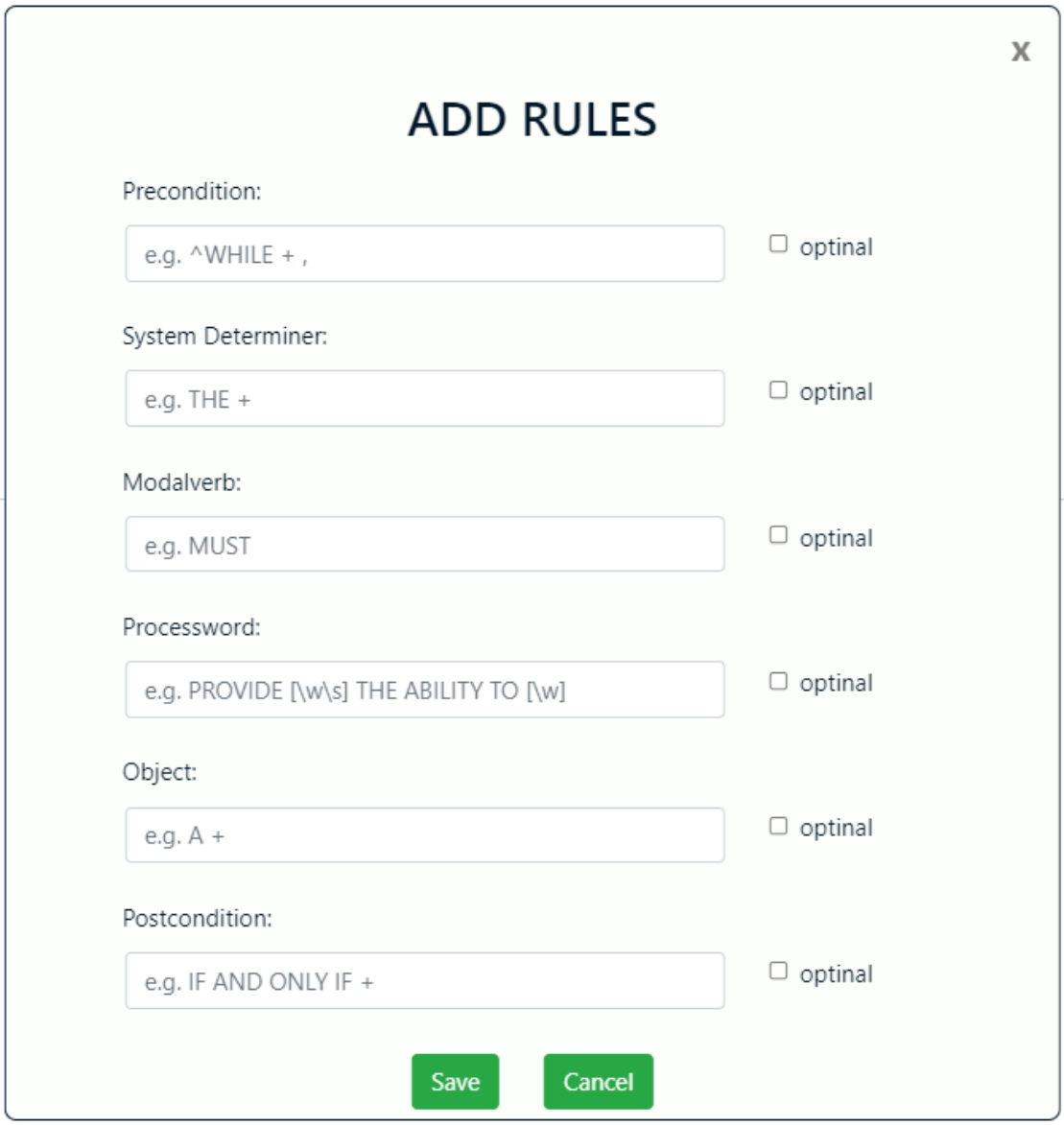


Abbildung 21 Add Rules -Fenster

Nachdem der Nutzer seine Eingaben auf den gewünschten Eingabefelder getätigt hat und auf den *Save*-Button klickt, wird die Methode *saveRules()* aufgerufen. Mit der Methode werden alle Eingabefelder validiert. Eine Fehlermeldung wird angezeigt, falls alle Felder leer sind. In Erfolgsfall werden die Daten der Anfrage verarbeitet (Codezeile 1-8 in Quelltext 2). Anschließend wird eine Anfrage mit Daten an dem Microservice gesendet (Codezeile 10). Die neuen Regeln werden in der Datenbank gespeichert und ein *Confirm*-Dialog wird angezeigt, dass der Befehl erfolgreich durchgeführt wurde (Codezeile 13).

```
1.    const params ={
2.                conditions: preCondition,
3.                systemName: systemDeterminer,
4.                modal     : modalVerb,
5.                object    : objectName,
6.                anchor    : processWord,
7.                details   : postCondition
8.              };
9.
10.   axios.post("http://localhost:8080/description/addRules", params)
11.        .then(response => {
12.           if(response.status === 201){
13.               this.setConfirmation(true);
14.           }
15.   })
16.
```

Quelltext 2 POST-Anfrage der AddRules Komponente

6.2 Backend

In dem Kapitel 6.2.1 lassen sich die verwendeten Technologien beschreiben, die für Implementierung des Microservices benötigt werden. Danach wird auf den Microservice im Kapitel 6.2.2 eingegangen.

6.2.1 Verwendete Technologie

OpenNLP

In den letzten Jahren wurde eine Vielzahl der NLP-Tools wie NLTK[18], OpenNLP[19], CoreNLP,[20] Pattern[21] usw. entwickelt. Pinto et al. (Pinto et al., 2016) haben die Eigenschaften der Standard-Tools in einer Tabelle erfasst (siehe Abbildung 22), um eine Übersicht über die Funktionen der Tools zu schaffen.

System	Programming Language	Target Text	Tokenization	PoS Tagging	Chunking	NER
NLTK	Python	Generic	✓	✓	✓	✓
OpenNLP	Java	Generic	✓	✓	✓	✓
CoreNLP	Java	Generic	✓	✓	✗	✓
Pattern	Python	Generic	✓	✓	✓	✗
TweetNLP	Java	Social Media	✓	✓	✗	✗
TwitterNLP	Python	Social Media	✓	✓	✓	✓
TwitIE	Java	Social Media	✓	✓	✗	✓

Abbildung 22 Eigenschaften der NLP Tools (Pinto et al., 2016)

[18] https://www.nltk.org
[19] https://opennlp.apache.org
[20] https://nlp.stanford.edu/software/
[21] https://github.com/clips/pattern

NLTK, OpenNLP, Pattern und TwitterNLP[22] unterstützen für die Implementierung dieser Arbeit wesentlichen NLP-Techniken, die im Kapitel 2.2.1 beschrieben wurden. Darüber hinaus haben Pinto et al. (Pinto et al., 2016) die Performance der Tools auf zwei Datensammlungen getestet. Das Ergebnis zeigt, dass OpenNLP performanter als die übrigen NLP-Tools ist und bei der Richtigkeit von den sogenannten NLP-Techniken mindestens 83% liegt. Aus diesem Grund wird OpenNLP als *NLP-Tool* für die Entwicklung der Anwendung ausgewählt.

Docker

Docker[23] ist eine Plattform für Entwicklung und Ausführung der Anwendungen und ermöglicht, die Infrastruktur in verschiedenen Laufzeitumgebungen bzw. Container zu trennen und dort bereitzustellen. Zur Implementierung dieser Arbeit wird Docker verwendet, um die unterschiedlichen Container für Datenbank und das Microservice bereitzustellen.

Google Cloud Platform

Google Cloud Platform (GCP) ist eine Plattform zur Bereitstellung von Cloud Computing Services wie Datenspeicherung, Management-Tools, Datenanalyse und künstliche Intelligenz (Google Cloud, n.d.-b). Google App Engine ist ein Service von GCP und ist ein vollständig verwaltete und serverlose Plattform zur Entwicklung und Hosting von Webanwendungen in Rechenzentren (Google Cloud, n.d.-a). QDAcity wurde bereits auf Google App Engine bereitgestellt. Um die Anforderung *REQ-08* zu erfüllen, soll GCP für diese Anwendung verwendet werden.

6.2.2 Microservice

6.2.2.1 Visualisierung des Datenmodells

Für die Speicherung der neuen Regeln und Abfrage nach Regeln werden alle im Abschnitt 5.1.3 beschriebenen Entitäten verwendet. Es existiert für jede Entität eine Klasse mit den entsprechenden Attributen. Die Zugriffe und CRUD-Operationen auf die Tabelle werden durch die entsprechenden Repository-Klassen durchgeführt.

[22] https://github.com/aritter/twitter_nlp
[23] https://www.docker.com/

6.2.2.2 Einfügen der neuen Regeln

Nachdem der Benutzer auf den *save*-Button klickt, wird eine POST-Anfrage mit dem Inhalt an das Backend gesendet. Die Abbildung 23 zeigt die wesentlichen Komponenten im Backend für das Einfügen der neuen Regeln.

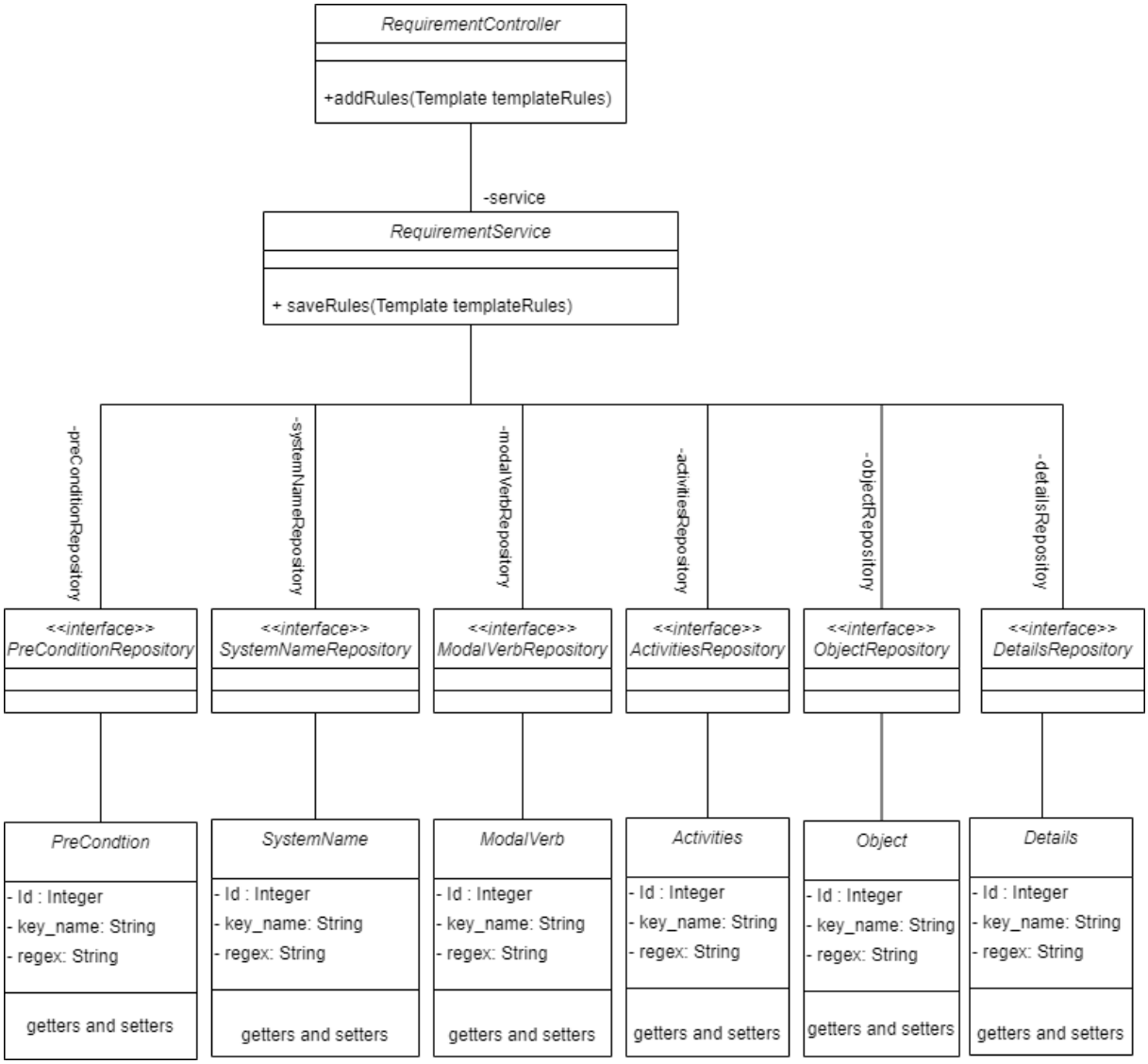


Abbildung 23 Klassendiagramm für Einfügen der neuen Regeln

Die Anfrage wird über REST-API mit der Methode *addRules* in die Komponente *RequirementController* empfangen. Das Quelltext 3 zeigt die Implementierung der *addRules* Methode.

```
1.    @RequestMapping(path = "/addRules", method = RequestMethod.POST)
2.    public ResponseEntity<HttpStatus> addRules(@Valid @RequestBody Template templateRules)
3.                        throws FileNotFoundException, IOException {
4.
5.              if (templateRules == null) {
6.                        return new ResponseEntity<>(HttpStatus.NO_CONTENT);
7.              } else {
8.                        service.saveRules(templateRules);
9.                        return new ResponseEntity<>(HttpStatus.CREATED);
10.             }
11.   }
12.
```

Quelltext 3 Methode in der RequirementController- Klasse zum Speichern der neuen Regeln

Die empfangenen Daten werden erst auf null überprüft (Codezeile 5). Falls der Wert null ist, wird die Weboberfläche mit einem HTTP Statuscode *NO_CONTENT* informiert. Falls nicht, werden die Daten an die *RequirementService*-Klasse über den Aufruf der *saveRules*-Methode weitergeleitet (Codezeile 8). Schließlich ruft die *RequirementService*-Klasse die *save*-Methode der *Repository*-Klasse von den entsprechenden Entitäten auf, um die Daten in der Datenbank zu speichern.

6.2.2.3 Überprüfung der Anforderungsbeschreibungen

Nachdem der Benutzer auf das *Check*-Button auf der Weboberfläche klickt, wird eine PUT-Anfrage an das Backend gesendet. Die Abbildung 24 zeigt eine Übersicht der wesentlichen Komponenten in der Backend Schicht, die zur Validierung der Anforderungsbeschreibungen verantwortlich ist.

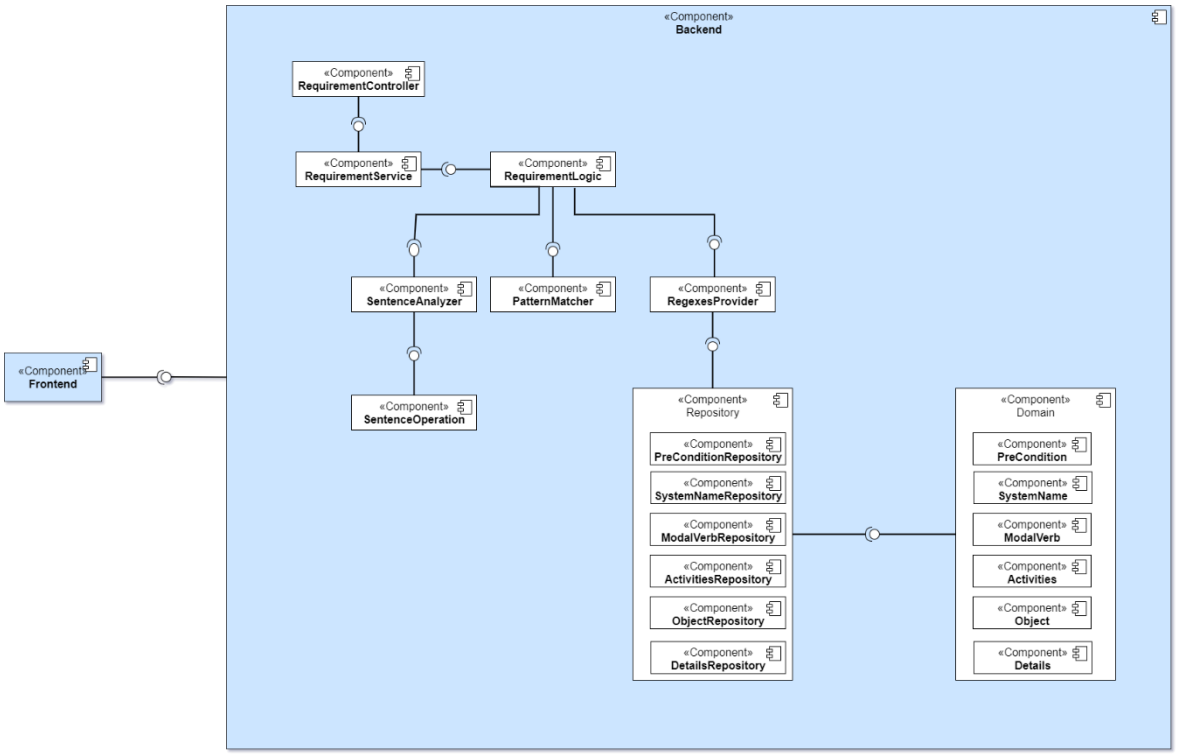


Abbildung 24 Komponentendiagramm für Überprüfung der Anforderungsbeschreibungen

Die Anfrage wird in der *checkRequirements*-Methode der *RequirementController*-Klasse abgefangen. Die Quelltext 4 zeigt die Implementierung der *checkRequirements* Methode.

```
1.  /**
2.     * check requirements description
3.     * @param requirement the requirements description
4.     * @return ResponseEntity<List<Map<Integer, String>>>
5.     */
6.    @RequestMapping(path = "/check", method = RequestMethod.PUT)
7.    public ResponseEntity<List<Map<Integer, String>>> checkRequirements(@Valid @RequestBody
    Requirement requirement) {
8.
9.              if (requirement == null) {
10.                     return new ResponseEntity<>(null, HttpStatus.NO_CONTENT);
11.             }
12.             List<Map<Integer, String>> response =
    service.checkRequirements(requirement.getDescription());
13.             return new ResponseEntity<>(response, HttpStatus.OK);
14.  }
```

Quelltext 4 Methode des REST-Controllers zur Überprüfung der Anforderungsbeschreibungen

Der Parameter der Methode wird vorerst überprüft (Codezeile 9). Falls der Wert null ist, wird eine Rückgabe mit HTTP Statuscode *NO_CONTENT* an das Frontend gesendet. Beim Erfolgsfall wird die Anforderung an die *RequirementService*-Klasse weitergeleitet, um die wesentlichen Schritte zur Überprüfung durchzuführen (Codezeile 12). Hierbei werden die Anforderungen an die *RequirementLogic*-Klasse gesendet und werden dort auf einzelne Sätze aufgeteilt. Die Modelle der OpenNLP-Bibliothek werden durch die *SentenceOprations*-Klasse

geladen, damit die *SetenceAnalyzer*-Klasse die Tokens, Tags, Chunks aus dem Satz analysieren kann. Die *RequirementLogic*-Klasse wird anhand der Tokens, Tags, und Chunks das Satz mit den regulären Ausdrücken aus der Datenbank mithilfe der *PatternMatcher*-Klasse verglichen. Die regulären Ausdrücke werden über die *Repository*-Klassen geladen und durch *RegexesProvider*-Klasse verteilt. Am Ende wird das Ergebnis nach der Überprüfung an die Frontend Schicht mit dem HTTP Statuscode *OK* gesendet und wird dort angezeigt (Codezeile 13).

6.3 Kommunikation zwischen Frontend und Backend

Axios[24] ist eine JavaScript Bibliothek und dient als ein HTTP-Client, um die Anfrage an Server zu senden und die Antwort vom Server zu verarbeiten (*Axios*, n.d.). Die Weboberfläche kann mithilfe von *axios* die Anfrage an dem Backend zusenden. Hierbei greift die Weboberfläche auf Port 8080 zu, auf dem der Microservice gestartet wurde.

Die Anfrage *axios.put(url)* und *axios.post(url)* werden verwendet, um mit dem Backend zu kommunizieren und das Antwortobjekt zu erhalten. Folgend lässt sich die Kommunikation zwischen Frontend und Backend in der Darstellung eines Sequenzdiagramm in Abbildung 25 abbilden.

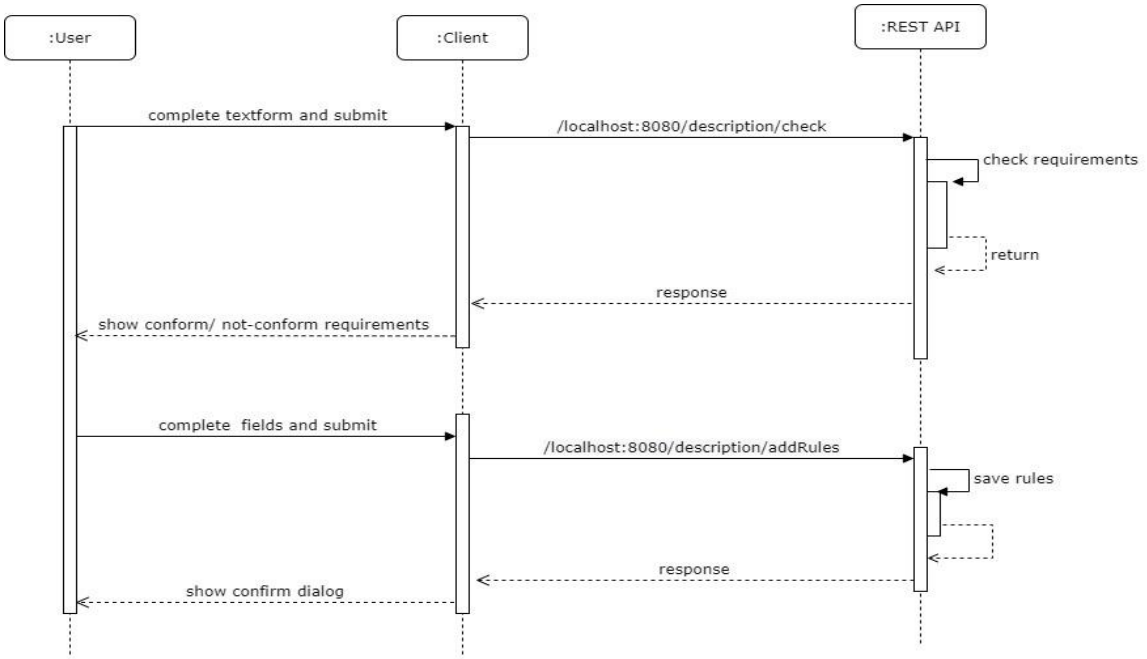


Abbildung 25 Kommunikation zwischen Frontend und Backend

[24] https://axios-http.com/

Eine PUT-Anfrage mit dem entsprechenden API-Endpunkt und die Eingabe werden von Frontend Schicht zu dem Microservice gesendet. Die Eingabe wird im Backend verarbeitet und wird anschließend als das Antwortobjekt zurück an die Frontend Schicht zurückgesendet.

Eine POST-Anfrage mit der URL */description/addRules* und den Daten der Textfelder werden an das Backend gesendet, um die Daten in der Datenbank zu persistieren. Anschließend wird eine Antwort zur Bestätigung an Frontend gesendet.

6.4 Testen

Unit-Tests wurden für das Testen der Backend Schicht verwendet. JUnit[25] ist ein Framework zum Testen von Methoden bzw. Klassen und unterstützt Unit-Tests in Java. Hiermit können die Funktionalitäten und konkreten Verhalten der Anwendung getestet werden. Für jede Komponente der Backend Schicht wird eine Testklasse implementiert. Außer die Klassen für Entitäten des Datenmodells, Interfaces und das Importieren der *OpenNLP*-Module wurde kein Test gebaut, da diese Klassen keine Logik umfassen und *getters* und *setters* Methoden beinhalten. Außerdem wurde eine Testsuite implementiert, in der alle Testklasse in Reihenfolge aufgerufen werden.

[25] https://junit.org/junit5/

7 Evaluierung

In diesem Kapitel werden die im Kapitel 4 vordefinierten Anforderungen an dieser Arbeit überprüft. Die Erfüllung der Anforderungen wird mithilfe von *erfüllt*, *teilweise erfüllt* oder *nicht erfüllt* gekennzeichnet.

7.1 Funktionale Anforderungen

REQ-01: *Die Anwendung muss dem Benutzer die Möglichkeit bieten, die Anforderungen zu beschreiben und zu überprüfen: erfüllt*

> **REQ-01.1**: *Die Anwendung muss ein Textfeld beinhalten, um die Anforderungsbeschreibungen einzugeben: erfüllt*
>
> **REQ-01.2**: *Die Anwendung muss ein Button beinhalten, um die Anfragen zu senden und die Antworte von Server zu empfangen: erfüllt*

Die Weboberfläche bietet Eingabefelder für Anforderungen und einen Button, um die eingegebenen Anforderungen an Server zu senden. Die Abbildung 26 gibt einen Überblick über die Startseite der Weboberfläche.

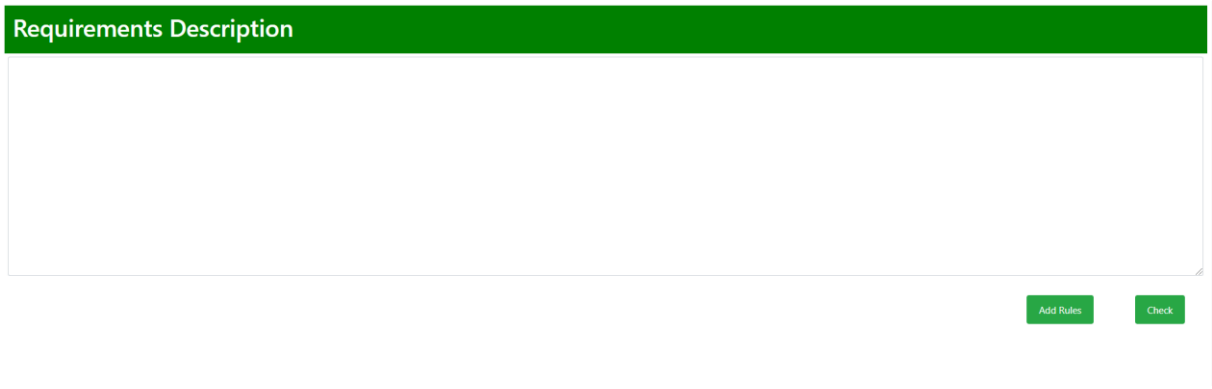


Abbildung 26 Startseite der Anwendung

REQ-02: *Die Anwendung muss auf Google Cloud Plattform bereitgestellt werden: erfüllt*

Ein Service von Cloud Run wurde erstellt, um eine serverlose URL für die Anwendung bereitzustellen. Zusätzlich wurde ein Cloud SQL-Service erstellt, um die im Abschnitt 5.1.3 beschriebenen Entitäten zu speichern. Die Anwendung kann unter der URL *https://nlp4re-f4inxobjmq-ey.a.run.app* aufgerufen werden.

REQ-03: *Die Anwendung muss dem Benutzer die Möglichkeit bieten, die konformen und nicht konformen Anforderungen zu zeigen: erfüllt*

REQ-03.1: *Die Anwendung sollte die Gründe für die nicht konformen Anforderungen zeigen: erfüllt*

Die Weboberfläche zeigt die konformen und nicht-konformen Anforderungen. Falls die Anforderungen als nicht konform klassifiziert werden, können die Gründe per Mausklick angezeigt werden. Ein Beispiel für die Funktionalität ist in der Abbildung 20 im Abschnitt 6.1 zu sehen.

7.2 Nicht-funktionale Anforderungen

– **Portability**

REQ-05: *Die Anwendung sollte gestaltet sein, dass diese in verschiedenen Browsern wie Google Chrome, Firefox usw. ausführbar ist: erfüllt*

Die von Cloud Run erzeugte URL wurde in verschiedenen Browsern aufgerufen. Alle Funktionalitäten der Anwendung funktionierten einwandfrei und identisch in unterschiedlichen Browsern.

– **Maintainability**

- **Reusability**

REQ-06: *Die Anwendung sollte gestaltet sein, dass gemeinsam nutzbarer Code von den existierten Komponenten bei neuen Komponenten wiederverwendet werden kann und nicht mehrfach definiert wird: erfüllt*

Die Gemeinsamkeiten der Komponenten wurden in der Implementierungsphase erkannt und entsprechend ausgelagert. Dadurch wird redundanter Code vermieden.

- **Modifiability**

REQ-07*: Die Anwendung muss gestaltet sein, dass sie einfach erweitert werden kann*: *erfüllt*

Wie in den Abschnitten 5 und 6 beschrieben wurde die Anwendung modularisiert entworfen und implementiert, damit es um neue Funktionen einfach erweitern kann.

- **Modularity**

REQ-08: *Die Anwendung muss gestaltet sein, dass sie vom bestehenden QDAcity entkoppelt bzw. verbunden werden kann: erfüllt*

Die im Abschnitt 5.1 beschriebene Architektur der Anwendung orientiert sich an der Microservice Architektur, um diese Anwendung mit QDAcity verbunden bzw. entkoppelt werden zu können.

- **Testability**

REQ-09: *Alle Methode der Anwendung müssen mit Unit-Tests abgedeckt werden: teilweise erfüllt*

Die Entität-Klassen beinhalten nur *getters* und *setters* Methode und umfassen keine Logik. Daher ist es nicht nötig, die alle Methode zu testen.

REQ-10: *Die gesamte Testabdeckung der Anwendung sollte bei mindestens 75% liegen: erfüllt*

Die Testabdeckung ist bei 77%. Für die Darstellung der Testabdeckung wird Codecov[26] verwendet.

– **Performance**

- **Time Behaviour**

REQ-11: *Die Anwendung muss gestaltet sein, dass die Ausführungszeit eines Diensts unter eine Minute begrenzt ist: teilweise erfüllt*

Die Zeit zur Überprüfung der Anforderungsbeschreibungen hängt von der Größe der Anforderungsdokumente ab. Deshalb wird dieser Anforderung teilweise erfüllt.

[26] https://codecov.io/gh

8 Diskussion

Im Hinblick auf die Vorteile bietet diese Anwendung dem Nutzer einen Ansatz für die automatische Überprüfung nach Konformität der Anforderungen. Die nicht-konformen Anforderungen werden markiert und die Gründe für die Nichtkonformität werden angezeigt. Dies ermöglicht, gegen mögliche Fehler präventiv vorzugehen und diese besser zu verfolgen und schneller korrigierbar zu machen. Zusätzlich kann der Aufwand bei der Überprüfung für eine große Sammlung der Anforderungen verringert werden. Darüber hinaus können die Spracheffekte wie Mehrdeutigkeiten bei den Anforderungsdokumenten reduziert und die Qualität der Anforderungen erhöht werden.

Im Abschnitt 3 wurden die unterschiedlichen Anforderungsschablonen vorgestellt. Nach der Diskussion im Abschnitt 3.7 wurde die Mazo & Jaramillo Anforderungsschablone für die Entwicklung der Anwendung gewählt, da diese Schablone mehr Vorteile zum Vergleich zu anderen Schablonen anbietet. Diese Anforderungsschablone wurde basierend auf der MASTeR Anforderungsschablone entwickelt, um die Mehrdeutigkeit bei der Anwendung dieser Anforderungsschablone zu reduzieren. Außerdem wurden die im Abschnitt 3.2 beschriebene Easy Approach to Requirements Syntax (EARS) und der Standard ISO/IEC/IEEE 29148:2011 in der Entwicklungsphase der Mazo & Jaramillo Anforderungsschablone berücksichtigt. Um die Erweiterung dieser Anforderungsschablone zu unterstützen, bietet diese entwickelte Anwendung den Nutzern eine Weboberfläche mit Eingabefeldern, um neue Regeln für die Anforderungsschablone hinzuzufügen (siehe Abbildung 21).

Neben den Vorteilen gibt es auch den Optimierungsbedarf für den entwickelten Ansatz. Die Anwendung unterstützt die Überprüfung in englischer Sprache, da die Regeln der Mazo & Jaramillo Anforderungsschablone auf Englisch sind. Zusätzlich bietet OpenNLP-Bibliothek die vollständigen Modelle wie Chunking, Parser, Part-Of-Speech Tagging und Tokenizing für die englische Sprache, bei anderen Sprachen fehlen noch Chunking und Parser Modelle. Aufgrund des Zeitrahmens dieser Arbeit wurde die Überprüfung der in anderen Sprachen formulierten Anforderungen noch nicht entwickelt. Deshalb empfiehlt es sich zukünftig die Anwendung, um weitere Sprachen zu ergänzen. Darüber hinaus soll ein Nutzertest durchgeführt werden, um zu testen, ob die ausgewählte Anforderungsschablone mit der entwickelten Anwendung in der Praxis gut umsetzbar ist. Aktuell fokussiert diese entwickelte Anwendung nur auf Mazo & Jaramillo Anforderungsschablone. Zukünftig werden die anderen Anforderungsschablonen vorgestellt, die möglicherweise besser als die Mazo & Jaramillo Anforderungsschablone sind. Deshalb empfiehlt es sich diese Anwendung, um andere Anforderungsschablonen zu erweitern.

Basierend auf der Evaluierung im Abschnitt 7 kann man festhalten, dass das Ziel dieser Arbeit erreicht wurde.

9 Zusammenfassung

Ziel dieser Arbeit ist es, Anforderungsbeschreibungen von Softwaresystemen automatisiert mit Anforderungsschablonen zu überprüfen. Basierend auf der Anforderungsschablone und Natural Language Processing (NLP) soll eine Anwendung zur Evaluierung der Anforderungsbeschreibungen als Microservice implementiert werden, um die zu entwickelnde Anwendung künftig in eine existierende Anwendung integrieren zu können.

Im Kapitel 2 wurden die Begriffe zu Requirements Engineering (RE) definiert und die Qualitätskriterien für die Software- und Systemanforderungen vorgestellt. Als nächstes wurden die Grundlagen zu NLP in RE im Kapitel 2.2 erörtert. Daraufhin erfolgte die Betrachtung der NLP Techniken wie Tokens, Part-Of-Speech Tagging, Parser und Chunking, mit denen die Anwendung entwickelt wurde.

Im dritten Kapitel wurde auf den Stand der Wissenschaft zu den Anforderungsschablonen eingegangen, um die passende Anforderungsschablone für die Entwicklung der Anwendung auszuwählen.

Die Anforderungen an diese Arbeit wurden im Kapitel 4 definiert. Hierbei wurde zunächst der Zweck dieser Arbeit erläutert. Danach folgt die Beschreibung der funktionalen und nicht-funktionalen Anforderungen.

Im Kapitel 5 wurde auf die Architektur und Design der Anwendung eingegangen. Hierfür wurden die Frontend, Backend Schichten und die jeweiligen Komponenten dieser Schicht beschrieben. Letztendlich wurde eine dynamische Sicht der Anwendung vorgestellt, die Interaktionen zwischen Frontend und Backend beschreibt.

Nach der Architektur und Design kommt die Implementierung dieser Arbeit, die im Kapitel 6 betrachtet wurde. Hierbei wurden die für die Entwicklung dieser Arbeit benötigten Technologien wie Docker, Google Cloud Platform vorgestellt. Danach folgt die Implementierung der Frontend und Backend Schicht sowie die Kommunikation der beiden Schichten. Schließlich wurde die Implementierung des Testens für die Entwicklung betrachtet.

Im Kapitel 7 wurde auf die Evaluierung der Arbeit eingegangen. Die funktionalen und nicht-funktionalen wurden hier bewertet, ob diese erfüllt, teilweise erfüllt oder nicht erfüllt wurden.

Zuletzt wurde die entwickelte Anwendung im Kapitel 8 diskutiert, in dem die Stärken und Schwächen der Anwendung betrachtet wurden.

Durch die Anwendung von NLP in RE kann die Qualität der Anforderungsbeschreibungen verbessert werden. Damit können Fehler im RE vorgebeugt und die Kosten für nötige

Korrekturen der Fehler in den nächsten Phasen eines Softwareprojekts vermieden werden. Potenziell verringert dies die Wahrscheinlichkeit für das Scheitern des Projekts.

Reference

Axios. (n.d.). Retrieved June 2, 2021, from https://axios-http.com/

Bauer, W., & Warschat, J. (2020). *Smart Innovation durch Natural Language Processing: Mit Künstlicher Intelligenz die Wettbewerbsfähigkeit verbessern*.

Bird, S., Klein, E., & Loper, E. (2009). *Natural Language Processing with Python*.

Dalpiaz, F., Ferrari, A., Franch, X., & Palomares, C. (2018). *Natural Language Processing for Requirements Engineering The Best Is Yet to Come*. *October*, 115–119.

Denger, C., Berry, D. M., & Kamsties, E. (2003). Higher quality requirements specifications through natural language patterns. *Proceedings - IEEE International Conference on Software- Science, Technology and Engineering, SwSTE 2003*, 80–90. https://doi.org/10.1109/SWSTE.2003.1245428

Eckhardt, J., Vogelsang, A., Femmer, H., & Mager, P. (2016). Challenging Incompleteness of Performance Requirements by Sentence Patterns. *Proceedings - 2016 IEEE 24th International Requirements Engineering Conference, RE 2016*, 46–55. https://doi.org/10.1109/RE.2016.24

Existek.com. (n.d.). *Top Front-End Frameworks in 2021 | Existek Blog*. Retrieved May 29, 2021, from https://existek.com/blog/top-front-end-frameworks-2021/

Google Cloud. (n.d.-a). *Dokumentation zu Google App Engine | App Engine-Dokumentation*. Retrieved May 30, 2021, from https://cloud.google.com/appengine/docs?hl=de

Google Cloud. (n.d.-b). *Products and Services* . Retrieved May 30, 2021, from https://cloud.google.com/products

Grande, M. (2014). *100 Minuten für Anforderungsmanagement*.

Hellwig, P. (1989). Computational Linguistics / Computerlinguistik. *Computational Linguistics/Computerlinguistik*, 348–378.

IEEE Standards Board. (1990). IEEE Standard Glossary of Software Engineering Terminology. *Office*, *121990*(1), 1. https://doi.org/10.1109/IEEESTD.1990.101064

ISO/IEC/IEEE 29148: 2011. (2011). *Systems and software engineering — Life cycle processes — Requirements engineering*. 1–83.

ISO/IEC 25010:2011. (2011). *ISO/IEC 25010:2011(en), Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*. https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en

Jurafsky, D., & Martin, J. H. (2020). *Speech and Language Processing*.

Kamsties, E., & Paech, B. (2000). *Taming Ambiguity in Natural Language Requirements*.

Kof, L. (2005). Natural Language Processing: Mature Enough for Requirements Documents Analysis? In *Lecture Notes in Computer Science* (Vol. 3671).

Larrucea, X., Santamaria, I., Colomo-palacios, R., & Ebert, C. (2021). *Microservices*.

Liddy, E. D. (2001). Natural language processing. *Artificial Intelligence*, *19*(2), 131–136. https://doi.org/10.1016/0004-3702(82)90032-7

Majumdar, D., Sengupta, S., Kanjilal, A., & Bhattacharya, S. (2011). Automated Requirements Modelling With Adv-Ears. *International Journal of Information Technology Convergence and Services*, *1*(4), 57–67. https://doi.org/10.5121/ijitcs.2011.1406

Mavin, A., Wilkinson, P., Harwood, A., & Novak, M. (2009). EARS (Easy Approach to Requirements Syntax). *Proceedings of the IEEE International Conference on Requirements Engineering*, *October*, 317–322. https://doi.org/10.1109/RE.2009.9

Mazo, R., Jaramillo, C., Medina, J. M., & Vallejo, P. (2020). Towards a new template for the specification of requirements in semi-structured natural language. *Journal of Software Engineering Research and Development*, *8*(Sophist 2014), 3. https://doi.org/10.5753/jserd.2020.473

Mich, L., Franch, M., & Novi Inverardi, P. L. (2004). Market research for requirements analysis using linguistic tools. *Requirements Engineering*, *9*(2), 151–151. https://doi.org/10.1007/s00766-004-0195-3

microservices.io. (n.d.). *What are microservices?* Retrieved May 27, 2021, from https://microservices.io/

Neumann, A., Laranjeiro, N., & Bernardino, J. (2018). *An Analysis of Public REST Web Service APIs*. *November*. https://doi.org/10.1109/TSC.2018.2847344

Pinto, A., Oliveira, H. G., & Alves, A. O. (2016). Comparing the performance of different NLP toolkits in formal and social media text. *OpenAccess Series in Informatics*, *51*(3), 31–316. https://doi.org/10.4230/OASIcs.SLATE.2016.3

Pohl, K., & Rupp, C. (2015a). *Basiswissen Requirements Engineering : Aus- und Weiterbildung nach IREB-Standard zum Certified Professional for Requirements Engineering Foundation Level*. *April*, 35–52.

Pohl, K., & Rupp, C. (2015b). Requirements Engineering Fundamentals A Study Guide for the Certified Professional for Requirements Engineering Exam. In *Decisions*.

https://doi.org/10.4337/9781788110396.00004

Rakshith, R. R., & Swamy, S. R. (2020). Review on Spring Boot and Spring Webflux for Reactive Web Development. *International Research Journal of Engineering and Technology (IRJET)*, *0*(0), 0.

Reactjs.org. (n.d.). *React*. Retrieved May 29, 2021, from https://reactjs.org/docs/faq-internals.html#gatsby-focus-wrapper

Rupp, C., & Joppich, R. (2014). *Anforderungsschablonen — der MASTER-Plan für gute Anforderungen*. 215–246.

Sharma, N. (2018). *Integrating Natural Language Processing and Software Engineering*. *November 2015*. https://doi.org/10.14257/ijseia.2015.9.11.12

Singh, M. ., & Vyas, R. (2012). Analysis of Requirement Engineering Problems and Proposed Solutions. *... in Computer Science and Electronics Engineering ...*, *1*(7). http://www.ijarcsee.org/index.php/IJARCSEE/article/view/174

Singh, S. (2010). *Natural Language Processing for Information Extraction*. 1–24.

Steinkamp, J. M. (2019). *Toward Complete Structured Information Extraction from Radiology Reports Using Machine Learning*. 554–564.

Toro, a D., Jiménez, B. B., Cortés, a R., & Bonilla, M. T. (1999). A Requirements Elicitation Approach Based in Templates and Patterns ? *Requirements Engineering*, 17–29. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.92.4047&rep=rep1&type=pdf

Villavicencio, A., Moreira, V., Abad, A., Caseli, H., Gamallo, P., Ramisch, C., Gonçalo Oliveira, H., & Henrique Paetzold, G. (2018). *Computational Processing of the Portuguese Language*. https://doi.org/10.1007/978-3-319-99722-3

Zhao, L., Alhoshan, W., Ferrari, A., Letsholo, K. J., Ajagbe, M. A., Chioasca, E. V., & Batista-Navarro, R. T. (2020). Natural Language Processing (NLP) for requirements engineering: A systematic mapping study. *ArXiv*, *v*.