ulm university universität ulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für Ingenieurwissenschaften und Informatik
Institut für Datenbanken und Informationssysteme

Konzeption und Entwicklung einer Web-Applikation für kollaboratives Checklisten-Management

Bachelorarbeitan der Universität Ulm

Vorgelegt von:
Norman Thiel
norman.thiel@uni-ulm.de

Gutachter:
Prof. Dr. Manfred Reichert

Betreuer:
Nicolas Mundbrod

2013

Fassung 13. November 2013

Satz: PDF-LaTeX 2$_\varepsilon$

Kurzfassung

Der Stellenwert der Wissensarbeit nimmt in der heutigen Wirtschaft stetig zu. Die Schaffung und Verbreitung neu erworbener Kenntnisse steht im Vordergrund. Um die Wissensarbeit zu fördern, müssen geeignete Hilfsmittel entwickelt werden. Da Wissensarbeit jedoch keinem planbaren Prozess unterliegt, ist die Entwicklung geeigneter Hilfsmittel schwer umsetzbar.
Im Rahmen des Projektes proCollab am Institut DBIS, der Universität Ulm, wird untersucht in wie weit sich Checklisten für die Unterstützung von Wissensarbeit eignen. In der vorliegenden Arbeit wird eine Administrationsumgebung einer Checklisten-Applikation konzipiert und entwickelt. Die Administrationsumgebung ist für die Verwaltung der Checklisten und ihrer kontextuellen Elemente verantwortlich.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Die Wissensarbeit nimmt in unserer heutigen Informationsgesellschaft einen immer größeren Stellenwert ein [PS98]. Immer mehr Menschen arbeiten an der Schaffung und Verbreitung neuen Wissens. Standardisierte Tätigkeiten, wie die Produktion von Gütern, werden in andere Länder ausgelagert. Damit steigt der Anteil an Wissensarbeit in vielen Ländern weiter. Klassische Berufe der Wissensarbeit sind zum Beispiel Forschung, Entwicklung oder die Medizin.
Für den Erfolg der Wissensarbeit ist es nicht ausreichend, nur die Korrektheit neuer Erkenntnisse heranzuziehen, denn auch fehlerhafte Erkenntnisse sind ein Gewinn und eine Erhöhung der Wissensgrundlagen. Ein weiterer Erfolgsfaktor ist die Effizienz, also wie schnell neues Wissen erarbeitet und verbreitet wird. Eine Möglichkeit die Effizienz von Wissensarbeit zu erhöhen besteht darin, sie zu unterstützen.

1.2 Problemstellung

Wissensarbeit unterliegt keinem planbaren Prozess. Sie ist unvorhersehbar und zeitlich unbestimmt, Erkenntnisse und eventuell auftretende Probleme können nicht kalkuliert

werden [Hub05]. Ebenfalls unterliegt die Wissensarbeit einer hohen Dynamik, da der Ablauf der Wissensarbeit von den Kenntnissen der Wissensarbeiter abhängt.
Bei der Güterproduktion können Maschinen entwickelt werden, die die Produktion automatisieren und die Effizienz erhöhen. Dies ist möglich, da man Vorgaben und zeitliche Abläufe bestimmen kann. Da dies jedoch bei der Wissensarbeit nicht machbar ist, gestaltet sich hier die Entwicklung geeigneter Hilfsmittel als schwierig. Um Wissensarbeit zu fördern bedarf es Hilfsmittel, die die Erkenntnisgewinnung unterstützen. Das heißt, dass die Förderung der Wissensarbeit durch eine Unterstützung der Wissensarbeiter, bei ihrem Wissen schaffenden Prozess, erfolgt.
Wissensarbeiter auf vielen Wissensgebieten nutzen bereits Checklisten, zur Unterstützung der Wissensarbeit. Checklisten sind hervorragend dafür geeignet definierte Aufgaben zu dokumentieren, strukturieren und Ergebnisse zu notieren. Jedoch werden Checklisten bis heute auf Papier festgehalten.
Papierbasierte Checklisten haben jedoch erhebliche Nachteile. Geht eine solche Checkliste verloren, sind unter Umständen auch Ergebnisse verloren. Im schlechtesten Fall muss die Arbeit von vorne beginnen. Ebenfalls ist es nicht möglich Vorlagen zu erstellen. Wenn nur einige Fragen ausgetauscht werden müssen, weil eine Checkliste in verschiedenen Kontexten genutzt werden soll, müssen alle Checklisten einzeln erstellt werden. Eine Synchronisation aller Checklisten unter den Bearbeitern, ist nur durch das Zusammentragen der Ergebnisse aller Checklisten möglich. Der Gefahr der Ergebnisverfälschung ist nur durch Sorgfalt zu entgegnen.

1.3 Beitrag und Ziele

Diese Arbeit ist Teil des proCollab Projektes. In dem Projekt soll evaluiert werden ob sich digitalisierte Checklisten zur Unterstützung von Wissensarbeit eignen. Dafür wird, basierend auf Fallstudien und der theoretischen Grundlage der Wissensarbeit, ein Software Prototyp für ein innovatives Checklisten-Managements entwickelt.
Die Architektur der Software unterliegt einer strengen Trennung zwischen dem Speichern, Verarbeiten und Anzeigen von Daten (siehe Abbildung 1.1). Diese einzelnen Komponenten werden voneinander unabhängig implementiert. Die Kommunikation zwi-

schen den Komponenten findet über fest definierte Schnittstellen statt. Da die Software auf möglichst vielen Endgeräten zur Verfügung stehen soll, werden Implementierungen für verschiedene Endgeräte vorgenommen.

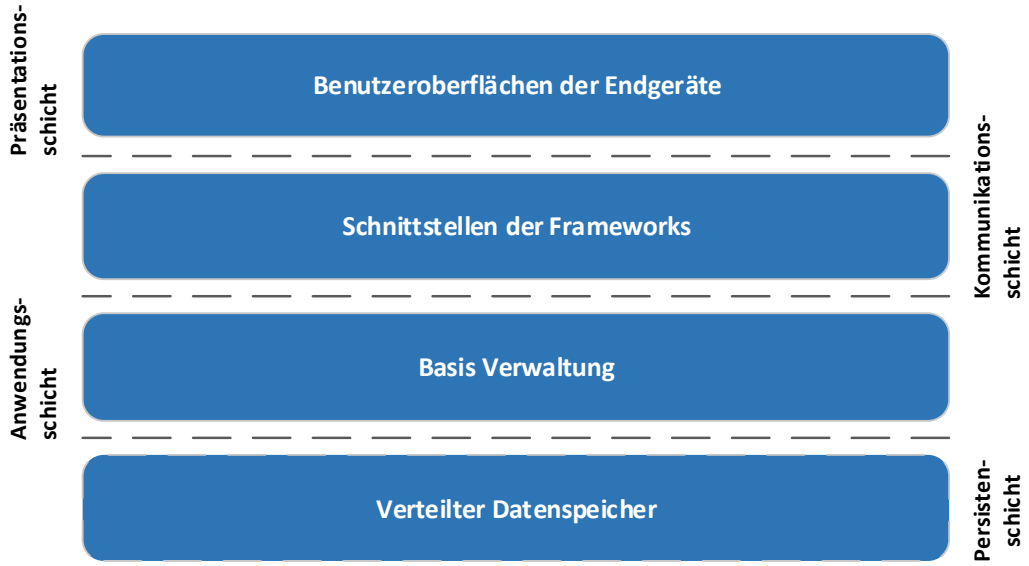


Abbildung 1.1.: ProCollab Systemarchitektur

In dieser Arbeit wird eine administrative Umgebung entwickelt, die über den Browser erreichbar ist. Da diese speziell für den administrativen Gebrauch implementiert wird, unterliegt sie gesonderten Richtlinien. Software, für die verschiedene Benutzergruppen definiert sind, haben in der Regel eine Administrationsumgebung. Diese dient der Wartung und Verwaltung kontextueller Elemente des proCollab Prototypen. Die Administrationsumgebung bietet auch Schutz gegenüber unerwünschten Handlungen. Beispielsweise können Nutzer, die unsachgemäßen Gebrauch der Software vornehmen, vom System entfernt werden oder eine fehlerhafte Nutzung korrigiert werden. Da Administrationsumgebungen zumeist in hohem Maße Schreibrechte an der jeweiligen Software anbieten, sollte der Zugang nur über eine angemessene Authentifizierung erfolgen.
Ziel ist es eine administrative Benutzeroberfläche zu entwickeln, die eine Verwaltung der

Checklisten und ihrer kontextuellen Elemente ermöglicht. Eine hohe Bedienbarkeit der Benutzeroberfläche, sowie ein schneller Datenaustausch und kurze Übertragungszeiten werden angestrebt.

1.4 Aufbau der Arbeit

Kapitel 1 schafft ein Überblick über die Thematik und die vorliegende Arbeit. In Kapitel 2 werden grundlegende Begriffe definiert und Anwendungsfälle von Checklisten aufgezeigt. Kapitel 3 diskutiert zunächst aktuell erhältliche Software zum Verwalten von Checklisten und eine Software mit administrativer Anzeige. Anschließend wird eine Auswahl typischer Anwendungsfälle einer Checklisten-Applikation kurz beschrieben und funktionale, sowie nicht-funktionale Anforderungen erhoben. Anhand der definierten Grundlagen und Anforderungen wird das System in Kapitel 4 konzipiert. Die Struktur der Systemarchitektur wird erläutert und die einzelnen Schichten benannt. Im Anschluss wird die Administrationsumgebung entwickelt und das Zusammenspiel der einzelnen Komponenten beschrieben. Die, bei der Benutzeroberfläche, zugrunde liegenden Prinzipien werden erklärt und Modelle der Benutzeroberfläche, in Form von MockUps, präsentiert. Auf Basis der Konzeption wird die Implementierung des Systems in Kapitel 5 vorgestellt. Einer Übersicht aller verwendeten Technologien folgen beispielhafte Ausschnitte der Implementierung. Die Verwendung von Quellcodebeispielen und Screenshots veranschaulicht dies. In Kapitel 6 wird die Arbeit zusammengefasst und ein Ausblick gegeben.

Kapitel 2

Grundlagen

In diesem Kapitel werden die Grundlagen dieser Arbeit gelegt. Es wird geklärt, was der Begriff Wissensarbeit bedeutet und wie sie durch Checklisten unterstützt werden kann. Anschließend werden Anwendungsfälle von Checklisten betrachtet, sowie die grundlegenden Begrifflichkeiten einer Checkliste erläutert.

2.1 Wissensarbeit

Die Wissensarbeit ist der Prozess der Erkenntnisgewinnung. Dieser Prozess setzt sich zusammen aus dem Erwerb, der Verbreitung und der Erschaffung neuer Erkenntnisse. Dem voraus gehen kognitive, objektive und immaterielle Tätigkeiten [HR13]. Diese sind typischerweise recherchieren, untersuchen, forschen, analysieren, auswerten oder entwickeln. Wissensarbeiten sind unter anderem Forschung, empirische Studien oder die Entwicklung neuartiger technischer Systeme oder geistiger Modelle.
Im Besonderen ist Wissensarbeit von großer Bedeutung in den Industrienationen und stellt bereits heute das dominierende Arbeitsfeld dar [PS98]. Da bereits standardisierte Routinearbeiten oft in Länder mit niedrigen Gehaltsforderungen ausgelagert werden, steigt als Resultat der Anteil der Wissensarbeit in den Industrienationen an. Als Wissensarbeiter werden die an der Wissensarbeit tätigen Personen bezeichnet [MKR12].
Wissensarbeit ist unvorhersehbar, zeitlich unbestimmt und nicht planbar. Es gibt keinen

vordefinierten Prozess, der zum Ziel führt. Daher kann nicht bestimmt werden, ob und wann Ziele erreicht werden, welche Probleme auftreten werden und ob alle Vorgaben eingehalten werden können. Der Prozess der Wissensarbeit ist dynamisch, da der Ablauf von den Fachkenntnissen und Erfahrungen der beteiligten Wissensarbeiter bestimmt wird. Mit der Anzahl der zusammenarbeitenden Wissensarbeiter steigt auch die allgemeine Dynamik [MKR12].
Bei Tätigkeiten, die nicht der Wissensarbeit zugeordnet werden, sind Arbeitsabläufe und Zielstellungen klar formulierbar. Aufgaben können zugewiesen, Vorgaben definiert und auftretende Probleme eingegrenzt werden. Anhand dessen können Modelle entwickelt werden, die es ermöglichen, spezielle Software oder maschinelle Hilfen zu entwickeln, die den Arbeitsprozess erleichtern und verbessern. Während diese Arbeiten leicht unterstützt werden können, erweist sich dies bei der Wissensarbeit schwieriger.
Um die Wissensarbeit zu unterstützen, liegt der Fokus auf den Wissensarbeitern. Unterstützung wird geboten, indem Arbeiten abgenommen oder erleichtert werden. Beispielsweise erleichtert ein Diktiergerät einem Forscher, beim Niederschreiben seiner Arbeit, seine Gedanken zu rekonstruieren.
Da zumeist mehrere Wissensarbeiter zusammenarbeiten, besteht ein erhöhter Aufwand in der Kommunikation und Koordination. Zum Beispiel müssen Recherche- oder Forschungsergebnisse verbreitet und übersichtlich aufbereitet werden, um ein weiteres Vorgehen Aller bestimmen zu können. Um Kommunikation und Koordination zu erleichtern gibt es bereits geeignete Konzepte.

2.2 Unterstützung von Wissensarbeit

Bewährte Hilfsmittel der Wissensarbeiter sind papierbasierte Checklisten oder Aufgabenlisten (ToDo-Liste). Aufgabenlisten sind üblicherweise Aufzählungen von Checkpunkten, die, nach Erledigung, abgehakt werden. Eine Checkliste ist ebenfalls eine Auflistungen von Checkpunkten. Diese Checkpunkte beinhalten allerdings konkrete Fragestellungen oder Aufforderungen. Um einen Checkpunkt zu erledigen muss die Frage beantwortet werden.
Eine bessere Möglichkeit ist es, die Checkliste als Liste von Aufgaben zu betrachten.

Diese Aufgaben können in weitere darunterliegende Aufgaben aufgeteilt werden. Somit unterliegt die Checkliste einer Baumstruktur aus Aufgaben. Das folgende Beispiel in Abbildung 2.1 zeigt eine Checkliste und die möglichen Sichtweisen auf sie anhand von Aufgaben in medizinischem Umfeld. Abbildung 2.1a zeigt hierbei die traditionelle Sicht auf eine Checkliste, als Aufgabenliste mit Kästchen zum Abhaken. Abbildung 2.1b zeigt die gleiche Checkliste als hierarchischen Aufgabenbaum.

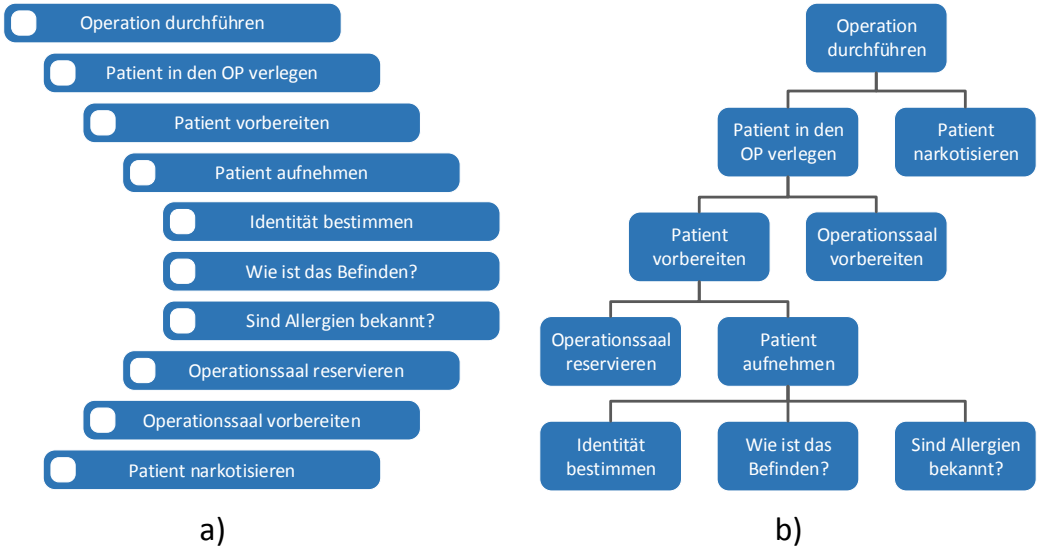


Abbildung 2.1.: Vorbereitung einer Operation als Aufgabenbaum

2.3 Anwendungsfälle von Checklisten

Checklisten haben bereits in vielen Arbeitsbereichen Einzug gehalten und werden benutzt, um bestehende Arbeitsprozesse zu verbessern oder Sorgfalt zu gewährleisten. Einige, repräsentative Anwendungsfälle werden deshalb kurz beschrieben, um Einblicke über die aktuelle Nutzung und Bedeutung von Checklisten zu gewähren.

2.3.1 Entwicklung mechatronischer Bauteile in der Automobilbranche

2.3.1.1 Bereich

In Entwicklungsprozessen für mechatronische Bauteile in der Automobilbranche sind beteiligte Entwickler angewiesen der Entwicklungsmethodik des V-Modells zu folgen [Mun]. Da jedoch ein Entwicklungsprozess nicht linear, sondern in viele Projektabschnitte aufgeteilt ist, wird die Methodik für jeden Projektabschnitt eigenständig angewendet. Ein Projektabschnitt ist beendet, sobald angestrebte Ziele erreicht werden, die durch Meilensteine definiert sind. Um diese Meilensteine zu erreichen, enthalten Projektabschnitte oft weitere eigenständige, parallele Entwicklungsprozesse.

2.3.1.2 Problem

Um die Qualität einzelner Projektabschnitte sicherzustellen, müssen diese genauestens koordiniert und Fortschritte kommuniziert werden. Hierbei kann ein Projektabschnitt die Fertigstellung eines anderen benötigen. Wird keine Übersicht geboten, kommt es hierbei zu Verzögerungen. Ingenieure unterschiedlichster Fachrichtungen sind bei der Entwicklung in verschiedenen Projektabschnitten gleichzeitig beteiligt. Diese besitzen unter Umständen nur wenig Kenntnisse über die Ergebnisse anderer. Sie benötigen daher eine Möglichkeit, ihre Ergebnisse mit den Teams zu synchronisieren und einen Überblick zu schaffen. Damit kann der Entwicklungsfortschritt verbessert und hohe Qualitätsansprüche gewährleistet werden.

2.3.1.3 Anwendung

Um die Koordination und Kommunikation der Entwickler zu verbessern sowie zur Sicherung der Qualität während der Entwicklung, wurden Checklisten eingeführt. Zu jedem Entwicklungsprozess erstellt der Qualitätssicherungsbeauftragte eine Checkliste auf Basis von standardisierten Vorlagen, die in einem extra System verwaltet werden. Dabei nutzt er typischerweise mehrere Vorlagen so, dass er diese erst für das Projekt zusammenführt und anschließend die neue Checkliste anpasst. Eine Checkliste enthält eine

Liste von Einträgen, die wiederum hierarchischen Kategorien zugewiesen sind. Jeder Eintrag enthält verschiedene standardisierte Attribute, die die Bedürfnisse der Qualitätssicherungsbeauftragten und der Entwickler widerspiegeln. Anhand der Checkliste können Fortschritte eingesehen und Ergebnisse synchronisiert werden.

2.3.2 Sicherheit bei chirurgischen Eingriffen

2.3.2.1 Bereich

Die chirurgische Versorgung ist ein wichtiger Bestandteil der weltweiten Gesundheitsversorgung. Mit dem Anstieg an medizinischen Kenntnissen und dem Fortschritt in der Medizintechnik, steigt auch die Anzahl der durch eine Operation heilbarer Krankheiten. Krebs, Herz-Kreislauf-Erkrankungen und traumatische Verletzungen spielen eine immer größere Rolle und erfordern zumeist eine chirurgische Behandlung. Oftmals ist ein chirurgischer Eingriff die einzig verbleibende Behandlung, um ein Leiden zu lindern oder einen Patienten vor dem Tod zu bewahren [Wor13a].

2.3.2.2 Problem

Jedes Jahr werden viele Millionen Menschen Opfer von Komplikationen binnen eines chirurgischen Eingriffs. Während chirurgische Eingriffe dafür vorgesehen sind, ein Leiden zu lindern oder ein Leben zu retten, kann ein einziger Fehler erheblichen Schaden verursachen und sogar zum Tod führen. Die Nichtbeachtung einer Allergie, die Blutzufuhr mit dem Blut einer falschen Blutgruppe oder eine Infektion durch fehlende Hygienemaßnahmen sind schnell gefundene Beispiele. Postoperative Begleiterkrankungen sind die Folge. Im Angesicht der weltweiten Bedeutung von chirurgischen Eingriffen hat dies erhebliche Auswirkungen auf die öffentliche Gesundheit. [Wor13a]

2.3.2.3 Anwendung

Im Jahr 2007 arbeitete die Weltgesundheitsorganisation an einer einfachen und kostengünstigen Möglichkeit die chirurgische Versorgung zu verbessern. Die Definition von

Sicherheitsstandards in Form einer Checkliste ist das Resultat (siehe Abbildung 2.2). Diese vereint die wichtigsten Informationen und Standards vor der Anästhesie, vor dem chirurgischen Eingriff und vor dem Verlassen des Operationssaals. Hierdurch wurde eine dramatische Verbesserung der chirurgischen Sorgfalt erzielt. Das Auftreten von Komplikationen während eines Eingriffs sank deutlich und somit auch postoperative Schäden und Todesfälle [Wor13a]. Die Checkliste ist in Bereiche der grundlegenden Operationsphasen aufgeteilt. Es werden vor der Anästhesie, vor dem ersten Hautschnitt und bevor der Patient den Operationssaal verlässt, Checkpunkte abgearbeitet. Als Kontrolle ist in jeder Phase angegeben, welches Personal dabei anwesend sein muss. Dadurch wird sichergestellt das die am häufigsten vorkommenden, zu vermeidenden Fehler minimiert werden.

Surgical Safety Checklist

World Health Organization | Patient Safety
A World Alliance for Safer Health Care

Before induction of anaesthesia → **Before skin incision** → **Before patient leaves operating room**

(with at least nurse and anaesthetist)

Has the patient confirmed his/her identity, site, procedure, and consent?
- ☐ Yes

Is the site marked?
- ☐ Yes
- ☐ Not applicable

Is the anaesthesia machine and medication check complete?
- ☐ Yes

Is the pulse oximeter on the patient and functioning?
- ☐ Yes

Does the patient have a:

Known allergy?
- ☐ No
- ☐ Yes

Difficult airway or aspiration risk?
- ☐ No
- ☐ Yes, and equipment/assistance available

Risk of >500ml blood loss (7ml/kg in children)?
- ☐ No
- ☐ Yes, and two IVs/central access and fluids planned

(with nurse, anaesthetist and surgeon)

- ☐ **Confirm all team members have introduced themselves by name and role.**
- ☐ **Confirm the patient's name, procedure, and where the incision will be made.**

Has antibiotic prophylaxis been given within the last 60 minutes?
- ☐ Yes
- ☐ Not applicable

Anticipated Critical Events

To Surgeon:
- ☐ What are the critical or non-routine steps?
- ☐ How long will the case take?
- ☐ What is the anticipated blood loss?

To Anaesthetist:
- ☐ Are there any patient-specific concerns?

To Nursing Team:
- ☐ Has sterility (including indicator results) been confirmed?
- ☐ Are there equipment issues or any concerns?

Is essential imaging displayed?
- ☐ Yes
- ☐ Not applicable

(with nurse, anaesthetist and surgeon)

Nurse Verbally Confirms:
- ☐ The name of the procedure
- ☐ Completion of instrument, sponge and needle counts
- ☐ Specimen labelling (read specimen labels aloud, including patient name)
- ☐ Whether there are any equipment problems to be addressed

To Surgeon, Anaesthetist and Nurse:
- ☐ What are the key concerns for recovery and management of this patient?

This checklist is not intended to be comprehensive. Additions and modifications to fit local practice are encouraged.

Revised 1 / 2009

Abbildung 2.2.: WHO surgical safety checklist [Wor13b]

2.3.3 Sicherheit in der Luftfahrt

2.3.3.1 Bereich

Die Luftfahrt hat sich in den vergangenen Jahrzehnten rasant entwickelt und ist nicht mehr mit der Zeit der Flugzeugpioniere zu vergleichen. Das Thema Sicherheit ist vorherrschend sowohl am, als auch im Flugzeug. Die Bordtechnik im Cockpit erlaubt den Überblick über alle notwendigen Daten, wie etwa die aktuelle Höhe, Geschwindigkeit oder Lage des Flugzeuges. Die Piloten unterliegen einer strikten Aufgabenteilung. Während der eine das Flugzeug fliegt, übernimmt der oder die anderen die restlichen Aufgaben. Dazu gehören die Navigation, die Überwachung des Flugfunks und aller Bordsysteme, sowie das Abarbeiten der Checklisten [Gaw09].

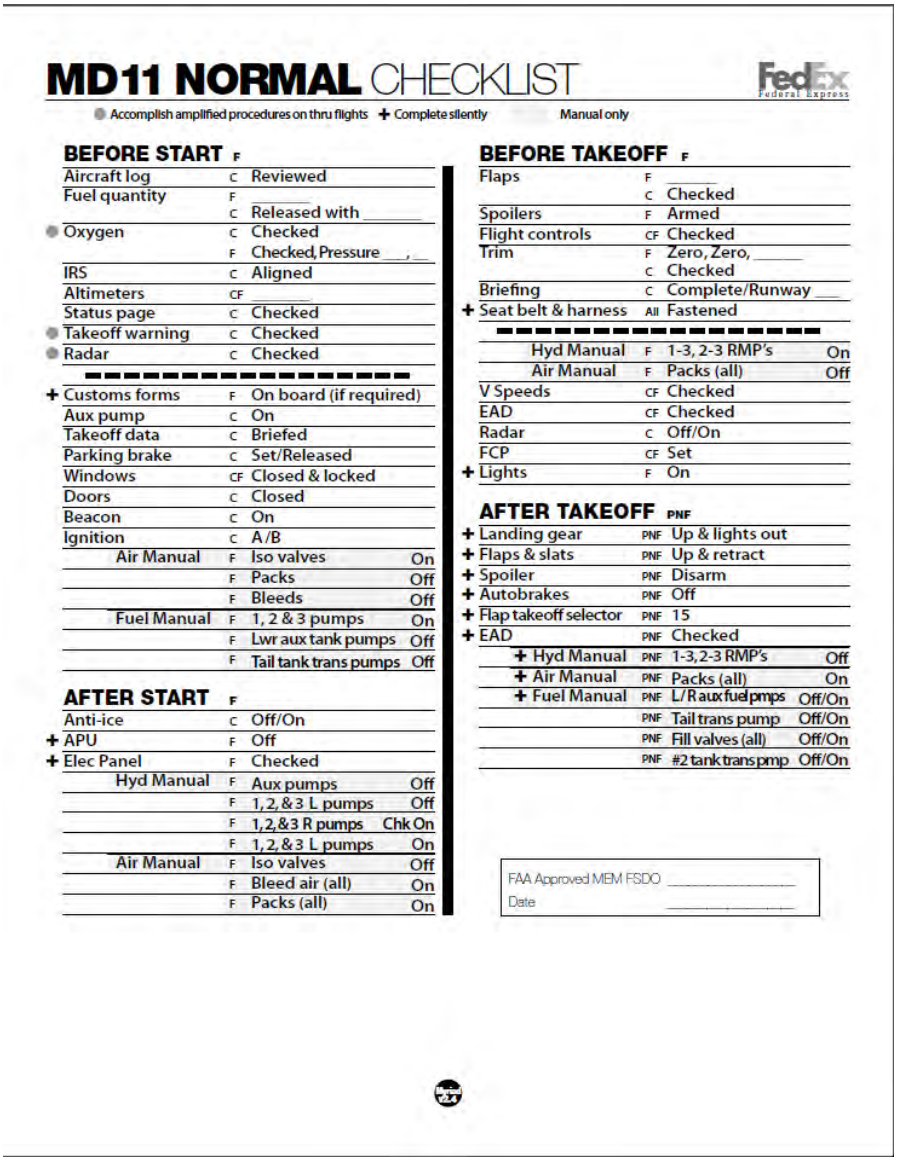
2.3.3.2 Problem

Beim Fliegen eines Flugzeuges spielen minimale Änderungen eine große Rolle. Eine zu hohe Anfluggeschwindigkeit beim Landeanflug kann das Flugzeug beschädigen oder eine Landung unmöglich machen. Werden die Räder zu spät oder gar nicht ausgefahren, endet der Landeanflug gar in einer Katastrophe. Da der Flugverkehr immer noch jährlich zunimmt ist die Bedeutung der Flugsicherheit außerordentlich hoch. Mit der Bedeutung steigt auch der Bedarf anhand eines Kontrollinstrumentes unerwünschte Situationen zu vermeiden und somit Sicherheit zu gewähren [Gaw09].

2.3.3.3 Anwendung

Für jede Flugphase, zum Beispiel Start und Landung, gibt es in der Luftfahrt etablierte Checklisten (siehe Abbildung 2.3). Diese Checklisten zeigen den Piloten alle notwendigen Einstellungen auf und ermöglichen es, diese zu überprüfen. Menschliche Unaufmerksamkeit und Vergesslichkeit wird somit vermieden. Falsche Konfigurationen werden erkannt und korrigiert. Zusätzlich sind in einigen modernen Flugzeugen Computer integriert die, wie bei einer Checkliste, bestimme Flugphasen erkennen und bei

falscher Konfiguration warnen [Gaw09]. Sinn und Zweck der Checklisten ist es keine Checklistenpunkte zu vergessen oder außer Acht zu lassen. Deshalb werden bei vielen Fluglinien Checklisten immer nur vollständig abgearbeitet. Wird diese unterbrochen, muss von vorne begonnen werden.



MD11 NORMAL CHECKLIST — FedEx Federal Express

● Accomplish amplified procedures on thru flights + Complete silently — Manual only

BEFORE START F

Aircraft log	C	Reviewed	
Fuel quantity	F	_____	
	C	Released with _____	
● Oxygen	C	Checked	
	F	Checked, Pressure ___, __	
IRS	C	Aligned	
Altimeters	CF	_____	
Status page	C	Checked	
● Takeoff warning	C	Checked	
● Radar	C	Checked	
+ Customs forms	F	On board (if required)	
Aux pump	C	On	
Takeoff data	C	Briefed	
Parking brake	C	Set/Released	
Windows	CF	Closed & locked	
Doors	C	Closed	
Beacon	C	On	
Ignition	C	A/B	
Air Manual	F	Iso valves	On
	F	Packs	Off
	F	Bleeds	Off
Fuel Manual	F	1, 2 & 3 pumps	On
	F	Lwr aux tank pumps	Off
	F	Tail tank trans pumps	Off

AFTER START F

Anti-ice	C	Off/On	
+ APU	F	Off	
+ Elec Panel	F	Checked	
Hyd Manual	F	Aux pumps	Off
	F	1, 2, & 3 L pumps	Off
	F	1, 2, & 3 R pumps	Chk On
	F	1, 2, & 3 L pumps	On
Air Manual	F	Iso valves	Off
	F	Bleed air (all)	On
	F	Packs (all)	On

BEFORE TAKEOFF F

Flaps	F	_____	
	C	Checked	
Spoilers	F	Armed	
Flight controls	CF	Checked	
Trim	F	Zero, Zero, _____	
	C	Checked	
Briefing	C	Complete/Runway ___	
+ Seat belt & harness	All	Fastened	
Hyd Manual	F	1-3, 2-3 RMP's	On
Air Manual	F	Packs (all)	Off
V Speeds	CF	Checked	
EAD	CF	Checked	
Radar	C	Off/On	
FCP	CF	Set	
+ Lights	F	On	

AFTER TAKEOFF PNF

+ Landing gear	PNF	Up & lights out	
+ Flaps & slats	PNF	Up & retract	
+ Spoiler	PNF	Disarm	
+ Autobrakes	PNF	Off	
+ Flap takeoff selector	PNF	15	
+ EAD	PNF	Checked	
+ Hyd Manual	PNF	1-3, 2-3 RMP's	Off
+ Air Manual	PNF	Packs (all)	On
+ Fuel Manual	PNF	L/R aux fuel pmps	Off/On
	PNF	Tail trans pump	Off/On
	PNF	Fill valves (all)	Off/On
	PNF	#2 tank trans pmp	Off/On

FAA Approved MEM FSDO _____
Date _____

Abbildung 2.3.: FedEx MD9711 Norm Checkliste [Blu13]

2.4 Zentrale Begriffe

Checklisten unterscheiden sich im Kontext ihrer Benutzung voneinander. Bei Entwicklungsprojekten (siehe Kapitel 2.3.1) werden andere Funktionalitäten verlangt als bei Fällen in der Medizin (siehe Kapitel 2.3.2). Dieser kontextuelle Bezug wird durch den organisatorischen Rahmen festgelegt. Zentrale Elemente der Checkliste sind die Checklisteneinträge. Um Checklisten und Checklisteneinträge ableiten zu können werden, wie in Kapitel 2.3.1, Vorlagen (Typen) eingeführt. Die aus Typen erstellten Checklisten oder Checklisteneinträge werden Instanzen genannt.

2.4.1 Organisatorischer Rahmen

Der organisatorische Rahmen (OR) definiert den kontextuellen Bezug und die Art der Nutzung einer Checkliste. Beispielsweise kann dies ein Projekt, Fall oder eine einfache Zusammenarbeit zwischen Wissensarbeitern sein. Ein OR kann weitere untergeordnete ORs beinhalten. Dies kann so verstanden werden, dass zum Beispiel ein Großprojekt auch aus vielen Teilprojekten bestehen kann.

2.4.1.1 Organisatorischer Rahmentyp

Der organisatorische Rahmentyp (ORT) ist die generische Vorlage eines OR. Von einem ORT werden eine oder mehrere ORIs abgeleitet. Beim Ableiten werden die Parameter des ORT, von der ORI übernommen und können im Nachhinein angepasst werden. Parameter, die für einige ORIs gleich sind, müssen auf diesem Weg nur einmal im ORT deklariert werden.

2.4.1.2 Organisatorischer Rahmeninstanz

Eine organisatorische Rahmeninstanz (ORI) ist ein konkret instanziierter OR. Eine ORI muss von einem ORT abgeleitet sein.

2.4.2 Checkliste

Eine Checkliste (CL) wird als Schnittstelle zwischen Fachwissen und Projektmanagement interpretiert. Jede CL ist einem OR zugeordnet. In der Regel wird ihr Gebrauch, als Auflistung noch zu erledigender Checklisteneinträge (siehe Kapitel 2.2) verstanden. Checklisten können aus weiteren untergeordneten Checklisten bestehen. Sind alle Einträge und untergeordnete Checklisten abgeschlossen, ist die Checkliste abgeschlossen. In dieser Arbeit wird eine Checkliste als Liste mit Einträgen oder weiteren Checklisten interpretiert (siehe Abbildung 2.4).

2.4.2.1 Checklistentyp

Ein Checklistentyp (CLT) ist eine generische Vorlage einer CL. Von einem CLT werden eine oder mehrere CLIs abgeleitet. Beim Ableiten werden die Parameter des CLT, von der CLI übernommen und können im Nachhinein angepasst werden. Parameter, die für einige CLIs gleich sind, müssen auf diesem Weg nur einmal im CLT deklariert werden.

2.4.2.2 Checklisteninstanz

Eine Checklisteninstanz (CLI) ist eine konkret instanziierte CL. CLI können von einem CLT abgeleitet sein, müssen es jedoch nicht.

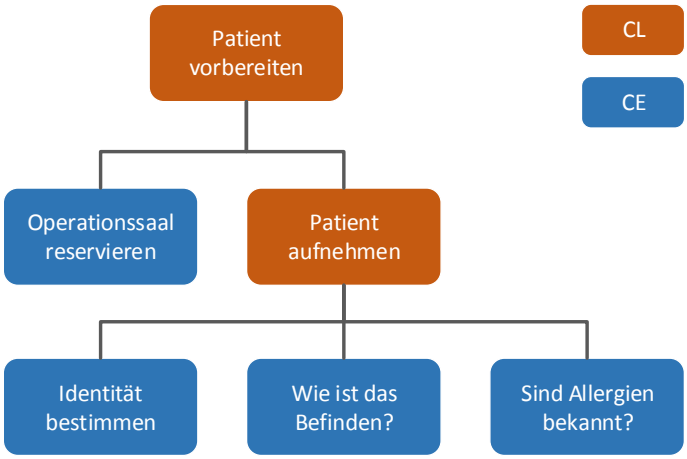


Abbildung 2.4.: CL mit untergeordneter CL und CEs

2.4.3 Checklisteneintrag

Ein Checklisteneintrag (CE) beschreibt eine Aufgabe oder ein Vorgang einer Checkliste. Er besteht aus einer Frage, einer Skala oder eines Satzes, beziehungsweise Schlagwortes (siehe Abbildung 2.5). Um ihn als abgeschlossen zu markieren, muss die Frage beantwortet oder ein Skalenniveau eingetragen werden. Verlangt ein Eintrag keine spezielle Angabe, so muss der Eintrag vom Benutzer explizit als abgeschlossen markiert werden.

2.4.3.1 Checklisteneintragstyp

Ein Checklisteneintragstyp (CET) ist die generische Vorlage eines CE. Von einem CET werden CEIs abgeleitet. Die in einem CET deklarierten Parameter, werden beim instanziieren einer CEI übernommen und können im Nachinein angepasst werden.

2.4.3.2 Checklisteneintragsinstanz

Eine Checklisteneintragsinstanz (CEI) ist ein konkret instanziierter CE. CEI können von einem CET abgeleitet sein, müssen es jedoch nicht.

Abbildung 2.5.: CE mit und ohne Angabe eines Wertes

Kapitel 3

Anforderungen

Im vorigen Kapitel 2 wurden Anwendungsfälle von CLs sowie die zentralen Begrifflichkeiten beschrieben. Zusätzlich sind vielfältige Softwareangebote, mit der Thematik des Checklistenmanagements, erhältlich. Einige von ihnen sollen in Kapitel 3.1 betrachtet und beschrieben werden. Darauf aufbauend werden ausgewählte Anwendererzählungen (siehe Kapitel 3.2) vorgestellt sowie die funktionalen (siehe Kapitel 3.3) wie auch nicht-funktionalen (siehe Kapitel 3.4) Anforderungen gestellt.

3.1 Analyse existierender Software

Software für das Erstellen und Nutzen von Checklisten ist bereits erhältlich. Ein Überblick über die angebotene Funktionalität und den Umfang aktueller Applikationen wird im Folgenden gewährt.

3.1.1 Wunderlist

3.1.1.1 Über die Applikation

Wunderlist ist eine mobile Applikation, die auf dem Smartphone läuft. Außerdem gibt es eine Desktop Variante [wun13]. Für die Nutzung ist eine Verbindung mit dem Internet

und eine Registrierung notwendig. Die Registrierung erfolgt durch Angabe einer E-Mail Adresse oder durch den Internetdienst Facebook.

3.1.1.2 Funktionalität

Checklisten können selbstständig erstellt werden (siehe Abbildung 3.1a) oder mit Freunden geteilt werden. Checklistenvorlagen gibt es nicht. In den einzelnen CLs können CEs angelegt werden (siehe Abbildung 3.1b). Diese können mit einer Erinnerungsfunktion, einer Notiz und untergeordneten CEs versehen werden (siehe Abbildung 3.1c). Des Weiteren kann ein CE als wichtig markiert werden. Voraussetzungen, oder Prioritäten können nicht für einen Eintrag definiert werden. Alle CEs können in ihrer zugehörigen CL umplatziert werden. Ist ein CE erledigt bleibt er halbtransparent erhalten (siehe Abbildung 3.1b), kann jedoch auch entfernt werden. Mit anderen Nutzern geteilte CLs werden über einen Knopf synchronisiert. Das Zusammenführen mehrerer CLs zu einer neuen CL ist nicht möglich.

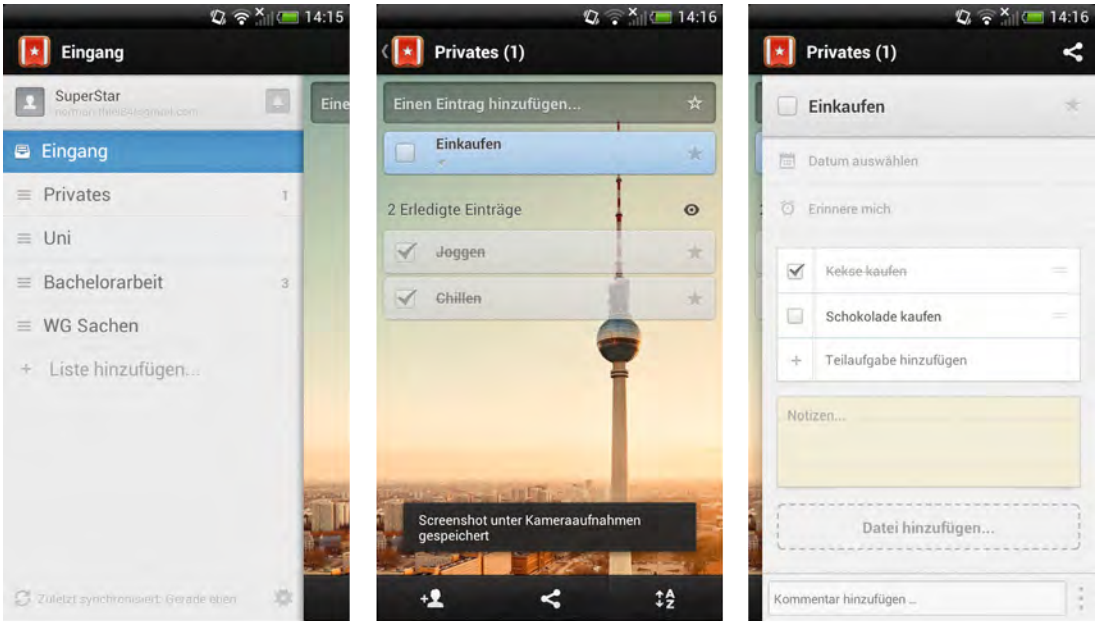
3.1.1.3 Usability

Wunderlist zeichnet sich durch eine gute Bedienbarkeit aus. Die Struktur ist einfach gehalten und nachvollziehbar. Sie besteht im groben aus drei Teilen. Dem CL-Menü, indem alle CL erreichbar sind (siehe Abbildung 3.1a). Ist eine CL ausgewählt werden ihre Informationen angezeigt (siehe Abbildung 3.1b). Wird ein Eintrag ausgewählt können diesem zusätzliche Informationen hinzugefügt werden (siehe Abbildung 3.1c). Diese einfach Darstellung und klare Struktur ermöglicht eine schnelle Navigation durch die Applikation. Durch die Wahl des Hintergrundes kann der Nutzer die Applikation persönlicher Gestalten.

3.1.1.4 Fazit

Wunderlist ist hervorragend zur Nutzung als Aufgabenliste geeignet. Es können schnell neue CEs hinzugefügt und abgehakt werden. Das Teilen mit Freunden unterstützt zum

Beispiel das Planen von Gruppenevents. Die durchweg einfach gehaltene Struktur ermöglicht einen schnellen Einstieg. Die Gestaltung ist schlicht und ansehnlich. Zur Untersützung von Wissensarbeiten eignet sie sich jedoch nicht. Checklisten sind nur zum abhaken gedacht. Das formulieren von CEs, die das Eintragen eines Wertes erfordern ist nicht möglich. Weiterhin ist es nicht möglich einer CL weitere CLs hinzuzufügen. Somit ist das Erstellen einer CL mit Baumstruktur (siehe Abbildung 2.1) nicht möglich Beim Teilen mit anderen können keine Rechte vergeben werden. Jeder kann jederzeit alles ändern.



a) b) c)

Abbildung 3.1.: Screenshots der mobilen Applikation Wunderlist

3.1.2 4Checkers

3.1.2.1 Über die Applikation

4Checkers ist eine mobile Applikation, die auf dem Smartphone läuft. Desweiteren gibt es eine Desktop Variante [4Ch13]. Für die Nutzung ist eine Verbindung mit dem Internet

und eine Registrierung notwendig. Die Registrierung bei 4Checkers erfolgt durch Angabe einer E-Mail Adresse oder durch die Internetdienste Facebook oder Google Plus.

3.1.2.2 Funktionalität

Es können eigene Cls erstellt oder sogar Vorlagen, das heißt CLT, verwendet werden (siehe Abbildung 3.2a). Auch erstellte CLs anderer Nutzer können verwendet werden, sofern diese von den Nutzern veröffentlicht wurden. CL können weitere CL enthalten und besitzen somit die Struktur eines Aufgabenbaumes (siehe Abbildung 2.1). Verwendet man einen CLT oder eine CL eines anderen Nutzers, so können neue CEIs hinzugefügt oder existierende entfernt werden (siehe Abbildung 3.2b). Beim Erstellen eigener CEIs kann ein Datum zugewiesen, Notizen hinzugefügt und Prioritäten gesetzt werden (siehe Abbildung 3.2c). Voraussetzungen einzelner CEIs können nicht definiert werden. Des Weiteren lassen sich Helfer (andere Nutzer) zuweisen und somit CEIs delegieren. Die Möglichkeit mehrere CLs zu vereinen gibt es nicht.

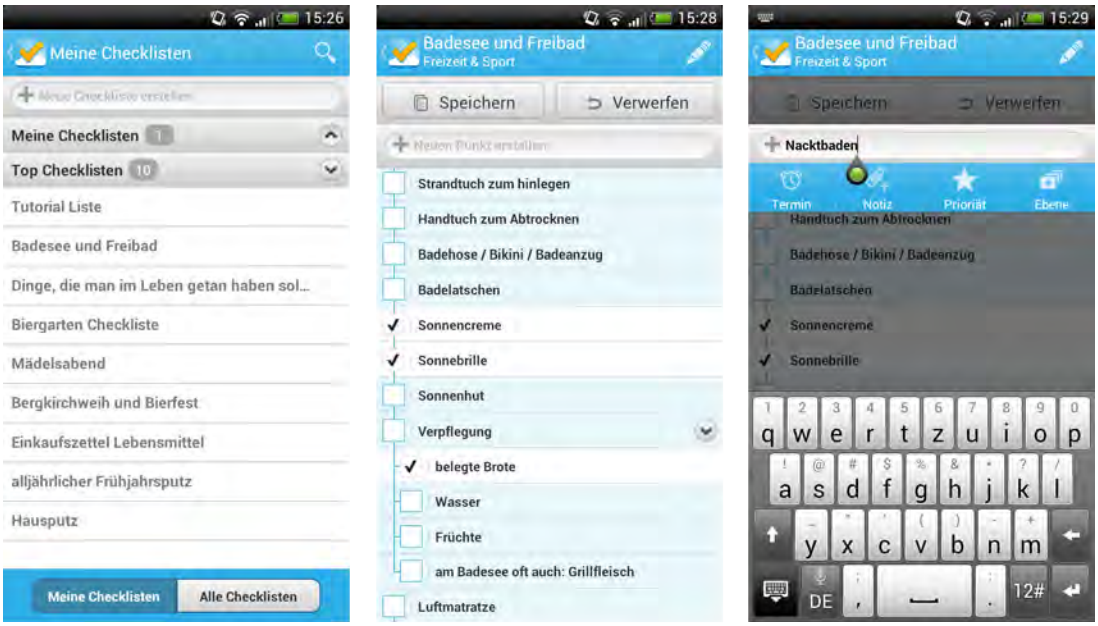
3.1.2.3 Usability

Die Gestaltung der Benutzeroberfläche von 4Checkers ist nicht immer nachvollziehbar. Funktionen sind nicht dort angebracht wo man sie erwartet. Zum Beispiel können CE nach der Auswahl nicht entfernt werden (siehe Abbildung 3.2c). Auch ein herauswischen, wie in anderen Applikationen durchaus üblich, entfernt den CE nicht. Auf dem Bildschrim werden zu viele Elemente angezeigt (siehe Abbildung 3.2a und b). Es ist anstrengend die Übersicht zu behalten. Eine Möglichkeit 4Checkers individuell und persönlicher zu gestalten gibt es nicht.

3.1.2.4 Fazit

4Checkers bietet umfangreiche Funktionen für CL an. Zum Beispiel können CLT genutzt, Prioritäten gesetzt oder Termine zugewiesen werden. Jedoch ist die Bedienung nicht immer nachvollziehbar und klar. Zur Untersützung von Wissensarbeiten eignet sich

4Checkers nicht. CEs sind nur zum abhaken gedacht. Spezifische Checkfragen können nicht formuliert werden. Beim Zuweisen von Helfern können keine Rechte vergeben werden.



a) b) c)

Abbildung 3.2.: Screenshots der mobilen Applikation 4Checkers

3.1.3 Google Keep

3.1.3.1 Über die Applikation

Google Keep ist eine mobile Applikation, die auf dem Smartphone läuft. Die Desktop Variante ist über ein Google Konto erreichbar [Goo13a]. Für die Nutzung ist eine Verbindung mit dem Internet und das Anlegen eines Google Kontos, zu dem eine Google Mail Adresse gehört, notwendig. Die Registrierung bei Google Keep erfolgt durch Angabe einer Google Mail Adresse.

3.1.3.2 Funktionalität

Ist man auf der Startseite, können Notizen hinzugefügt werden (siehe Abbildung 3.3b). Diese Notizen können aber auch Listenpunkte beinhalten und somit einfache CLs repräsentieren (siehe Abbildung 3.3a). Erledigte CEs bleiben ausgegraut und durchgestrichen erhalten (siehe Abbildung 3.3c). Bedingungen und Voraussetzungen lassen sich nicht angeben. Weiterhin kann eine Notiz auch eingesprochen werden oder ein Bild sein (siehe Abbildung 3.3b). Alle Notizen können gefärbt und verschoben werden und dementsprechend lassen sich Prioritäten darstellen (siehe Abbildung 3.3b). Eine Erinnerungsfunktion steht ebenfalls zur Verfügung (siehe Abbildung 3.3a). Ist ein CE erledigt, so kann dieser entfernt oder archiviert werden. Archivierte CEs können eingesehen (siehe Abbildung 3.3c) und wieder aktiviert werden. Das Hinzufügen und Bearbeiten durch Helfer ist nicht möglich. Das Teilen von CEs ist nur bedingt möglich, da ein Export nicht verfügbar ist und die Informationen nur textuell oder als Bild per E-Mail oder andere Dienste verschickt werden können.

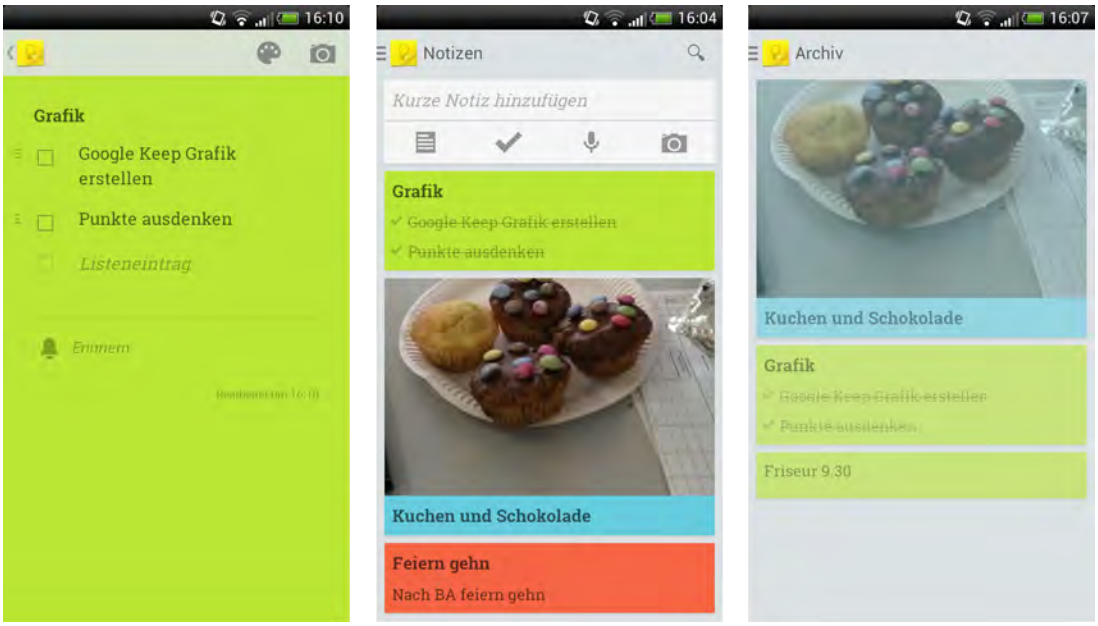
3.1.3.3 Usability

Das Augenmerk bei der Darstellung von Google Keep liegt in der Einfachheit und Übersichtlichkeit. Mit verschiedenen Ansichten wird gespart. Es gibt eine Startansicht, in der alle Notizen aufgelistet werden, eine Ansicht beim Erstellen oder Bearbeiten einer Notiz und eine Ansicht für archivierte Notizen. Diese Einfachheit gewährt eine jederzeit nachvollziehbare Navigation durch die Benutzeroberfläche. Die Notizen werden schon in der Startansicht großflächig präsentiert, so dass zumeist alle Listeneinträge sichtbar sind. Wischgesten zum archivieren der Notizen, steigern weiter die Bedienbarkeit. Eine persönliche Gestaltung von Google Keep ist nicht möglich.

3.1.3.4 Fazit

Google Keep ist, für den Gebrauch als Aufgabenliste, sehr gut umgesetzt. Notizen können, wie eigenständige ToDo-Listen, genutzt werden. Die Einfachheit der Darstellung

und die nachvollziehbare Struktur der Applikation, ermöglichen einen schnellen Einstieg und eine leichte Bedienung. Zur Untersützung von Wissensarbeiten eignet sie sich jedoch nicht. Das Erstellen einer CL mit weiteren CLs oder CEs, die spezifische Fragen enthalten, ist nicht möglich. Auch das Teilen von Notizen ist nicht möglich.



a) b) c)

Abbildung 3.3.: Screenshots der mobilen Applikation Google Keep

3.1.4 Joomla

3.1.4.1 Über die Applikation

Web-Content-Management-Systeme sind Systeme zur Erstellung und Verwaltung von Seiteninhalte, in Webseiten. Diese Nutzung kommt der Administrationsumgebung des proCollab Prototypen nahe. Deshalb wird es als Vertreter administrativer Software betrachtet. Joomla ist ein freies Web-Content-Management-System und basiert auf der Programmiersprache PHP [Joo13].

3.1.4.2 Funktionalität

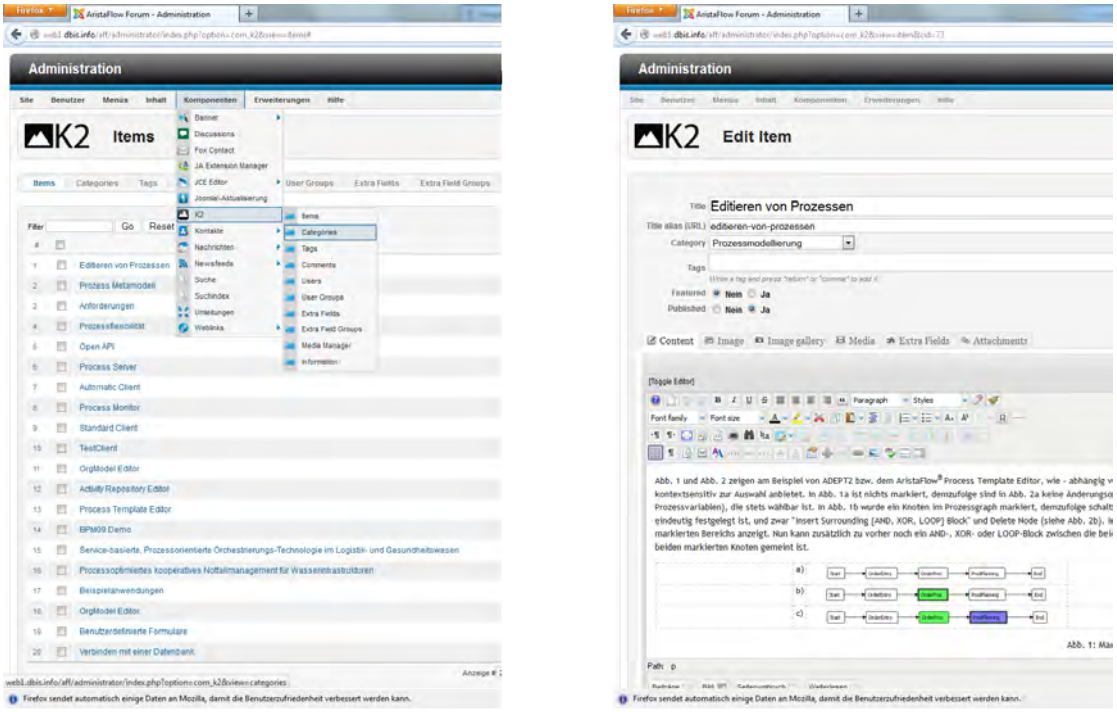
Mit einer Administrationsumgebung, die sich über den Browser aufrufen lässt, werden Inhalte verwaltet und bearbeitet (siehe Abbildung 3.4b). Eine Anmeldung in der Administrationsumgebung ist notwendig. Administratoren können weitere Benutzer hinzufügen, existierende entfernen oder Zugriffs- und Schreibrechte festlegen. Mit einem Dropdown-Menü werden alle Seiteninhalte erreicht und können verändert oder entfernt werden (siehe Abbildung 3.4a). Die Gestaltung der Webseiten erfolgt durch Templates. Diese können nach Belieben durch den Administrator angepasst werden und ermöglichen ein einheitliches optisches Erscheinungsbild der Webseite

3.1.4.3 Usability

Die Navigation in der Benutzeroberfläche erfolgt über ein Menü in der Kopfleiste, die jederzeit sichtbar ist (siehe Abbildung 3.4a). Das Menü ist nach den kontextuellen Elementen sortiert und schafft eine Übersicht über alle Funktionen. Unter der Kopfleiste werden die gewählten Inhalte präsentiert und können bearbeitet werden(siehe Abbildung 3.4a). Durch das geordnete und immer erreichbare Menü kann, trotz hoher Komplexität der Anwendung, leicht durch die Benutzeroberfläche navigiert werden. Die Gestaltung ist schlicht.

3.1.4.4 Fazit

Die Gestaltung der Administrationsumgebung ist in diesem Layout in vielen administrativen Softwareumgebungen zu finden. Sie bietet Übersicht und Ordnung über alle Systemfunktionen. Da bei der Entwicklung neuer Software auch für die optimale Gestaltung hohe Entwicklungskosten und großer Aufwand betrieben werden, ist die Gestaltung als Best Practice zu betrachten.



a) b)

Abbildung 3.4.: Screenshot der Administrationsumgebung von Joomla

3.2 Anwendererzählungen

Auf Basis der Analyse der Anwendungsfälle (siehe Kapitel 2.3) und der existierenden Software (siehe Kapitel 3.1) soll nun eine Auswahl an Anwendererzählungen vorgestellt werden. Die vollständige Auflistung aller Anwendererzählungen ist, aus Gründen der Übersichtlichkeit im Anhang A zu finden.

3.2.1 Grundannahmen

Jeder Nutzer verfügt über ein Benutzerkonto und muss sich über dieses anmelden. Jedes Benutzerkonto kann durch den Administrator eingesehen und verändert werden.

Daher ist es naheliegend die Anmeldung am System und Änderungen am Benutzerkonto zu betrachten. Der proCollab ermöglicht das Benutzen von Vorlagen. Deshalb wird die instanziierung einer CLI, aus einem CLT, gewählt. Weiterhin können ORTs, CLTs und CETs nur in der Administrationsumgebung erstellt werden. Deshalb wird das Anlegen eines ORT und anpassen eines CLT betrachtet. Ebenso elementar ist das Abschließen einer CEI innerhalb der CLI.

3.2.2 Einfache Anmeldung (BV2)

Ein Nutzer des proCollab Prototypen gibt seine E-Mail Adresse und Passwort ein. Anschließend werden seine Eingaben vom Server überprüft. Bei einer Bestätigung wird ihm das Backend angezeigt, bis er sich vom System abmeldet. Bei Ablehnung kann er erneut seine Eingaben tätigen (siehe Abbildung 3.5).

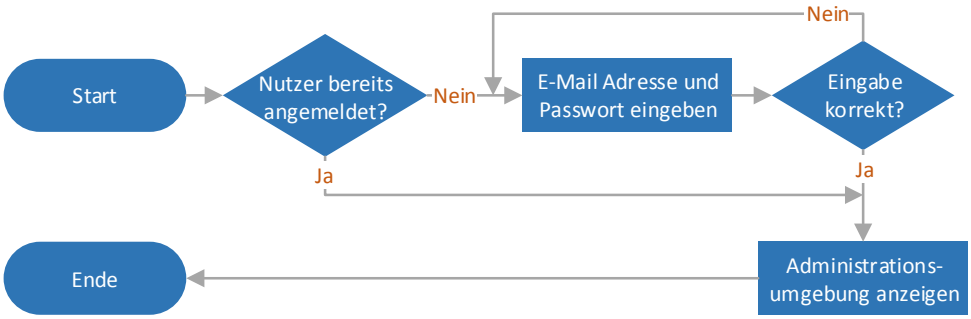


Abbildung 3.5.: Flussdiagramm der Anmeldung

3.2.3 Benutzerdaten anpassen (BV3)

Ein Administrator kann die Daten eines Nutzers, in der administrativen Umgebung, ändern. Dazu kann er den Vorname, Nachnamen, Passwort, E-Mail Adresse und die Organisation des Nutzers ändern. Wird die E-Mail Adresse verändert, muss diese erneut

vom Nutzer bestätigt werden. Weiterhin kann ein Administrator den Nutzer einer ORI hinzufügen oder ihn von einer entfernen. Beim Hinzufügen muss die Rolle angegeben werden. Ein Administrator kann einen anderen Nutzer zum Administrator ernennen oder ihm dieses Recht nehmen (siehe Abbildung 3.6).

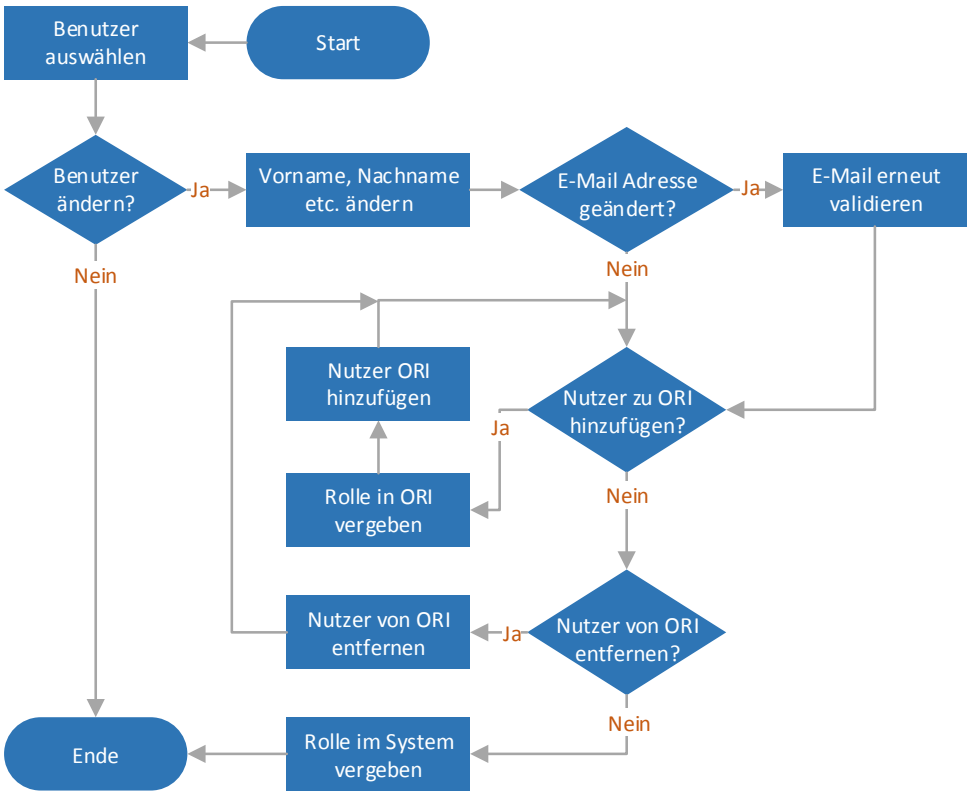


Abbildung 3.6.: Flussdiagramm der Anpassung von Benutzerdaten

3.2.4 Instanziieren einer CLI (CLI2)

Ein Administrator kann, beim Erweitern einer ORI, neue CLI anhand eines existierenden CLT instanziieren. Hierzu muss ein CLT gesucht werden. Die Details des CLT werden angezeigt und können angepasst werden. Mit einem Knopf kann der CLT zu einer CLI instanziiert werden. Diese wird dann der ORI hinzugefügt (siehe Abbildung 3.7).

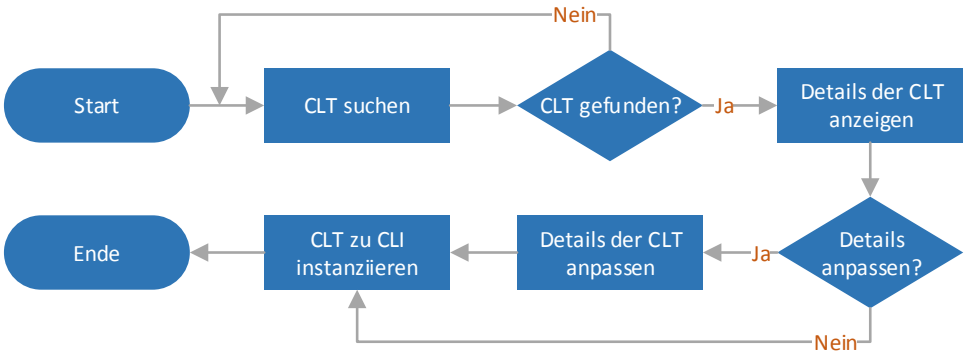


Abbildung 3.7.: Flussdiagramm der Instanziierung einer CLI

3.2.5 Erstellen eines neuen ORT (ORT1)

Ein Administrator kann in der administrativen Umgebung einen ORT erstellen. Dazu muss er einen Rahmennamen angeben und kann zusätzlich eine Beschreibung, Ziel, Startdatum, Enddatum und die Bearbeitungszeit in Tagen angeben. Die Bearbeitungszeit repräsentiert die Zeit, die für die Erfüllung der ORI nach dem Start zur Verfügung stehen. Zusätzlich kann ein ORT einem anderen ORT untergeordnet werden (siehe Abbildung 3.8).

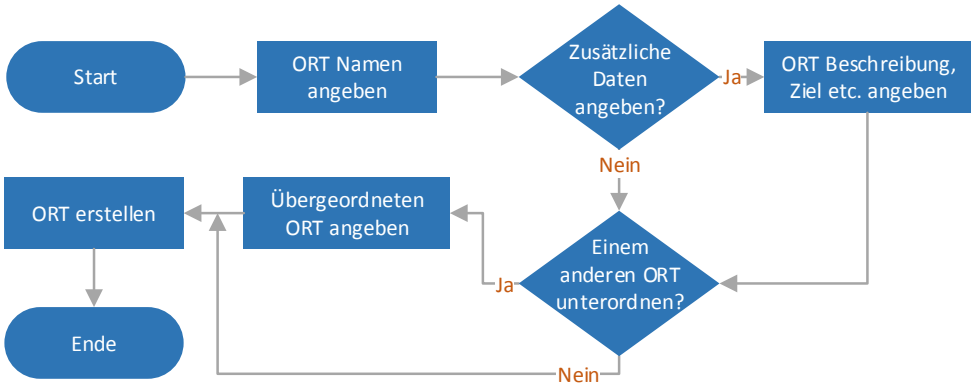


Abbildung 3.8.: Flussdiagramm der Erstellung eines ORT

3.2.6 Anpassen eines CLT (CLT2)

Nach der Suchen und Auswahl eines CLT kann ein Administrator den CLT anpassen. Er kann den Namen, die Beschreibung und den Status verändern.

3.2.7 Abschluss einer CEI (CEI6)

Nach der Auswahl einer CEI, kann ein Administrator diese mit einem Klick abschließen.

3.3 Funktionale Anforderungen

Funktionale Anforderungen beschreiben, welche Funktionalitäten ein System besitzen soll [RR06]. Basierend auf die Anwendererzählungen der Administrationsumgebung des proCollab Prototypen, werden im Folgenden die funktionalen Anforderungen erhoben. Die Systemfunktionen der administrativen Umgebung werden in die Kategorien der Benutzerverwaltung, Vorlagenverwaltung und Instanzverwaltung unterteilt.

3.3.1 Benutzerverwaltung

Jeder Nutzer verfügt über ein Benutzerkonto. Dieses ist notwendig, um sich am System anzumelden (siehe A.1.1, BV2). Die administrative Umgebung ist ausschließlich als Administrator betretbar. Dieser kann dort weitere Nutzer erstellen (siehe A.1.1, BV1) oder existierende Nutzer vom System entfernen (siehe A.1.1, BV4). Weiterhin kann er einen Nutzer suchen (siehe A.1.1, BV5) und seine Benutzerdaten verändern (siehe A.1.1, BV3).

3.3.2 Vorlagenverwaltung

Ist ein Administrator am System angemeldet, kann er einen CET erstellen (siehe A.1.4, CET1) oder entfernen (siehe A.1.4, CET3). Bereits existierende CETs können gesucht (siehe A.1.4, CET4) und anschließend verändert werden (siehe A.1.4, CET2).
Ebenfalls können CLT erstellt (siehe A.1.6, CLT1) oder entfernt (siehe A.1.6, CLT3) werden. Vorhandene CLT können gesucht (siehe A.1.6, CLT4) und verändert (siehe A.1.6, CLT2) oder weiteren CLT hinzugefügt werden(siehe A.1.6, CLT6). Ein CET innerhalb eines CLT kann umpositioniert werden (siehe A.1.6, CLT5).
Weiterhin kann ein ORT erstellt (siehe A.1.2, ORT1) oder entfernt werden (siehe A.1.2, ORT5). Der Erstellung kann die Ableitung eines vorhandenen ORT (siehe A.1.2, ORT2) oder ORI vorausgehen (siehe A.1.2, ORT3). Ein ORT kann über die Suchfunktion gesucht (siehe A.1.2, ORT6) und anschließend verändert werden (siehe A.1.2, ORT4).

3.3.3 Instanzverwaltung

Ein angemeldeter Administrator kann eine CEI erstellen (siehe A.1.5, CEI2), instanziieren (siehe A.1.5, CEI1) oder entfernen (siehe A.1.5, CEI4). Bereits existierende CEIs können gesucht (siehe A.1.5, CEI5) und anschließend verändert werden (siehe A.1.5, CEI3). Alle CEIs können abgeschlossen werden (siehe A.1.5, CEI6).
Ebenfalls können CLIs erstellt (siehe A.1.7, CLI1), instanziiert (siehe A.1.7, CLI2) oder entfernt (siehe A.1.7, CLI5) werden. Vorhandene CLIs können gesucht (siehe A.1.7,

CLI6), verändert (siehe A.1.7, CLI4) oder weiteren CLIs hinzugefügt werden(siehe A.1.7, CLI8). Alle CLIs können abgeschlossen werden (siehe A.1.7, CLI3). Ein CEI innerhalb einer CLI kann umpositioniert werden (siehe A.1.7, CLI7).
ORIs müssen instanziiert werden (siehe A.1.3, ORI1). Vorhandene ORIs können abgeschlossen (siehe A.1.3, ORI4) oder entfernt werden (siehe A.1.3, ORI3). Eine ORI kann über die Suchfunktion gesucht (siehe A.1.3, ORI5) und anschließend verändert werden (siehe A.1.3, ORI2).

3.4 Nicht-funktionale Anforderungen

Nicht-funktionale Anforderungen geben Auskunft darüber, wie sich ein System bei der Benutzung verhalten soll [RR06]. Diese werden im Folgenden, für die administrative Umgebung des proCollab Prototypen, festgelegt und sind dabei in die Kategorien Bedienbarkeit, Erreichbarkeit, Performanz und Sicherheit gegliedert.

3.4.1 Bedienbarkeit

Alle Anzeigekomponenten der administrativen Umgebung werden übersichtlich gestaltet. Komponenten, die miteinander im Zusammenhang stehen, werden durch räumliche Nähe angeordnet. Beispielsweise sollen Benutzerdetails neben der Benutzerauswahl erscheinen, der Anmelde-Button unter oder neben den Eingabefeldern.
Des Weiteren soll die Strukturierung der Anzeigekomponenten nachvollziehbar sein. Schon bevor der Nutzer auf ein Menüpunkt klickt, hat er eine Vorstellung davon, was sich dahinter verbirgt. Deckt sich diese Vorstellung mit dem Resultat, war dieser Menüpunkt für den Nutzer intuitiv bedienbar. Deckt sie sich nicht, wurde die Reaktion des Systems nicht korrekt wahrgenommen. Dies wird *gulf of evaluation* genannt [Nor02].
Eine erhöhte Bedienbarkeit wird unter anderem auch durch eine geringe Komplexität der Anzeige erreicht. Hinter dem Begriff der Komplexität einer Anzeige, verbirgt sich vor allem der Begriff des *extraneous cognitive load*, aus der *Cognitive Load Theory* [SAK11] der Psychologie. Der extraneous cognitive load, beschreibt die extrinsische kognitive

Belastung, die ausschließlich durch die Wahl der Darstellung von Informationen entsteht. Dieser sollte möglichst gering gehalten werden.

3.4.2 Erreichbarkeit

Die administrative Umgebung ist speziell für die Nutzung im Browser entwickelt und unterstützt vorerst keine weiteren Endgeräte. Sie soll in jedem Browser, unabhängig vom Standort des Endgerätes, aufgerufen werden können. Dies bedeutet im Besonderen, dass sie nicht auf einen lokalen Server beschränkt werden soll.

3.4.3 Performanz

Das System wird überwiegend für die Erstellung, Bearbeitung und Ausführung von CLs benutzt. Hierbei wird der Nutzer schnell und häufig mit der Benutzeroberfläche interagieren. Hohe Interaktionsraten werden verlangt. Deshalb ist es notwendig einen schnellen Datenaustausch und damit kurze Übertragungszeiten der Informationen, sowie kurze Seitenladezeiten zu erzielen.

3.4.4 Sicherheit

Da es sich um eine administrative Umgebung handelt, kann das System nur als Administrator betreten werden. Findet ein unsachgemäßer Gebrauch eines Nutzers statt, muss ein Administrator befugt sein den Nutzer vom System zu entfernen.

Kapitel 4

Konzeption

Anhand der in Kapitel 3 definierten Anforderungen wird in diesem Kapitel ein Softwarekonzept erarbeitet. Es werden zuerst wichtige Programmierparadigmen genannt, die die Umsetzung unterstützen (siehe Kapitel 4.1). Im Weiteren wird das Datenmodell beschrieben (siehe Kapitel 4.2). Anschließend wird auf die Architektur des proCollab Prototypen eingegangen (siehe Kapitel 4.3) und die Architektur der administrativen Umgebung eingeordnet (siehe Kapitel 4.4). Weiterhin wird die Gestaltung der Benutzeroberfläche konzipiert und durch MockUps (siehe Kapitel 4.6) visualisiert.

4.1 Programmierparadigmen

Um die Administrationsumgebung des proCollab Prototypen sauber zu implementieren, wird auf gängige Programmierparadigmen zurückgegriffen. Als Programmierparadigma wird das einer Programmiertechnik zugrunde liegende Prinzip bezeichnet [CF06].

4.1.1 Model View Controller

Das *Model View Controller* Konzept ist ein Architekturmuster bei der Implementierung einer Software [Bus96]. Dabei wird die Software in die Komponenten *Model*, *View* und *Controller* strukturiert. Durch diese Trennung wird eine hohe Skalierbarkeit erreicht, da

die Komponenten voneinander unabhängig sind und ohne Weiteres erweitert oder verändert werden können. Auch eine Kapselung der Daten außerhalb der Komponenten wird erreicht, da die Komponenten keine Informationen über die spezielle Implementierung der jeweils anderen Komponenten besitzen. Die View ist für die grafische Darstellung verantwortlich. Das Model ist für die Berechnung, Verwaltung und Verarbeitung von Daten zuständig. Der Controller nimmt Interaktionen der View entgegen, wertet sie aus, leitet sie an das Model weiter und schickt die berechneten Daten des Models wieder an die View zurück (siehe Abbildung 4.1). Die Architektur des proCollab Prototypen (siehe Kapitel 1.3) wie auch die Architektur der administrativen Umgebung unterliegt diesem Konzept.

Abbildung 4.1.: Das Model View Controller Konzept nach [Bus96]

4.1.2 Datenkapselung

Bei der Datenkapselung werden Objekte so implementiert, dass Details über den inneren Aufbau verborgen werden und Änderungen von Objektzuständen nur über dafür vorgesehene Methoden, zum Beispiel *Getter*- und *Setter* Methoden, erfolgen können [Bus96]. Ein direkter Zugriff auf die Objektattribute ist nicht möglich. Mit der Wahl der Zugriffsmethoden eines Attributes wird somit festgelegt, ob ein Attribut nur gelesen oder auch geschrieben werden kann. Da administrative Umgebungen vorwiegend auch Schreibrechte gewähren, ist vor allem hier nicht auf diese Technik zu verzichten, um ungewollte Zugriffe zu erschweren. Weiterhin fördert dies auch die Modularität und Wiederverwendbarkeit des Programmcodes.

4.1.3 Wiederverwendbarkeit

Um einen bestimmten Programmcode mehrmalig zu benutzen, soll dieser nicht dupliziert sondern wiederverwendet werden. Dies ermöglicht das Strukturieren und gering halten des Sourcecodes einer Anwendung. Hierfür stehen dem Entwickler Konzepte, wie Vererbung und Komposition zur Verfügung. Das Erstellen und Verwenden von Formularen, setzt beispielsweise diese Technik um.

4.2 Datenmodell

Für die Repräsentation der kontextuellen Elemente des proCollab Prototypen wurde ein Datenmodell ausgearbeitet. Aus Gründen der Übersicht wird im Folgenden nur der Aufbau der Entitäten, die einen OR, CL oder CE repräsentieren, beschrieben und durch ein Klassendiagramm visualisiert. Die Klassendiagramme der übrigen Entitäten sind im Anhang A.1.8 zu finden. Die Entitäten des Datenmodells werden zur Kommunikation, unter den verschiedenen Schichten des proCollab Prototypen (siehe Abbildung 1.1), genutzt.

4.2.1 Aufbau von Instanzen und Typen

Der Aufbau der Instanzelemente ORI, CLI und CEI unterliegt den in Kapitel 2.4 beschriebenen Grundlagen. Eine ORI kann aus CLIs oder weiteren ORIs bestehen. Da CLIs einer Baumstruktur unterliegen können sie weitere CLIs oder mehrere CEIs enthalten. CEIs entsprechen den Blattknoten des Baumes und können keine weiteren Elemente enthalten. Analog dazu ist der Aufbau der Typenelemente ORT, CLT und CET. Abbildung 4.2 veranschaulicht den Aufbau.

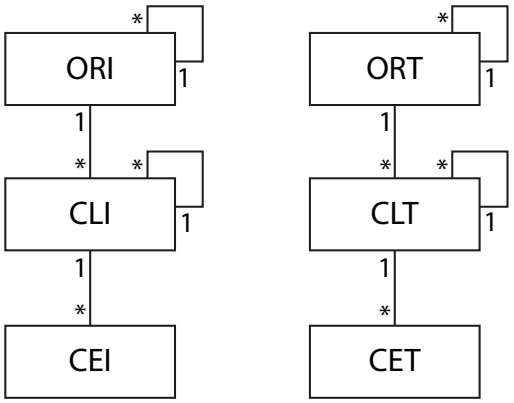


Abbildung 4.2.: Aufbau der Instanzen und Typen

4.2.2 Repräsentation des OR, ORT und der ORI

Die abstrakte Klasse *FrameElement* repräsentiert den OR. Ein FrameElement kann nicht instanziiert werden. Es weist alle Attribute, die sich ORT und ORI teilen, aus. Das sind im Besonderen die Attribute, die beim Instanziieren einer ORI (siehe Kapitel A.1.3, ORI1) von dem ORT übernommen werden. Die Klassen *CFrameType* und *CFrameInstance* erben von FrameElement. CFrameType stellt den ORT und CFrameInstance die ORI dar. Das Klassendiagramm von OR, ORT und ORI ist in Abbildung 4.3 zu sehen.

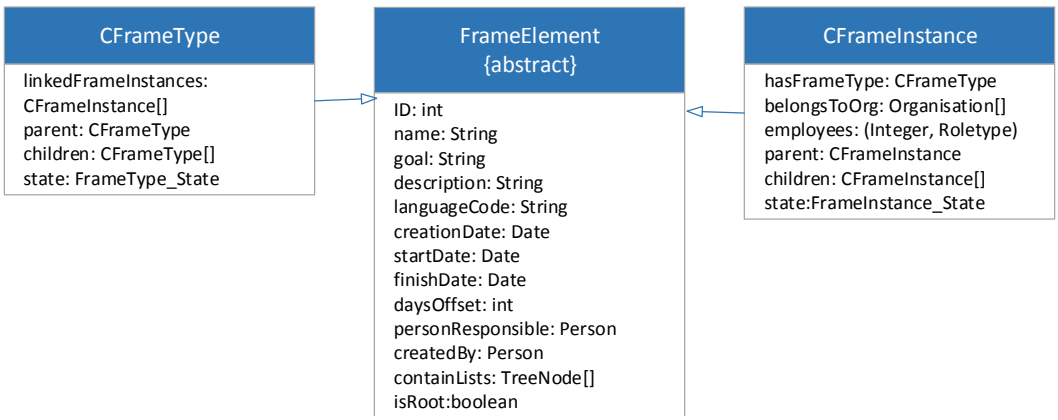


Abbildung 4.3.: UML-Klassendiagramm von OR, ORT und ORI

4.2.3 Repräsentation der CL, des CLT und der CLI

Im proCollab Prototypen besitzen CLs eine Baumstruktur. Ein Knoten des Baumes kann entweder ein Blattknoten oder ein weiterer Baumknoten sein. Baumknoten sind CLs und werden durch die abstrakte Klasse *TreeNode* repräsentiert. Die abstrakte Klasse *TreeElement* beinhaltet alle Attribute sich TreeNode und *TreeLeaf* (siehe Kapitel 4.2.4) teilen. Die abstrakte Klasse TreeNode erbt von TreeElement und weist wiederum alle Attribute, die sich CLT und CLI teilen, aus. Das sind die Attribute, die beim Instanziieren einer CLI (siehe Kapitel A.1.7, CLI2) von dem CLT übernommen werden. Die Klassen *CListType* und *CListInstance* erben von TreeNode. Im Datenmodell stellt ein CListType den CLT und eine CListInstance die CLI dar. Das Klassendiagramm von CL, CLT und CLI ist in Abbildung 4.4 zu sehen.

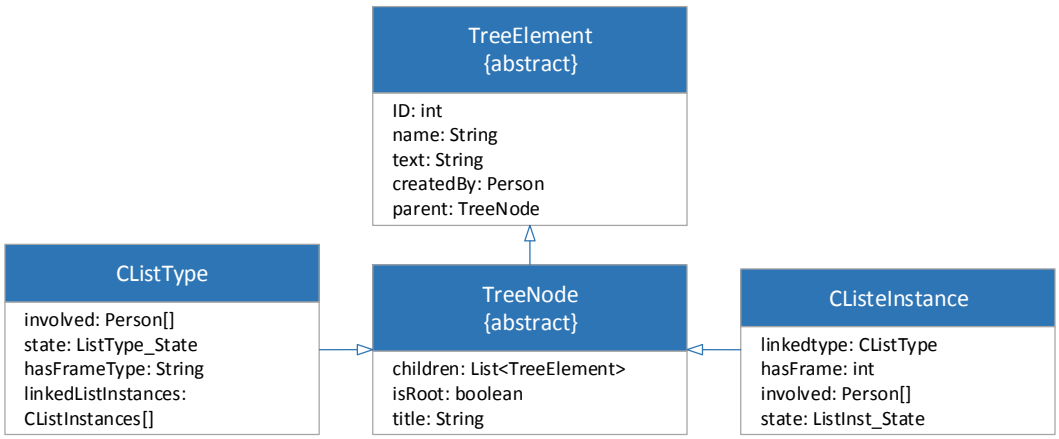


Abbildung 4.4.: UML-Klassendiagramm von CL, CLT und CLI

4.2.4 Repräsentation des CE, CET und der CEI

CEs werden durch die abstrakte Klasse *TreeLeaf* repräsentiert, die von TreeElement (siehe Kapitel 4.2.3) erbt.*CItemType* und *CItemInstance* erben von TreeLeaf. *CItemType* repräsentiert ein CET und *CItemInstance* eine CEI. Das Klassendiagramm von CE, CET

und CEI ist in Abbildung 4.5 zu sehen. Das TreeElement ist vereinfacht dargestellt und in Abbildung 4.4 vollständig sichtbar.

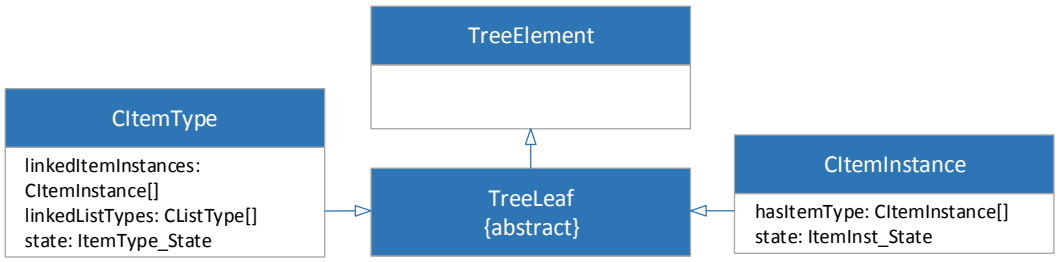


Abbildung 4.5.: UML-Klassendiagramm von CE, CET und CEI

4.3 Systemarchitektur

Da die Entwicklung einer administrativen Umgebung ein Teilaspekt des proCollab Prototypen ist, wird zuerst die Systemarchitektur beschrieben, die die Architektur der Software in die Schichten Persistenzschicht, Anwendungsschicht, Kommunikationsschicht und Präsentationsschicht unterteilt (siehe Abbildung 1.1). Diese Schichten werden unabhängig voneinander implementiert und kommunizieren nur über definierte Schnittstellen miteinander. Die modulare Struktur gewährleistet hohe Wartbarkeit, Erweiterbarkeit und Austauschbarkeit der einzelnen Softwarekomponenten. Genügt beispielsweise eine Komponente nicht mehr den Anforderungen oder die Technologie ist veraltet, so kann diese problemlos ersetzt werden. Die neue Implementierung muss lediglich an die definierten Schnittstellen angebunden werden [HC01].
Die Persistenzschicht ist verantwortlich für die Bereitstellung und Beständigkeit der Daten. Die Anwendungsschicht steht für die Verwaltung und Bearbeitung der Systemelemente. Sie stellt etwa die Methodik bereit, um CL (siehe Abbildung 2.1) zu bearbeiten. Um eine hohe Verfügbarkeit der Software zu gewährleisten, werden Anzeigekomponenten für das Smartphone, Tablet und den Browser implementiert. Diese Komponenten sind Teil der Präsentationsschicht. Die Lieferungsschicht ist für die Kommunikation der

einzelnen Anzeigekomponenten mit der Persistenzschicht verantwortlich. Sie ist außerdem für die Aufbereitung der Daten, für die jeweils unterschiedlichen Implementierung, zuständig.

4.4 Architektur der Administrationsumgebung

Die zugrunde liegende Architektur der Administrationsumgebung, wird wie die Systemarchitektur des proCollab Prototypen (siehe Kapitel 4.3), modular gestaltet. Die Architekturkomponenten werden nach ihrer Funktion unterteilt und im Folgenden beschrieben.

4.4.1 Ansicht

Die Aufgabe einer Ansicht ist es, vorher eingebundene Inhalte im Browser anzuzeigen, sobald diese geladen wird. In Web-Applikationen wird häufig beim Klick auf einen Menüpunkt oder eine Komponente die Seite im Browser neu geladen. Die teils hohen Seitenladezeiten vermindern den Komfort im Umgang mit der Applikation und bremsen die Interaktionen des Nutzers aus. Im Gegensatz dazu stehen Desktop Anwendungen. Da diese nicht im Browser laufen, haben sie keine Seitenladezeiten. In der Administrationsumgebung soll der Komfort und die Interaktionsmöglichkeiten einer Desktop Anwendung als Vorbild dienen. Da es sich hierbei um eine Web-Applikationen handelt, ist ein Neuladen der Seite nicht gänzlich zu umgehen, dies soll aber nur sehr reduziert geschehen. Um dieses Ziel zu erreichen, wird es nur für jede Funktionalität eine Ansicht geben. Demnach wird es beispielsweise eine Ansicht für die Benutzerverwaltung, eine für die Vorlagenverwaltung und eine für die Instanzverwaltung geben. Mit dem Wechsel der Funktionalität durch den Nutzer wird die entsprechende Ansicht geladen (siehe Abbildung 4.6).

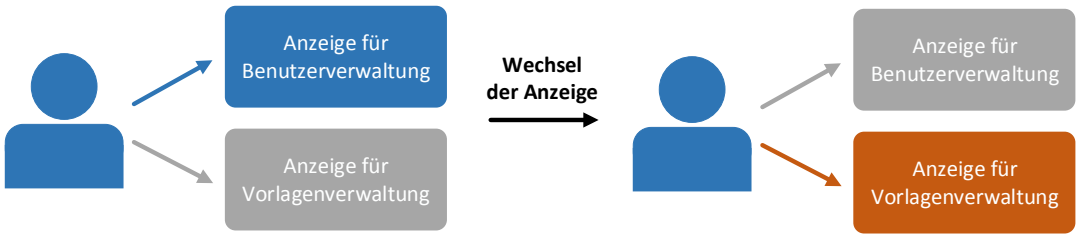


Abbildung 4.6.: Wechsel der Funktionalität und damit auch der Ansicht

4.4.2 Formulare

Häufig verwendete Elemente einer Ansicht, beispielsweise Tabellen, werden in eigenständigen Komponenten, genannt Formulare, zusammengefasst. Methoden zur Verwaltung des Inhalts stellt das jeweilige Formular zur Verfügung, so dass außerhalb des Formulars keine Kenntnis über die Implementierung nötig ist. Sind Erweiterungen oder Änderungen notwendig, so müssen diese nur einmalig vorgenommen werden und gelten für die ganze Architektur. Ein Formular kann als Baustein im System aufgefasst werden. Diese Bausteine werden zur Nutzung auf den Ansichten platziert (siehe Abbildung 4.7). Ein dynamischer Wechsel des Inhaltes ist möglich, indem alle oder nur vereinzelte Formulare ausgetauscht werden, ähnlich einem Lego Stecksystem. Damit kann ohne ein erneutes Laden der Seite, der Inhalt im Browser ersetzt werden.

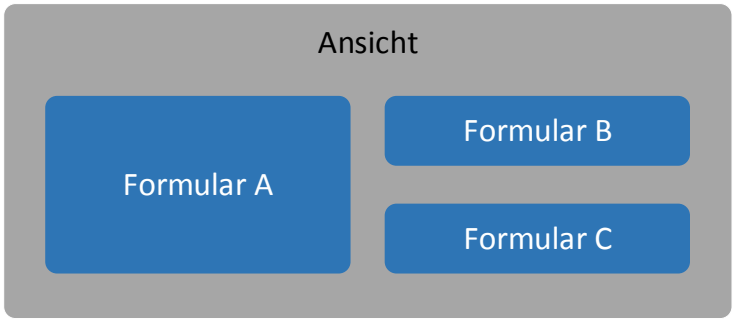


Abbildung 4.7.: Grundlegende Ansicht mit Formularen

4.4.3 Wizard

Ein Wizard ist eine spezielle Form eines Formulars. Er fasst, anders als ein Formular (siehe Kapitel 4.4.2), einen Prozess oder Verwaltungsvorgang zusammen. Zum Beispiel ist ein häufig angenommener Prozess der Administrationsumgebung, das Erstellen eines CLT.

4.4.4 Bussystem

In vielen Web-Applikationen wird nach einem Klick, über eine Verlinkung, auf eine neue Webseite verwiesen. Um Informationen zu übergeben, werden der Verlinkung parametrisiert Daten angefügt. Beim Laden der Webseite werden diese dann verarbeitet. Dieses Vorgehen ist bei dieser Architektur nicht möglich, da beim Wechsel eines Formulars die Seite nicht erneut geladen wird. Eventuelle Informationen müssen differenziert übergeben werden. Um auch hier einer Desktop Anwendung nahe zu kommen, wird ein Bus genutzt. Ein Bus ist ein System zur Datenübertragung zwischen mehreren Teilnehmern. Auf diesem Bus werden Ereignisse gelegt. Jedes Formular kann sich für ein oder mehrere Ereignisse am Bus registrieren. Findet ein Formularwechsel statt und es sollen Informationen übergeben werden, wird ein entsprechendes Ereignis auf dem Bus gelegt. Das Zielformular, das sich für das entsprechende Ereignis registriert hat, bekommt die Information und kann diese verarbeiten (siehe Abbildung 4.8).

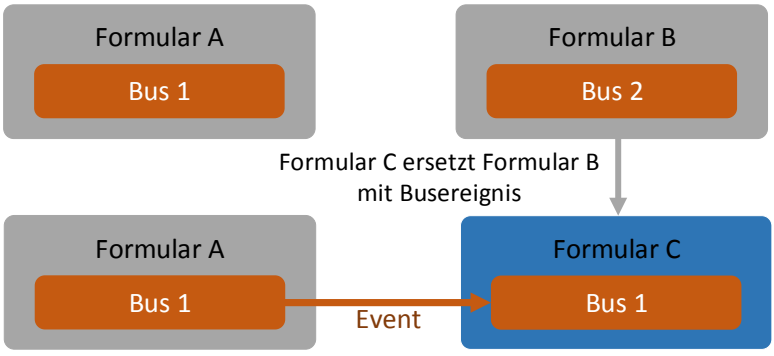


Abbildung 4.8.: Austausch eines Formulars mit Busereignis

4.4.5 Controller

Ein Controller nimmt Benutzerinteraktionen der Ansichten entgegen, wertet sie aus und leitet diese gegebenenfalls an die Schnittstellen der anderen Schichten des proCollab Prototypen weiter. Die Controller der Administrationsumgebung werden unterteilt in Controller für die Benutzerverwaltung, Typen- und Instanzverwaltung, Authentifizierung und Zugriffskontrolle.

4.5 Gestaltung der Benutzeroberfläche

Bei der Gestaltung der Benutzeroberfläche der Administrationsumgebung müssen die in Kapitel 3.4 definierten nicht-funktionalen Anforderungen berücksichtigt werden. Eine übersichtliche Gestaltung wird erreicht, wenn zum einen nur eine limitierte Anzahl an Elementen die Aufmerksamkeit des Nutzers fordern. Hier kann als Grenze die Millersche Zahl genommen werden [Geo55]. Zum Anderen reduziert eine konsistente Anordnung der gleichen Elemente und Formulare den extraneous cognitive load und somit die Komplexität der Ansicht [SAK11]. Der Nutzer benötigt weniger Aufmerksamkeit um sich zurecht zu finden und kann sich mehr auf die Systemfunktionalitäten konzentrieren. Des Weiteren wird eine unkomplizierte Gestaltung als angenehm empfunden und bei der Auswahl der Software berücksichtigt.
Als Folgerung wird in jeder Ansicht eine Kopfleiste angeboten. Über diese sind alle Systemfunktionalitäten erreichbar und gleichzeitig wird eine Übersicht aller angebotenen Funktionen geschaffen.

4.6 Mock Ups

Die Gestaltung der Benutzeroberfläche lässt sich in Teilbereiche zusammenfassen. Einerseits gibt es die Gestaltung der Anmeldung, sowie der Registration. Hier sind keinerlei Systeminformationen relevant. Deshalb werden nur ein Formular zur Anmeldung oder eines zur Registrierung angezeigt (siehe Abbildung 4.9). Andererseits gibt es nach der Anmeldung am System auf jeder Ansicht die selbe Kopfleiste, in der alle Funktionen anwählbar sind. Darunter werden die jeweiligen Formulare einheitlich angezeigt und gegebenenfalls gewechselt (siehe Abbildung 4.10). Um die Komplexität der Benutzeroberfläche gering zu halten werden Dialog-Fenster, für die Suche und Auswahl von Systemelementen, eingeblendet (siehe Abbildung 4.11). Ist ein Dialog-Fenster aktiv, werden alle anderen Anzeigenkomponenten nur modal angezeigt. Dadurch kann die Aufmerksamkeit des Nutzers auf das Dialog-Fenster gerichtet werden.

Abbildung 4.9.: MockUp für die Anmeldung

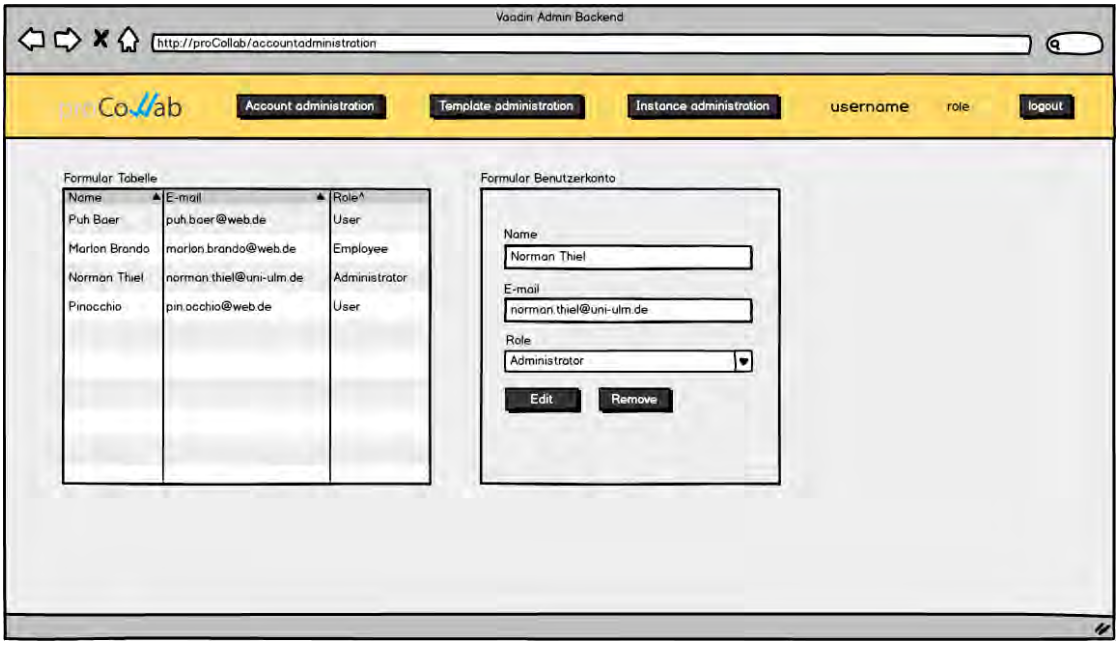


Abbildung 4.10.: MockUp einer Ansicht nach der Anmeldung

Abbildung 4.11.: MockUp des Dialog-Fensters

Kapitel 5

Implementierung

Im vorigen Kapitel 4 wurde die Konzeption der administrativen Umgebung ausgearbeitet. In diesem Kapitel soll nun die darauf basierende Implementierung vorgestellt werden. Dabei werden zuerst die zugrunde liegende Technologien erläutert (siehe Kapitel 5.1). Anschließend wird die Implementierung der einzelnen Architekturkomponenten dargelegt und ihr Zusammenspiel(siehe Kapitel 5.2) beschrieben.

5.1 Technologie

Im den folgenden Abschnitten werden die bei der Implementierung benutzten Technologien kurz erläutert.

5.1.1 Asynchronous JavaScript and XML

Asynchronous JavaScript and XML (AJAX) ist eine Webtechnologie für die Datenübertragung im Browser. Es ermöglicht den Inhalt einer Seite dynamisch zu verändern, ohne dass diese neu geladen werden muss. Dabei werden Daten zuerst vom Server angefordert. Sind die Daten fertig geladen wird eine JavaScript-Funktion aufgerufen, die die Daten weiter verarbeitet und zum Beispiel in die Webseite einbindet. AJAX ist damit keine Programmiersprache, sondern nutzt HTML, XML, XSLT, DOM und JavaScript.

AJAX-Aufrufe sind asynchron, dies bedeutet dass sie zu keinem festen Zeitschema ausgeführt werden, sondern unmittelbar, wenn der Befehl erfolgt, mit dem Webserver kommunizieren. Während dessen kann der Benutzer weiterhin mit der Oberfläche interagieren [DD08].

5.1.2 Rich Internet Application

Eine *Rich Internet Applikation* (RIA) ist eine Webanwendung, die die Funktionalität und den Interaktionsgrad einer Desktopanwendung bereitstellt. Klassische Webanwendungen sind seitenbasiert, das heißt bei jedem Klick wird ein kompletter Ladevorgang der Folgeseite und der Seitenaufbau durchgeführt. Eine RIA hingegen ist ereignisorientiert. Es werden nur die verlangten Daten nachgeladen, zum Beispiel über AJAX und damit die Ansicht aktualisiert. Durch die kürzeren Lade- und Interaktionszeiten werden gleichzeitig reichhaltigere Interaktionsmöglichkeiten für den Nutzer ermöglicht [DD08].

5.1.3 Google Web Toolkit

Das Google Web Toolkit (GWT) ist ein Framework mit dem Webanwendungen erstellt werden können. Sowohl die clientseitige-, als auch die serverseitige Implementierung wird in Java geschrieben. Dies ist möglich, weil GWT über einen integrierten Compiler verfügt, der den Java Code in JavaScript und HTML Code übersetzt. Die Kommunikation zwischen dem Server und dem Client erfolgt über *Remote Procedure Calls* [TH13]. Ein Remote Procedure Call ist eine Technik die Interprozesskommunikation realisiert. Das bedeutet, dass Aufrufe eines Prozesses oder einer Funktion eines anderen Adressraumes umgesetzt werden können [BN84].

5.1.4 Syntactically Awesome Stylesheet

Syntactically Awesome Stylesheet (Sass) ist eine Erweiterung der Stylesheet-Sprache *Cascading Style Sheet* (CSS) [lan13]. Anhand der definierten Regeln werden Daten, wie beispielsweise Texte, Grafiken und Tabellen, interpretiert und für die Bildschirmausgabe

formatiert. Es existieren zwei verschiedene Syntaxe: Zum Einen die ältere SASS Syntax und die, in der dritten Version von Sass vorgestellte, SCSS Syntax. Letztere wurde an die, den Entwicklern bekannteren, CSS Syntax angelehnt. SCSS erweitert CSS um das Benutzen von Variablen und Mixins, Verschachtelung und Vererbung von Selektoren [CC11].

5.1.5 Java Servlet

Ein *Java Servlet* ist eine Java Klasse aus der *Java Enterprise Edition* (Java EE) Technologie, die der Java Servlet API entspricht [Ora13]. Java Servlets werden instanziiert und auf einem Java Webserver ausgeführt werden, um Anfragen von Clients entgegenzunehmen beziehungsweise zu beantworten. In der Regel sind sie auf einer, auf dem Server laufenden, Java Webanwendung integriert [Per04].

5.1.6 Vaadin

Vaadin ist ein freies Framework zum Erstellen einer Webanwendung. Das Framework übernimmt die Kommunikation zwischen Server und Client, sodass das Request/Response Verfahren des HTTP- Protokolls in der Implementierung nicht weiter berücksichtigt werden muss. Die Implementierung erfolgt in Java. Es werden bereits eine Vielzahl an Komponenten der Benutzeroberfläche mitgeliefert, deren Implementierung sonst aufwendig sind. Beispielsweise gibt es Tabellen und Bäume, die nur mit Daten zu befüllen sind und anschließend im Browser angezeigt werden können. Beim Ausführen der Webanwendung wird die in Java erstellte Benutzeroberfläche in ein, für den Browser, kompatibles, HTML und JavaScript übersetzt.

Vaadin ist ein serverseitiges Framework mit einer clientseitigen Engine. Die Engine ist ein eigenständiger Teil des Frameworks, das als JavaScript Programm im Browser läuft und ist für die Übersetzung der Benutzeroberfläche in HTML und Javascript verantwortlich. Beim Verwenden einer Vaadinanwendung werden keine weiteren Plugins im Browser benötigt. Benutzereingaben und Interaktionen überträgt die clientseitige Engine zum

Webserver. Die AJAX Technologie (siehe Kapitel 5.1.1) hilft hier die Seitenladezeiten gering zu halten und ermöglicht eine hohe Interaktionsrate für den Nutzer. Die Entwicklung der Webanwendung findet ausschließlich serverseitig statt. Das Vaadin Framework läuft als Teil des Servlets (siehe Kapitel 5.1.5) auf einem Java Anwendungsserver, der die Laufzeitumgebung stellt. Das Servlet erhält clientseitige Requests (siehe Abbildung 5.1). Der Terminal Adapter interpretiert die Requests und ordnet sie entsprechenden Events zu (siehe Abbildung 5.1, Terminal Adapter). Komponenten der Benutzeroberfläche (siehe Abbildung 5.1, UI Komponente), die mit diesen Events verbunden sind übergeben diese an die Logik der Benutzeroberfläche (siehe Abbildung 5.1, UI Logik), wo sie verarbeitet werden. Wenn Veränderungen an der serverseitigen Benutzeroberfläche festgestellt werden, werden diese durch eine Response an die clientseitige Engine weitergereicht, dort verarbeitet und angezeigt (siehe Abbildung 5.1. [Grö11]

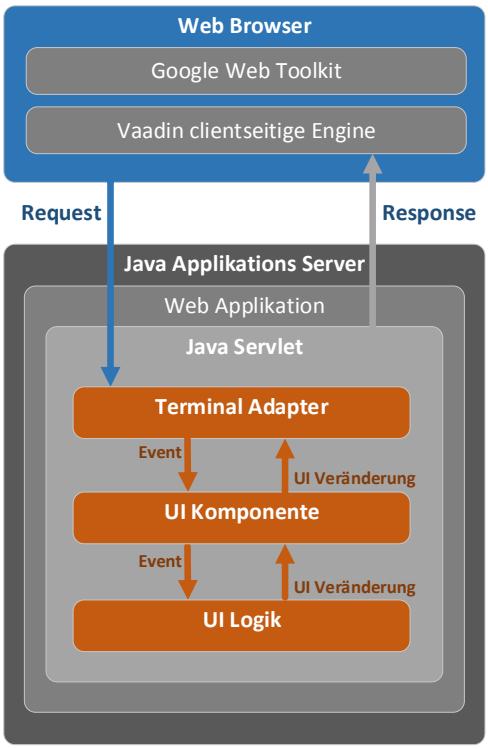


Abbildung 5.1.: Vaadin Architektur
nach [Grö11]

Die Darstellung der Webanwendung wird über *Themes* in Sass (SCSS Syntax) (siehe Kapitel 5.2.7) gesteuert. Widgets und alle Komponenten der Benutzeroberfläche können mit Hilfe von Stylesheets formatiert werden.
In dieser Arbeit wurde das Vaadin Framework für die Implementierung vorgegeben. Das Framework ermöglicht das Erstellen einer RIA (siehe Kapitel 5.1.2) und somit große und vielfältige Möglichkeiten mit dem Nutzer schnell zu interagieren. Dies steht im Besonderen im Anforderungsprofil der Administrationsumgebung des proCollab Prototypen, da bei der Nutzung von Checklisten schnell Items geklickt und verschoben werden (siehe Kapitel 3.4). Treten Interaktionen auf, wird dies dem Server gesendet, welcher entscheidet was getan werden soll.

5.2 Ausgewählte Implementierungsaspekte

In diesem Kapitel werden im Folgenden ausgewählte Teile der Implementierung präsentiert. Im Detail wird in Kapitel 5.2.1 die Generierung der grafischen Oberfläche anhand sogenannter Views auf Basis von Vaadin (siehe Kapitel 5.1.6) und der Programmiersprache Java beschrieben. Anschließend wird in den Kapiteln 5.2.2 und 5.2.3 die Verwaltung von häufig genutzten Komponenten oder Prozessen mit Hilfe von sogenannten Forms oder Wizards beschrieben. Zur Kommunikation der Komponenten der administrativen Umgebung untereinander, wird ein System aus Eventbus (siehe Kapitel 5.2.4), Event (siehe Kapitel 5.2.4.1) und Handler (siehe Kapitel 5.2.4.2) genutzt. Das Zusammenspiel wird in Kapitel 5.2.4.3 beschrieben. Daraufhin wird auf Basis der Kapitel 4.5 und 4.6 die Umsetzung der Benutzeroberfläche (siehe Kapitel 5.2.5) und in Kapitel (siehe Kapitel 5.2.6) speziell die Gestaltung der Checkliste erläutert. Im Anschluß wird, die für die Gestaltung verwendete Stylesheet Sprache SCSS (siehe Kapitel 5.2.7) kurz betrachtet.

5.2.1 View

In der Implementierung der administrativen Umgebung des proCollab Prototypen ist die Klasse *View* (dt. Ansicht, vgl. Kapitel 4.4.1) für die grafische Darstellung des Systems im Browser verantwortlich. Die Klasse implementiert hierfür die 'View' Schnittstelle des

Vaadin Frameworks und kann damit alle vom Framework bereitgestellten grafischen Komponenten anzeigen.
Wie in Kapitel 4.4.1 beschrieben soll es in der administrativen Umgebung für jede Funktionalität eine Ansicht geben. Um zu garantieren, dass alle Ansichten über die selben Schnittstellen verfügen, wurde eine abstrakte Klasse *ViewPattern* erzeugt, die die 'View' Schnittstelle des Vaadin Frameworks und alle weiteren benötigten Schnittstellen implementiert (siehe Abbildung 5.2).

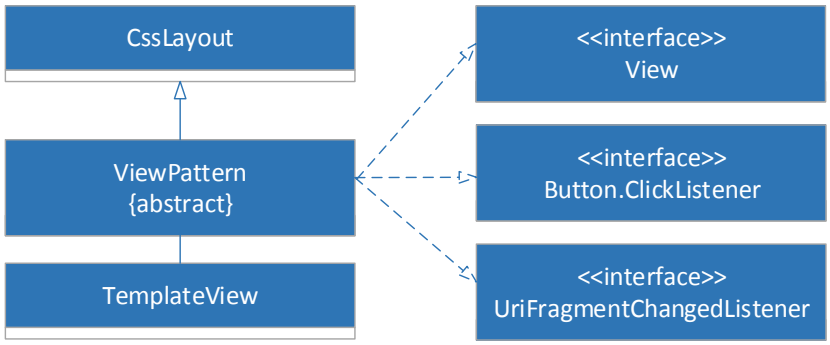


Abbildung 5.2.: Erstellung einer Ansicht am Beispiel der TemplateView

Somit wurde ein Grundgerüst für alle Ansichten der administrativen Umgebung geschaffen. Wird eine neue Ansicht erstellt erweitert diese das ViewPattern (siehe Abbildung 5.2), hier veranschaulicht an der Klasse *TemplateView*. Dies bedeutet nicht nur eine Quellcodeersparnis, sondern auch eine Erhöhung der Wartbarkeit. Ändern sich im Laufe der Benutzung die Anforderungen an die Ansichten, ist eine Änderung in der Klasse ViewPattern ausreichend, um eine Änderung in allen Ansichten zu erzeugen.
Um zwischen den Ansichten zu wechseln, bietet Vaadin die Klasse *Navigator* an. Dieser Klasse können durch die angebotene Methode *addView(View, ViewName)* Ansichten hinzugefügt werden. Alle hinzugefügten Ansichten können anschließend mit *navigateTo(ViewName)* erreicht werden.
Da es sich um die Implementierung einer administrative Umgebung handelt, müssen Zugriffe mit unzureichender Nutzerrolle verhindert werden. Deshalb werden jeder An-

sicht, die für sie gültigen Zugriffsrollen hinzugefügt (siehe Listing 5.1). Zugriffsrollen werden in der Systemarchitektur durch ein Enum *Roletype* repräsentiert. Dieser kann die Werte *Administrator, Manager, Employee, User* und *None* einnehmen. Zugriff zur Funktionalität der administrativen Umgebung wird ausschließlich mit der Zugriffsrolle Administrator gewährt.

```
1 public class TemplateView extends ViewPattern {
2    public static final Roletype[] permissionRoles
3       = { Roletype.ADMINISTRATOR };
4    ...
5 }
```

Listing 5.1: Definition der notwendigen Zugriffsrollen in der Klasse TemplateView

Natürlich ist es notwendig die Nutzerrollen vor dem Wechsel zwischen den Ansichten zu überprüfen. Hierfür bietet Vaadin die Schnittstelle *ViewChangeListener* mit unter anderem der Methode *beforeViewChange(ViewChangeEvent)* an. Soll nun eine Ansicht gewechselt werden, generiert die Laufzeitumgebung ein Ereignis mit dem Namen *ViewChangeEvent* und die Methode beforeViewChange(ViewChangeEvent) wird aufgerufen. Die Methode wurde in Bezug auf die Zugriffssteuerung so implementiert, dass diese, falls der Zugriff erlaubt werden soll, den Wahrheitswert true zurückgibt (ansonsten false). Um hieraus eine möglichst kompakte Klasse für die Zugriffskontrolle zu entwickeln, wurde eine neue Klasse *AccessControl* erstellt, die die Schnittstelle ViewChangeListener implementiert (siehe Abbildung 5.3).

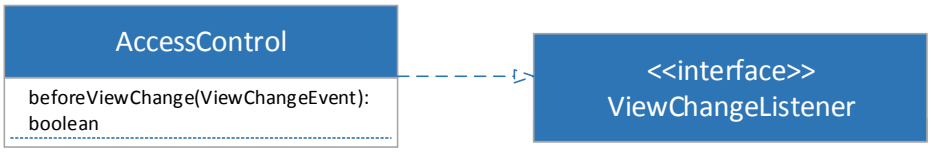


Abbildung 5.3.: Erstellung der Klasse AccessControl

Ziel dieser Klasse ist es, die Ereignisse vom Typ ViewChangeEvent entgegenzunehmen und zu kontrollieren, ob der Zugriff gewährt werden kann oder nicht. Für eine erfolgreiche Überprüfung muss zunächst die Zielansicht bekannt sein. Die Instanz der Zielansicht ist durch die Methode *getNewView()* des Ereignisses erhältlich. Nun kann mit dem *instanceof*-Operator die Zielansicht erfasst werden und eine individuelle Regelung erstellt werden.(siehe Listing 5.2).
Um die Überprüfung durchzuführen, werden die benötigten Zugriffsrollen der Zielansicht und die vorhandene Nutzerrolle benötigt. Da die Zugriffsrollen statisch sind (siehe Listing 5.1), können diese über den Klassennamen der Ansicht angefordert werden. Die Nutzerrolle erhält man über die Helferklasse *SessionHandler* mit der Methode *getAttribute(attribute)*. Die Überprüfung wird in eine eigene Methode ausgelagert. Die nun vollständige Methode beforeViewChange(ViewChangeEvent) führt damit die Überprüfung durch (siehe Listing 5.2).

```
1  private boolean checkPermission(Roletype[] roles, String level){
2      for (int i = 0; i < roles.length; i++) {
3          if (roles[i].toString().equals(level)) {
4              return true;
5          }
6      }
7      return false;
8  }
9  @Override
10 public boolean beforeViewChange(ViewChangeEvent event) {
11     if (event.getNewView() instanceof TemplateView) {
12         return checkPermission(TemplateView.permissionRoles,
13         SessionHandler.getAttribute("role"));
14     }
15 }
```

Listing 5.2: Zugriffskontrolle

5.2.2 Form

Auf einer Ansicht (siehe Kapitel 5.2.1) sollen im Besonderen sogenannte *Forms* (dt. Formulare, vgl. Kapitel 4.4.2) platziert werden (siehe Abbildung 4.7). Ein Formular ist für die Verwaltung häufig genutzter einzelner Komponenten einer Ansicht oder mehrerer solcher Komponenten im Zusammenspiel verantwortlich. Beispielsweise werden Tabellen genutzt, um mehrere Suchergebnisse aufzulisten. Tabellenelemente können dabei zum Beispiel Datenelemente der Nutzer sein. Um hierbei nicht für jede Entität eine eigene Tabelle mit Verwaltung ihrer Elemente implementieren zu müssen, wurde eine generische Klasse *FormTable* entwickelt. Diese Form übernimmt die Darstellung und Aufbereitung der Elemente aller Entitäten. Somit wird bei Bedarf wird noch das benötigte Formular einer Ansicht hinzugefügt und kann anschließend genutzt werden.
Da ein Formular verschiedene Elemente mit unterschiedlichem Nutzen und Gebrauch beinhaltet, kann nur bedingt ein allgemeines Grundgerüst definiert werden. Lediglich aus Gründen der Gestaltungsmöglichkeiten, wird als Layout eines jeden Formulars das von Vaadin angebotene *CssLayout* gewählt. Eine abstrakte, nicht instanziierbare Klasse *FormPattern* wird angelegt, die das CssLayout erweitert. Jedes neue Formular erweitert nun FormPattern (siehe Abbildung 5.4). Alle Formulare müssen dennoch eigenständig gewartet oder erweitert werden.

Abbildung 5.4.: Erstellung eines Formulars am Beispiel des FormTables

5.2.3 Wizard

Ein *Wizard* (vgl. Kapitel 4.4.3) ist eine spezielles Formular (siehe Kapitel 5.2.2) und bildet einen Prozess oder Vorgang auf einer Ansicht (siehe Kapitel 5.2.1) ab. Er beinhaltet typischerweise mehrere Formulare und ist für die Kommunikation unter ihnen

zuständig. In dieser Arbeit sind mehrere Wizards für das Erstellen und Bearbeiten der Systemelemente, wie zum Beispiel ORT, CLT oder CET verantwortlich.

5.2.4 Eventbus

In Bezug auf die Ansichten kann ein Formular kann nicht ohne weiteres mit einer anderen kommunizieren. Da jedoch, zum Beispiel bei einem Wechsel der Formulare des öfteren Daten übermittelt werden müssen um einen Zusammenhang in einer Ansicht zu ermöglichen, wird ein Architekturelement für die Kommunikation benötigt. Ein sogenannter *Eventbus* ist ein Bus-System zur Datenübertragung zwischen mehreren Formularen über einen gemeinsamen Übertragungsweg (vgl. Kapitel 4.4.4). Wenn bei einer Nutzerinteraktionen, zum Beispiel das Drücken eines Buttons, Daten an ein weiteres Formular übertragen werden sollen, wird ein *Event* (dt. Ereignis) gefeuert. Ist das Zielformular an dem Eventbus registriert, erhält es diese Daten.
Zur erleichterten Umsetzung wurde in dieser Arbeit der Google Guava Eventbus gewählt [Goo13b]. Dieser nutzt ein einfaches Beobachter-Entwurfsmuster, das sogenannte *Publish-Subscribe-Muster* [GR01]. Hierbei wird ein Ereignis nicht direkt an einen spezifisch adressierten Empfänger übertragen. Stattdessen wird ein Ereignis, wie zum Beispiel ein Klick im FormTable, über den Bus veröffentlicht, ohne Kenntnis darüber, ob es einen oder mehrere Empfänger gibt.
Für die Nutzung des Eventbusses wird zunächst eine Klasse *BackendBus* erstellt, die den eigentlichen Eventbus enthält (siehe Listing 5.3).

```
1 public class BackendBus {
2    public static volatile EventBus EVENT_BUS = new EventBus();
3 }
```

Listing 5.3: Auszug der Klasse BackendBus

Dem Eventbus werden die Modifizierer *volatile* und *static* zugewiesen (siehe Listing 5.3, Zeile 2). Static weist stets eine Klassenvariable aus. Das bedeutet, dass auch bei mehr-

maliger, eventuell fehlerhafter, Instanziierung der Klasse BackendBus, der Eventbus unter allen Instanzen geteilt wird. Er existiert auf jedem Server auf dem die administrative Umgebung installiert ist, also nur einmalig. Das ist wichtig, da sonst beim Auslösen mehrere Ereignisse gefeuert werden könnten. Volatile weist dem Bus eine weitere wichtige Eigenschaft zu. So gekennzeichnete Variablen werden nicht zwischengespeichert. Dadurch wird sichergestellt, dass beim Lesen mehrerer nebenläufiger Prozesse immer der richtige Wert übertragen wird.

5.2.4.1 Event

Ein *Event* (dt. Ereignis) tritt immer dann auf, wenn ein Nutzer mit der Benutzeroberfläche interagiert und deswegen eventuell Daten zwischen Formularen ausgetauscht werden müssen. Wenn ein Nutzer auf ein Tabellenelement klickt und anschließend die Details betrachtet, wurde in diesem Zug ein Ereignis ausgelöst. Genauer wird bei einem Klick auf ein Element in dem FormTable mit einem Ereignis unter anderem die Information übertragen, welches Element angeklickt wurde. Nach dem Empfang eines Ereignisses holt sich das Zielformular das entsprechende Element aus dem FormTable und kann die entsprechenden Daten anzeigen.
Um ein neues Ereignis zu erstellen muss die Klasse *GwtEvent* unter Angabe des entsprechenden *Handler*-Typs (siehe Kapitel 5.2.4.2) (siehe Listing 5.4) [Goo13b].

```
1 public class TableClickEvent extends
2    GwtEvent<TableClickEventHandler> {
3    ...
4 }
```

Listing 5.4: Erstellung eines Ereignisses vom Typ TableClickEvent

Anschließend können Ereignissen Attribute zugewiesen werden. Diese können genutzt werden, um Informationen in Form von einfachen Datentypen oder auch Objekten zu übertragen. Sie werden beim Aufruf des Konstruktors übergeben. Da es sich ausschließ-

lich um eine Übertragung von Daten handelt, werden keine Wert setzenden Methoden, wie sogenannte *Setter*-Methoden benötigt. Beim Empfang des Ereignisses können die Informationen der Attribute über sogenannte *Getter*-Methoden abgefragt werden (siehe Listing 5.5).

```
1  public class TableClickEvent extends
2     GwtEvent<TableClickEventHandler> {
3     // Attribut um ID des geklickten Elements zu uebergeben
4     private int clickedId;
5     // TableClickEvent Konstruktor
6     public TableClickEvent(int clickedId) {
7        this.clickedId = clickedId;
8     }
9     // Gibt die ID des geklickten Elements zurueck
10    public int getItemID() {
11       return clickedId;
12    }
13 }
```

Listing 5.5: Beispielhafte Zuweisung und Rückgabe eines Attributs in Ereignissen

Zusätzlich erfordert die Erweiterung von GwtEvent, dass zwei weitere Methoden implementiert werden. Zum Einen muss die Methode *getAssociatedType()* implementiert werden. Sie gibt den Typ des Handlers (siehe Kapitel 5.2.4.2) zurück und wird vom Eventbus benötigt, um das Ereignis an die betreffenden Handler zu verweisen. Zum Anderen muss *dispatch(Handler)* implementiert werden. Diese ruft die Methode des Handlers auf, mit der sich ein Formular am Bus registriert hat (siehe Listing 5.6).

```
1  public class TableClickEvent extends
2     GwtEvent<TableClickEventHandler> {
3     // Typ des Handlers
4     private static Type<TableClickEventHandler> TYPE
5        = new Type<TableClickEventHandler>();
6     private int clickedId;
7     public TableClickEvent(int clickedId) {
8        this.clickedId = clickedId;
9     }
10    public int getItemID() {
11       return clickedId;
12    }
13    // Gibt den Handlertyp zurueck
14    @Override
15    public Type getAssociatedType() {
16       return TYPE;
17    }
18    // ruft die Methode des Handler auf und
19    // uebergibt sich selbst
20    @Override
21    protected void dispatch(TableClickEventHandler handler) {
22       handler.onTableAction(this);
23    }
24 }
```

Listing 5.6: Implementierung der Klasse TableClickEvent

5.2.4.2 Handler

Um Ereignisse (siehe Kapitel 5.2.4.1) effektiv nutzen zu können, werden sogenannte *Handler* benötigt. Handler sind für die Verarbeitung von Ereignissen zuständig. Das heißt, sie empfangen ein Ereignis, analysieren es und verarbeiten die Nutzereingabe. Jeder Handler ist nur auf ein bestimmtes Ereignis spezialisiert. Das bedeutet, dass beim Erstellen eines Ereignisses immer auch der entsprechende Handler erstellt werden muss.
Ein Handler ist eine Schnittstelle und muss die Schnittstelle *EventHandler* erweitern. Es gilt lediglich eine Methode zu definieren, die bei der Registration des Handlers implementiert werden muss (siehe Listing 5.7)

```
1 public interface TableClickEventHandler extends EventHandler {
2    void onTableAction(TableClickEvent clickEvent);
3 }
```

Listing 5.7: Erstellung eines Handlers

5.2.4.3 Zusammenspiel von Ereignis, Handler und Eventbus

Um nun ein Ereignis über den Bus zu schicken, wird ein entsprechendes Ereignis instanziiert und mit der *post* Methode des Eventbus versendet (siehe Listing 5.8).

```
1 // Die ID eines in der Tabelle geklickten Elementes
2 // wird ueber den Bus versendet
3 BackendBus.EVENT_BUS.post(new TableClickEvent(clickedID));
```

Listing 5.8: Verschicken eines Ereignisses

Um das Ereignis empfangen zu können, muss der Eventbus mit dem entsprechenden Handler in dem Formular, das die Daten anzeigen soll, registriert werden. Dabei muss die Methode des Handlers entsprechend implementiert werden. Diese wird zur Analyse und Verarbeitung der Information genutzt. Nun muss sich der Handler noch am Eventbus anmelden. Dies geschieht mit der Annotation *@Subscribe* (siehe Listing 5.9).

```
1 BackendBus.EVENT_BUS.register(new TableClickEventHandler() {
2    // Anmelden des Handler am Bus
3    @Subscribe
4    @Override
5    public void onTableAction(TableClickEvent clickEvent) {
6       // tu etwas
7    }
```

Listing 5.9: Empfangen eines Events

5.2.5 Umsetzung der Benutzeroberfläche

Die Umsetzung der Benutzeroberfläche orientiert sich an den in Kapitel 4.6 erstellten MockUps und den nicht-funktionalen Anforderungen, speziell der Bedienbarkeit (siehe Kapitel 3.4.1). Dazu wird die Benutzeroberfläche unterteilt.
Bei Authentifizierungsvorgängen, wie der Anmeldung oder der Registrierung am System, werden nur die notwendigen Formulare angezeigt (siehe Kapitel 4.9). Die Aufmerksamkeit des Nutzers liegt ausschließlich bei die Authentifizierung. Bei der Anmeldung wird das Anmeldeformular (siehe Abbildung 5.5b) und bei der Registrierung das Registrationsformular eingeblendet (siehe Abbildung 5.5a).

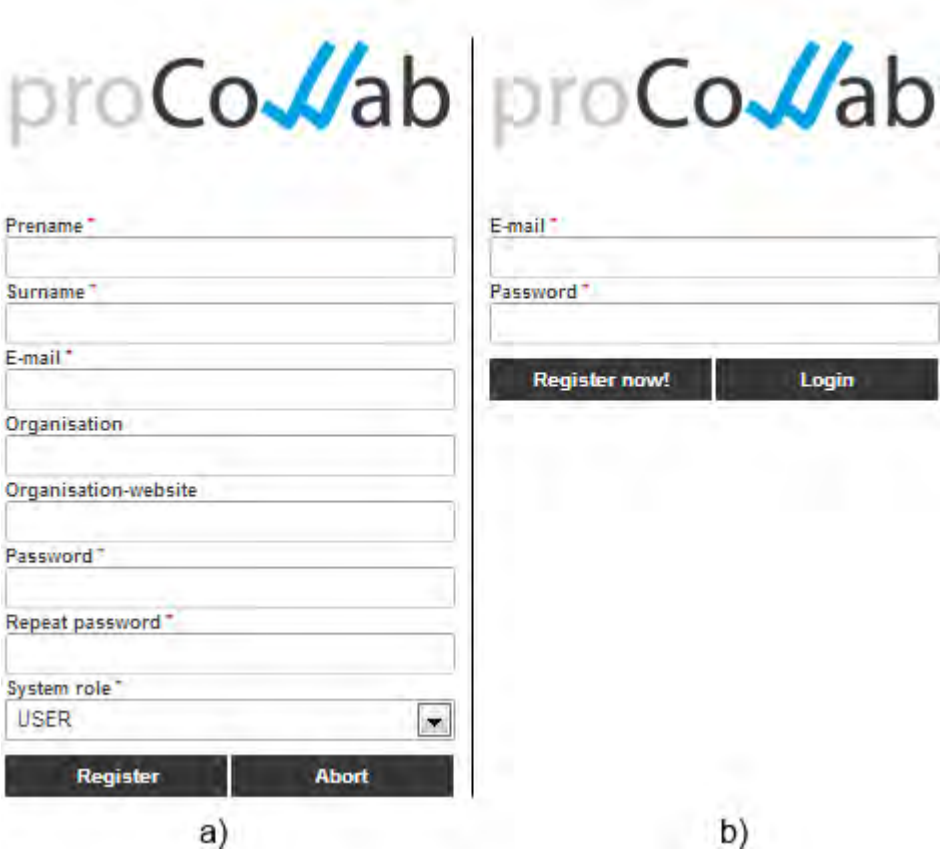


Abbildung 5.5.: Das Registrations- und Anmeldeformular

Hat sich ein Nutzer erfolgreich am System angemeldet, ist die Oberfläche in zwei Bereiche unterteilt. Es gibt einen Kopfbereich für die Kopfleiste und einen Inhaltsbereich, auf dem die Formulare angezeigt werden (siehe Kapitel 4.10). Die Kopfleiste beinhaltet ein geordnetes Dropdown-Menü über dessen alle Systemfunktionen auswählbar sind. Ist eine Systemfunktion ausgewählt, können im Inhaltsbereich die entsprechenden Formulare bearbeitet werden. Ist beispielsweise der Menüpunkt *All Persons* ausgewählt, werden alle im System vorhandenen Personen in einem Tabellenformular aufgelistet und können bearbeitet werden (siehe Abbildung 5.6).

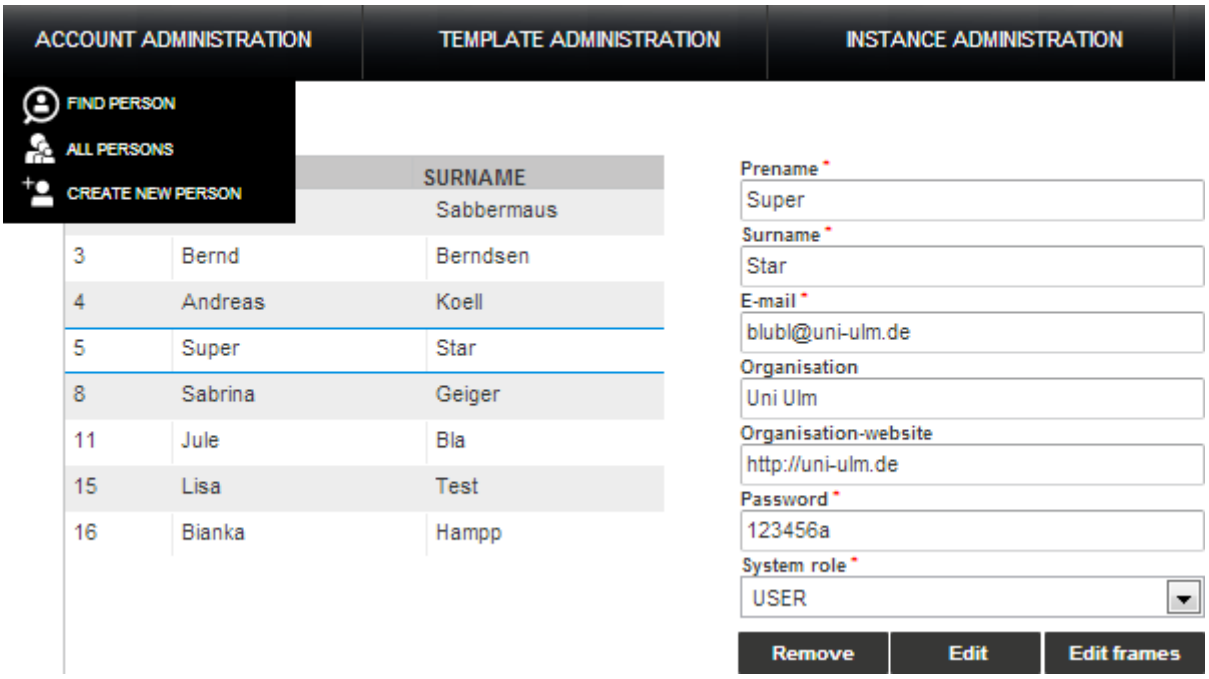


Abbildung 5.6.: Die Benutzeroberfläche der administrativen Umgebung nach der Anmeldung

Das Dropdown-Menü der Kopfleiste ist in den Verwaltungsbereichen Benutzerverwaltung, Vorlagenverwaltung und Instanzverwaltung gegliedert (siehe Abbildung 5.7). Jede Systemfunktion ist einem dieser Verwaltungsbereiche zugeordnet und unter einem Menüpunkt auswählbar. Menüpunkte können weitere Untermenüs besitzen, um beispielsweise eine feinere Unterteilung der Systemfunktionen zu erzeugen oder Funktionen aufzusplitten. Ein Untermenü wird eingeblendet, sobald sich der Mauszeiger über den übergeordneten Menüpunkt befindet. Um dem Nutzer anzuzeigen, dass sich der Mauszeiger über einem Menüpunkt befindet, wird der Menüpunkt halbtransparent angezeigt (siehe Abbildung 5.7).

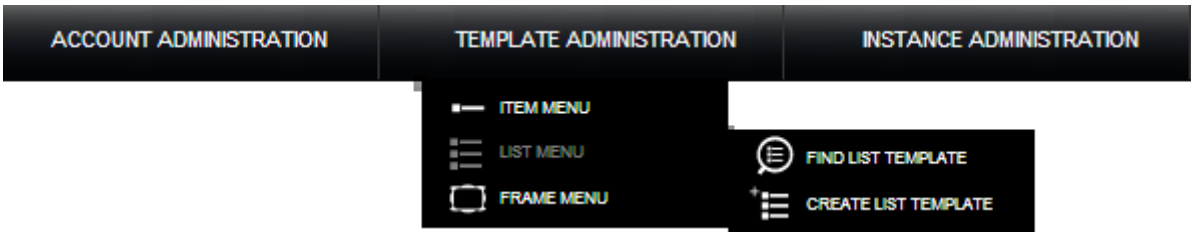


Abbildung 5.7.: Kopfleiste mit dem geöffneten Menü der Vorlagenverwaltung

Um dem Nutzer die Orientierung im Menü zu erleichtern, wird jedem Menüpunkt ein Icon zugeordnet. Ein Icon bildet die Systemfunktionalität eines Menüpunktes, anhand eines Bildes, ab. Das bedeutet, dass ein Nutzer, der ein Icon betrachtet, auf die hinter dem Menüpunkt verborgene Funktionalität schließen kann. Eine Übersicht über alle erstellten Icons bietet Abbildung 5.8.

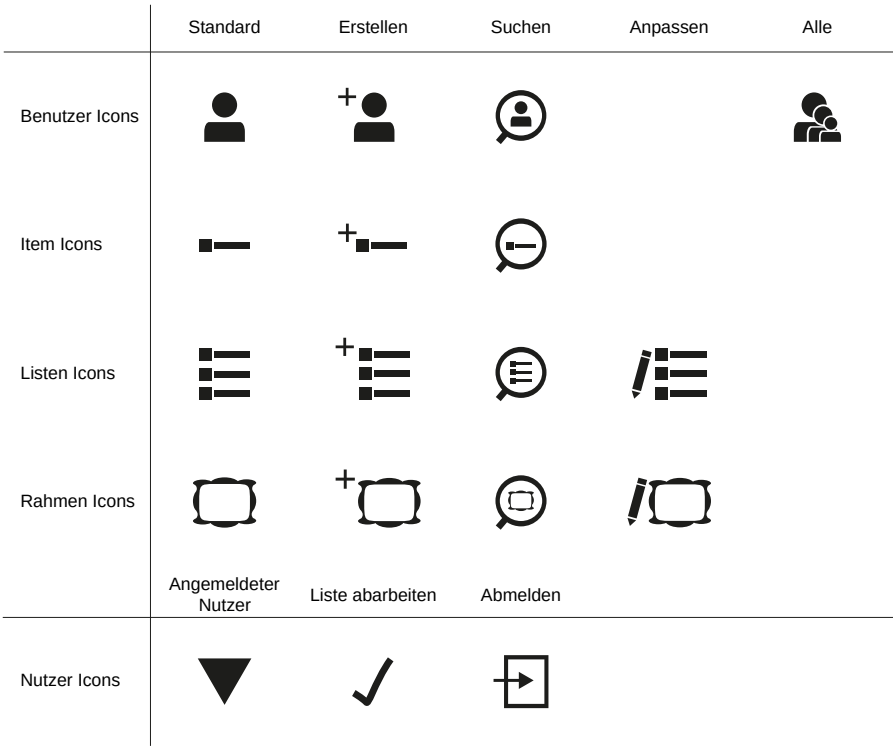


Abbildung 5.8.: Die in der administrativen Umgebung verwendeten Icons im Überblick

Für die Suche und die Auswahl von CLs, CEs oder Nutzern wird ein Dialog-Fenster genutzt. Bei Anzeige eines solchen Dialog-Fensters wird der Rest der administrativen Umgebung modal angezeigt (siehe Abbildung 5.9). Damit wird der Fokus von anderen Systemfunktionen genommen und auf das PopUp Fenster gerichtet.

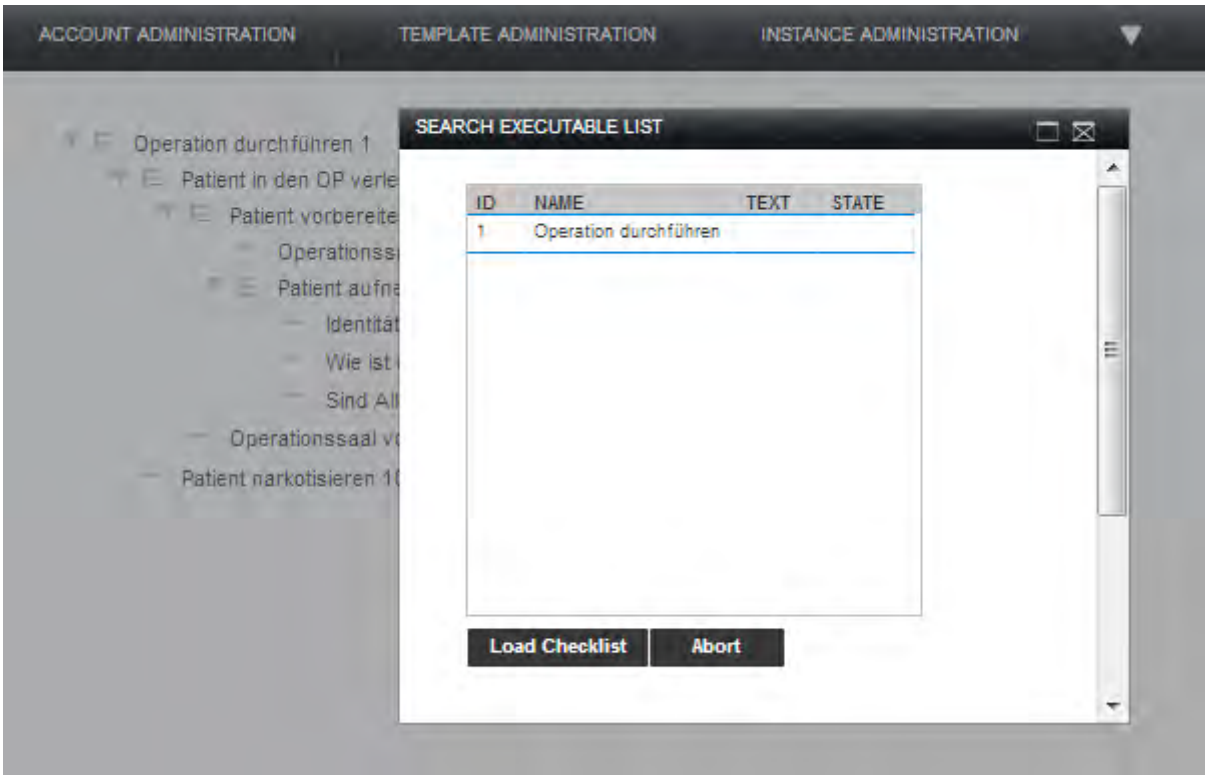


Abbildung 5.9.: Die Benutzeroberfläche mit Dialog-Fenster

5.2.6 Gestaltung der Checkliste

Die Aufteilung in Verwaltungsbereiche und die Strukturierung des Menüs stellen die Systemfunktionen für den Nutzer übersichtlich dar. Ein zentrales Element der administrativen Umgebung des proCollab Prototypen stellt die CL dar.
Da eine CL aus CLs und CEs besteht (siehe Kapitel 2.4.2), werden den Elementen einer CL ebenfalls Icons zugeordnet. CLIs und CEIs können den Status offen oder abgeschlossen haben. Um dem Nutzer ein optisches Feedback darüber zu geben, in

welchem Status sich ein CEI oder eine CLI befindet, sind weitere Icons erarbeitet worden (siehe Abbildung 5.10). Die Icons der abgeschlossenen CLIs oder CEIs werden mit einem Haken versehen. Um den gedanklichen Transfer der Icons mit einer Systemfunktion zu verstärken, werden die für das Menü entwickelten Icons wiederverwendet. Für eine erhöhte Identifikation mit der Web-Applikation wird der Haken dem proCollab-Logo entnommen.

Abbildung 5.10.: Die offenen und abgeschlossenen Checklistenicons

Um die Orientierung während dem Arbeiten mit CLs zu erleichtern, werden CLs und CEs mit steigender Baumtiefe zunehmend nach rechts versetzt. Alle CLs und CEs der gleichen Baumtiefe sind somit auch auf gleicher Tiefe angeordnet. Dies ermöglicht eine übersichtliche Darstellung der aktuell zu erledigenden und als nächstes anstehenden Punkte. Weiterhin werden CLs und CEs, über denen sich der Mauszeiger befindet, ausgegraut und eingeschoben. CLs können weitere untergeordnete CLs und CEs enthalten. Daher signalisiert ein Pfeil, ob eine CL ein- oder ausgeklappt ist. Zeigt dieser nach unten ist die CL ausgeklappt, zeigt er nach rechts ist sie eingeklappt. Abgeschlossene CLIs und CEIs werden dadurch gekennzeichnet, dass sie permanent ausgegraut sind und das Icon für abgeschlossene CLIs und CEIs gesetzt ist (siehe Abbildung 5.11).

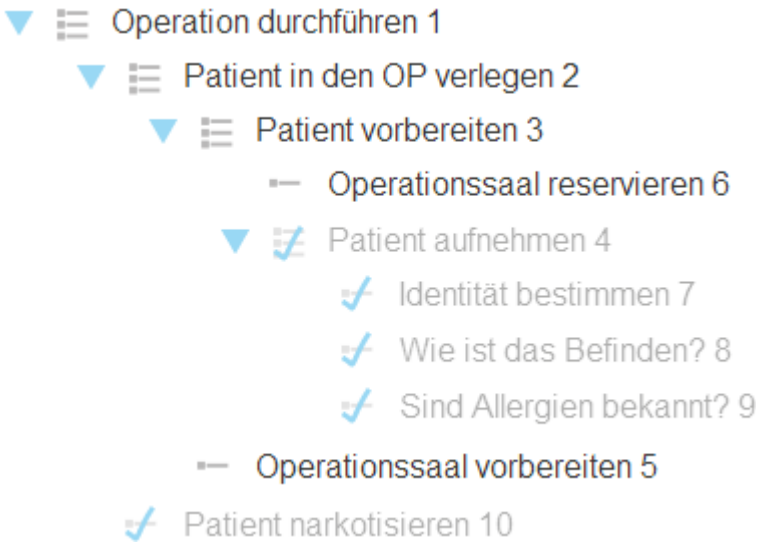


Abbildung 5.11.: Darstellung einer geschachtelten CL mit Icons

5.2.7 Gestaltung mit SCSS

Die mit Vaadin erstellte administrative Umgebung wird in HTML und JavaScript übersetzt und im Browser angezeigt (siehe Kapitel 5.1.6). Für die Gestaltung der Benutzeroberfläche könnenfür Vaadin sogenannte *Themes* mit SCSS erstellt werden. Die in den Themes definierten Regeln sind für die Formatierung der HTML Elemente verantwortlich. Bei der Erstellung eines eigenen Themes ist zu beachten, dass das von Vaadin voreingestellte *reindeer-Theme* vollständig überschrieben wird.
Für die Gestaltung der Ansichten und Formulare der administrativen Umgebung werden ID's vergeben. Anhand einer ID werden Elemente eindeutig bezeichnet und können individuell gestaltet werden. Als Beispiel kann hier die Kopfleiste FormHeader betrachtet werden, die die ID *"formheader"* besitzt. Ein Einblick über die definierten SCSS Regeln, der Ansichten und Formulare, soll anhand dieses *FormHeaders* erfolgen (siehe Listing 5.10). Mit SCSS können CSS Selektoren so verschachtelt werden, dass sie der hierarchischen Anordnung des HTML Dokuments entsprechen. Als Vergleich sind die selben Regeln in CSS definiert (siehe Listing 5.11).

```scss
#formheader {
    height:100px;
    width:100%;
    background: url(img/head_dark.svg);
    .v-horizontallayout{
        height:100px;
        width:100%;
        .v-slot{
            margin-right:20px;
        }
    }
}
```

Listing 5.10: SCSS Regeln für den FormHeader

```css
#formheader {
    height:100px;
    width:100%;
    background: url(img/head_dark.svg);
#formheader > .v-horizontallayout{
    height:100px;
    width:100%;
}
#formheader > .v-horizontallayout > .v-slot{
    margin-right:20px;
}
```

Listing 5.11: Die Regeln des FormHeader in CSS

Kapitel 6

Zusammenfassung und Ausblick

Abschließend wird noch einmal ein Rückblick über die Inhalte der Kapitel gegeben. Im Anschluss werden Ansatzpunkte für die zukünftige Weiterentwicklung der Administrationsumgebung des proCollab Prototypen benannt.
In Kapitel 1 wurden nach einer Motivation die Problem- und Zielstellung der vorliegenden Arbeit präsentiert. Danach wurde in Kapitel 2 zuerst Wissensarbeit erläutert und ein Ansatz zur Unterstützung von Wissensarbeit benannt. Anwendungsfälle von Checklisten gaben einen Einblick über die Nutzung und Struktur von Checklisten. Danach wurden die zentralen Begriffe einer Checkliste beschrieben. In Kapitel 3 ist zunächst aktuelle Software analysiert worden. Auf Basis von Anwendererzählungen wurden die funktionalen Anforderungen erhoben und anschließend nicht-funktionale Anforderungen der Administrationsumgebung bestimmt. Nach der Benennung wichtiger Programmierparadigmen wurde in Kapitel 4 zunächst das Datenmodell und dann die Systemarchitektur des proCollab Prototypen angegeben. Auf die detaillierte Beschreibung der zugrunde liegenden Architektur der Administraionsumgebung, folgten Kriterien zur Gestaltung der Benutzeroberfläche und MockUps zur Visualisierung. In Kapitel 5 sind zuerst die der Implementierung zugrunde liegende Technologien erklärt und abschließend ausgewählte Implementierungsaspekte der Administrationsumgebung detailliert beschrieben worden.

Für die zukünftige Entwicklung des proCollab Prototypen, im Speziellen der administra-

tiven Umgebung, gibt es noch einige Ansatzpunkte. Idealerweise sollte der Eventbus auf dem Client laufen, um bei der Kommunikation der Formulare keine oder nur wenig Anfragen an den Server zu schicken. Die Applikation könnte größtenteils lokal berechnet werden und nur wenig mit dem Server kommunizieren. Dadurch könnte die Serverlast verringert werden und die Performanz der Benutzerinteraktionen wäre ähnlich der einer Desktop-Anwendung. Da Vaadin jedoch serverzentriert arbeitet, ist dies nicht möglich und der Eventbus kann nur serverseitig implementiert werden.
Die mitgelieferten UI Komponenten besitzen ein voreingestelltes Design. Möchte man diese selbst gestalten, kann ein neues Theme erstellt werden. Vaadin bietet die Möglichkeit, das um einige Funktionen erweiterte SCSS zu benutzen. Trotz alledem ist die eigene Gestaltung mit sehr hohem Aufwand verbunden. Zum Einen muss das voreingestellte Theme auch an Stellen überschrieben werden, die für das eigene Design belanglos sind. Dadurch sucht man oft lang nach der Herkunft vordefinierter Regeln. Zum Anderen werden beim Übersetzen des Programmcodes in HTML, unverhältnismäßig viele Div-Container erstellt. So kommt es durchaus vor, dass man beim Gestalten eines Baumes oder anderen Komponenten zehn ineinander geschachtelte Div-Container vorfindet.
Abschließend lässt sich festhalten, dass einfache Web-Applikationen schnell und einfach in Vaadin erstellt werden können. Vaadin bietet eine umfangreiche Palette an UI-Komponenten und einen schnellen Einstieg. Entwickler, die über moderate Kenntnisse in HTML, CSS und JavaScript verfügen, werden sich jedoch mit den Einschränkungen schwer tun. Die Architektur der Administrationsumgebung würde ihr volles Potential erst durch eine Implementierung in den eben genannten Sprachen ausschöpfen. Ebenfalls ist eine Entwicklung in Frameworks, wie Google GWT, die sowohl serverseitige als auch clientseitige Implementierungen ermöglichen, denkbar.

Anhang A

Quelltexte

A.1 User Stories

A.1.1 Benutzerverwaltung

Benutzer erstellen (BV1)
Ein Administrator kann einen Nutzer, in der administrativen Umgebung, am System registrieren. Dazu kann er den Vorname, Nachnamen, Passwort, E-Mail Adresse und die Organisation des Nutzers angeben. Die E-Mail Adresse wird validiert und die Passworteingabe muss zweimalig erfolgen. Nach dem Bestätigen wird dem Nutzer eine Aktivierungs-E-Mail an sein E-Mail Konto gesendet. Nach der Aktivierung erhält der Nutzer eine weitere E-Mail mit all seinen Nutzerangaben.

Einfache Anmeldung (BV2)
Ein Nutzer des proCollab Prototypen gibt seine E-Mail Adresse und Passwort ein. Anschließend werden seine Eingaben vom Server überprüft. Bei einer Bestätigung wird ihm das Backend angezeigt, bis er sich vom System abmeldet. Bei Ablehnung kann er erneut seine Eingaben tätigen.

Benutzerdaten anpassen (BV3)
Ein Administrator kann die Daten eines Nutzers, in der administrativen Umgebung,

ändern. Dazu kann er den Vorname, Nachnamen, Passwort, E-Mail Adresse und die Organisation des Nutzers ändern. Wird die E-Mail Adresse verändert, muss diese erneut vom Nutzer bestätigt werden. Weiterhin kann ein Administrator den Nutzer einer ORI hinzufügen oder ihn von einer entfernen. Beim Hinzufügen muss die Rolle angegeben werden. Ein Administrator kann einen anderen Nutzer zum Administrator ernennen oder ihm dieses Recht nehmen.

Benutzer entfernen (BV4)
Ein Administrator kann einen Nutzer, in der administrativen Umgebung, vom System entfernen. Der Benutzer wird vom System abgemeldet und anschließend entfernt. Anschließend wird dem Nutzer eine E-Mail zugesendet, die seine Entfernung bestätigt.

Benutzer suchen (BV5)
Ein Administrator kann einen Nutzer mit der Suchfunktion, anhand der Attribute Vorname, Nachname, E-Mail Adresse und Organisation suchen. Wenn einer oder mehr Nutzer den Vorgaben entsprechen, wird eine Liste der Treffer angezeigt. Wird einer der Treffer in der Liste angeklickt, werden die Nutzerdaten angezeigt. Wird kein Nutzer zu den Vorgaben gefunden, wird eine Nachricht angezeigt.

A.1.2 ORT-Verwaltung

Erstellen eines neuen ORT (ORT1)
Ein Administrator kann in der administrativen Umgebung einen neuen ORT erstellen. Dazu muss er einen Rahmennamen angeben und kann zusätzlich eine Beschreibung, Ziel, Startdatum, Enddatum und die Bearbeitungszeit in Tagen angeben. Die Bearbeitungszeit repräsentiert die Zeit, die für die Erfüllung der ORI nach dem Start zur Verfügung stehen. Zusätzlich kann ein ORT einem anderen ORT untergeordnet werden.

Ableitung eines ORT von einem ORT (ORT2)
Ein Administrator kann einen ORT anhand einem bereits existierenden ORT ableiten. Dazu muss er einen existierenden ORT suchen und kann anschließend Details anpassen.

Ableitung eine ORT von einer ORI (ORT3)
Ein Administrator kann einen ORT anhand einer bereits existierenden ORI ableiten. Dazu muss er eine existierende ORI suchen und kann anschließend Details anpassen.

Anpassen eines ORT (ORT4)
Nach der Auswahl eines ORT, kann ein Administrator die Details des ORT anpassen. Er kann den Rahmennamen, Ziel, Startdatum, Enddatum und die Bearbeitungszeit verändern.

Entfernen eines ORT (ORT5)
Nach der Suchen und Auswahl eines ORT werden dem Administrator die Details des ORT angezeigt. Hier kann er mit Klick auf einen Button, den ORT vom System entfernen. Anschließend werden alle Verbindungen zu ORIs gelöscht.

Einen ORT suchen (ORT6)
Ein Administrator kann einen ORT mit der Suchfunktion, anhand der Attribute Rahmenname, Beschreibung und Ziel suchen. Wenn eine oder mehr ORTs den Vorgaben entsprechen, wird eine Liste der Treffer angezeigt. Wird einer der Treffer in der Liste angeklickt, werden die Details des ORT angezeigt. Wird kein ORT zu den Vorgaben gefunden, wird eine Nachricht angezeigt.

A.1.3 ORI-Verwaltung

Instanziieren einer ORI (ORI1)
Ein Administrator kann eine neue ORI anhand eines existierenden ORT instanziieren. Dazu muss er einen ORT suchen und instanziieren klicken. Anschließend kann ein Administrator die ORI-Details spezifizieren und muss einen verantwortlichen Rahmenmanager bestimmen. Weiterhin kann er die ORI einer anderen unterordnen.

Anpassen einer ORI (ORI2)
Nach der Auswahl einer ORI, kann ein Administrator die Details der ORI anpassen. Er kann den Rahmennamen, Ziel, Startdatum, Enddatum und die Bearbeitungszeit verändern.

Entfernen einer ORI (ORI3)
Nach dem Suchen und Auswahl einer ORI werden dem Administrator die Details der ORI angezeigt. Hier kann er mit Klick auf einen Button, die ORI vom System entfernen. Anschließend werden alle Verbindungen zu anderen ORIs gelöscht.

Abschließen einer ORI (ORI4)
Nach dem Suchen und Auswahl einer ORI kann ein Administrator die ORI abschließen. Nach dem Abschluss wird die ORI archiviert.

Eine ORI suchen (ORI5)
Ein Administrator kann eine ORI mit der Suchfunktion, anhand der Attribute Rahmenname, Beschreibung und Ziel suchen. Wenn eine oder mehr ORIs den Vorgaben entsprechen wird eine Liste der Treffer angezeigt. Wird einer der Treffer in der Liste angeklickt, werden die Details der ORI angezeigt. Wird keine ORI zu den Vorgaben gefunden, wird eine Nachricht angezeigt.

A.1.4 CET-Verwaltung

Erstellen eines CET (CET1)
Ein Administrator kann einen neuen CET erstellen. Hierzu muss er dem CET einen Namen und kann zusätzlich einen Text und einen Status angeben.

Anpassen eines CET (CET2)
Nach dem Suchen und Auswahl eines CET kann ein Administrator den CET anpassen. Er kann den Namen, den Text und den Status ändern.

Entfernen eines CET (CET3)
Nach dem Suchen und Auswahl eines CET kann ein Administrator den CET, mit einem Klick auf einen Button, entfernen.

Einen CET suchen (CET4)
Ein Administrator kann einen CET mit der Suchfunktion, anhand der Attribute Namen und Text suchen. Wenn eine oder mehr CETs den Vorgaben entsprechen wird eine Liste der Treffer angezeigt. Wird einer der Treffer in der Liste angeklickt, werden die Details

des CET angezeigt. Wird kein CET zu den Vorgaben gefunden, wird eine Nachricht angezeigt.

A.1.5 CEI-Verwaltung

Instanziieren einer CEI (CEI1)
Ein Administrator kann, beim Erweitern einer CLI, neue CEI anhand einer existierenden CET instanziieren. Hierzu muss eine CET gesucht werden. Die Details des CET werden angezeigt und können angepasst werden. Mit einem Knopf kann der CET zu einer CEI instanziiert werden. Diese wird dann der Liste hinzugefügt.

Erstellen einer CEI (CEI2)
Ein Administrator kann, beim Erweitern einer CLI, neue CEI ohne Instanziierung eines CET erstellen. Hierzu kann er der CEI einen Namen, einen Text und einen Status angeben.

Anpassen einer CEI (CEI3)
Nach dem Suchen und Auswahl einer CEI kann ein Administrator die CEI anpassen. Er kann den Namen, den Text und den Status ändern.

Entfernen einer CEI (CEI4)
Nach dem Suchen und Auswahl einer CEI kann ein Administrator die CEI, mit einem Klick auf einen Button, entfernen.

Eine CEI suchen (CEI5)
Ein Administrator kann eine CEI mit der Suchfunktion, anhand der Attribute Namen, Text, Status und Abschluss suchen. Wenn eine oder mehr CEIs den Vorgaben entsprechen wird eine Liste der Treffer angezeigt. Wird einer der Treffer in der Liste angeklickt, werden die Details der CEI angezeigt. Wird keine CEI zu den Vorgaben gefunden, wird eine Nachricht angezeigt.

Abschluss einer CEI (CEI6)
Nach der Auswahl einer CEI, kann ein Administrator diese mit einem Klick abschließen.

A.1.6 CLT-Verwaltung

Erstellen eines CLT (CLT1)
Ein Administrator kann einen neuen CLT erstellen. Hierzu kann ein Name, eine Beschreibung und ein Status angegeben werden. Anschließend kann der Administrator CETs hinzufügen und Nutzer mit ihren zugehörigen Rollen angeben.

Anpassen eines CLT (CLT2)
Nach der Suchen und Auswahl eines CLT kann ein Administrator den CLT anpassen. Er kann den Namen, die Beschreibung und den Status verändern.

Entfernen eines CLT (CLT3)
Nach dem Suchen und Auswahl eines CLT kann ein Administrator den CLT, mit einem Klick auf einen Button, entfernen.

Einen CLT suchen (CLT4)
Ein Administrator kann einen CLT mit der Suchfunktion, anhand der Attribute Namen und Beschreibung suchen. Wenn eine oder mehr CLTl den Vorgaben entsprechen wird eine Liste der Treffer angezeigt. Wird einer der Treffer in der Liste angeklickt, werden die Details des CLT angezeigt. Wird kein CLT zu den Vorgaben gefunden, wird eine Nachricht angezeigt.

Erweiterte Operationen
Bewegen von CET (CLT5)
Ein Administrator kann die Position von CETs verändern.

Einfügen eines CLT (CLT6)
Ein Administrator kann CLTs anderen CLTs hinzufügen. Hierfür muss der Administrator eine weiteren CLT auswählen, eine Position selektieren und diese, mit einem Knopf, hinzufügen.

A.1.7 CLI-Verwaltung

Erstellen einer CLI (CLI1)
Ein Administrator kann eine neue CLI erstellen. Hierzu kann ein Name, eine Beschreibung und ein Status angegeben werden. Anschließend kann der Administrator CEI hinzufügen und Nutzer mit ihren zugehörigen Rollen angeben.

Instanziieren einer CLI (CLI2)
Ein Administrator kann, beim Erweitern einer ORI, neue CLI anhand eines existierenden CLT instanziieren. Hierzu muss ein CLT gesucht werden. Die Details des CLT werden angezeigt und können angepasst werden. Mit einem Knopf kann der CLT zu einer CLI instanziiert werden. Diese wird dann der ORI hinzugefügt.

Abschluss einer CLI (CLI3)
Nach der Auswahl einer CLI, kann ein Administrator sie mit einem Klick abschließen. Anschließend wird die CLI archiviert.

Anpassen einer CLI (CLI4)
Nach dem Suchen und Auswahl einer CLI kann ein Administrator die CLI anpassen. Er kann den Namen, die Beschreibung, und den Status anpassen. angezeigt.

Entfernen einer CLI (CLI5)
Nach dem Suchen und Auswahl einer CLI kann ein Administrator die CLI, mit einem Klick auf einen Knopf, entfernen.

Eine CLI suchen (CLI6)
Ein Administrator kann eine CLI mit der Suchfunktion, anhand der Attribute Namen, Beschreibung, Status und Abschluss suchen. Wenn eine oder mehr CLIs den Vorgaben entsprechen, wird eine Liste der Treffer angezeigt. Wird einer der Treffer in der Liste angeklickt, werden die Details der CLI angezeigt. Wird keine CLI zu den Vorgaben gefunden, wird eine Nachricht angezeigt.

Erweiterte Operationen

Bewegen von CEI (CLI7)
Ein Administrator kann die Position von CEI verändern.

Einfügen einer CLI (CLI8)

Ein Administrator kann CLIs anderen CLIs hinzufügen. Hierfür muss der Administrator eine weitere CLI auswählen, eine Position selektieren und diese mit einem Knopf hinzufügen.

A.1.8 Klassendiagramme der Entitäten des Datenmodells

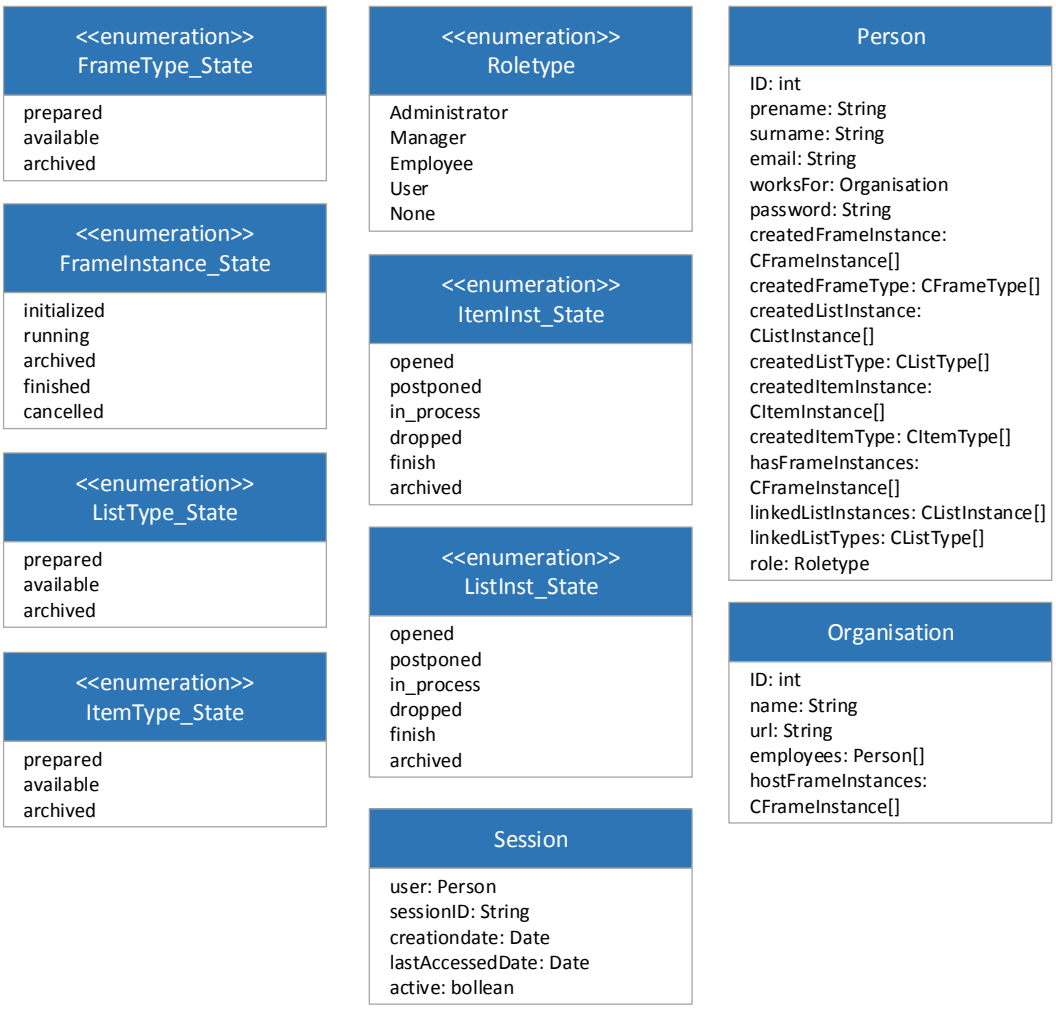


Abbildung A.1.: UML-Klassendiagramm der übrigen Entitäten des Datenmodells

Abbildungsverzeichnis

Literaturverzeichnis

[4Ch13] 4CHECKERS: *Checklisten und To Do Listen zu allen Themen kostenlos bei 4checkers.de*. `http://4checkers.de/`. Version: 2013

[Blu13] BLUECOAT: *Draft of FedEx MD-11 Checklist*. `http://www.bluecoat.org/reports/FedEx_97_MD11_Norm_Chklst.pdf`. Version: 2013

[BN84] BIRREL, Andrew D. ; NELSON, Bruce J.: *Implementing Remote Procedure Calls*. February 1984

[Bus96] BUSCHMANN, Frank: *Pattern-oriented software architecture / Frank Buschmann...*. Bd. [Vol. 1]: *A system of patterns*. Chichester [u.a.] : Wiley, 1996

[CC11] CATLIN, Hampton ; CATLIN, Michael L.: *Pragmatic guide to Sass*. Dallas and Tex : Pragmatic Bookshelf, 2011

[CF06] CLEVE, Jürgen ; FORBRIG, Peter: *Programmierung: Paradigmen und Konzepte : mit ... 85 Beispielen, 130 Aufgaben und Kontrollfragen und 19 Referatsthemen*. München [u.a.] : Fachbuchverl. Leipzig, 2006 (Lehr- und Übungsbuch Informatik)

[DD08] DEITEL, Paul J. ; DEITEL, Harvey M.: *Ajax, rich Internet applications, and web development for programmers*. Upper Saddle River and NJ : Prentice Hall, 2008 (Deitel developer series)

[Gaw09] GAWANDE, Atul: *The checklist manifesto: How to get things right*. New York : Picador, 2010, c2009

[Geo55] GEORGE A. MILLER: *http://spider.apa.org/ftdocs/rev/1994/april/rev1012343.html: The Magical Number Seven, Plus or Minus Two Some Limits on Our Capacity for Processing Information*. Bd. Vol. 101, No. 2. 1955 http://www.psych.utoronto.ca/users/peterson/psy430s2001/Miller%20GA%20Magical%20Seven%20Psych%20Review%201955.pdf

[Goo13a] GOOGLE ; GOOGLE (Hrsg.): *Google Keep*. https://drive.google.com/keep/. Version: 2013

[Goo13b] GOOGLE ; GOOGLE (Hrsg.): *GwtEvent (Google Web Toolkit Javadoc)*. http://www.gwtproject.org/javadoc/latest/com/google/gwt/event/shared/GwtEvent.html. Version: 2013

[GR01] GAMMA, Erich ; RIEHLE, Dirk: *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*. 5., korrigierter Nachdr. München and Boston [u.a.] : Addison-Wesley, 2001 (Programmer's choice)

[Grö11] GRÖNROOS, Marko: *Book of Vaadin*. 7th ed. Turku and Finland : Vaadin Ltd., 2011

[HC01] HEINEMAN, George T. ; COUNCILL, William T.: *Component-based software engineering: Putting the pieces together*. 1. Boston : Addison-Wesley, 2001

[HR13] HASLER ROUMOIS, Ursula: *UTB*. Bd. 2954: *Studienbuch Wissensmanagement: Grundlagen der Wissensarbeit in Wirtschafts-, Non-Profit- und Public-Organisationen*. 3. überarbeitete und erweiterte Auflage. Zürich : Orell Füssli, 2013

[Hub05] HUBE, Gerhard: *IPA-IAO-Forschung und -Praxis*. Bd. Nr. 422: *Beitrag zur Beschreibung und Analyse von Wissensarbeit. PhD thesis*. Heimsheim : Jost-Jetter, 2005 http://elib.uni-stuttgart.de/opus/volltexte/2005/2426/pdf/Diss_Hube_Wissensarbeit.pdf

[Joo13] JOOMLA! ; JOOMLA! (Hrsg.): *Joomla! The CMS Trusted By Millions for their Websites*. http://www.joomla.org/. Version: 2013

[lan13] LANG.COM sass: *Sass: Syntactically Awesome Style Sheets*. http://sass-lang.com/. Version: 2013

[MKR12] MUNDBROD, Nicolas ; KOLB, Jens ; REICHERT, Manfred: *Towards a System Support of Collaborative Knowledge Work*. 2012

[Mun] MUNDBROD, Nicolas: *Interner Bericht: Nutzung von Checklisten zur Unterstützung der Entwicklung mechatronischer Systeme im Automobilbereich, Universität Ulm, 2013*

[Nor02] NORMAN, Donald A.: *The design of everyday things*. 1st Basic paperback ed. [New York] : Basic Books, 2002

[Ora13] ORACLE ; ORACLE (Hrsg.): *javax.servlet (Java(TM) EE 7 Specification APIs)*. `http://docs.oracle.com/javaee/7/api/javax/servlet/package-summary.html`. Version: 2013

[Per04] PERRY, Bruce W.: *Java servlet and JSP cookbook*. 1st ed. Sebastopol and Calif : O'Reilly, 2004

[PS98] PFIFFNER, Martin ; STADELMANN, Peter D.: *Wissen wirksam machen: Wie Kopfarbeiter produktiv werden*. Bern [u.a.] : Haupt, 1998

[RR06] ROBERTSON, Suzanne ; ROBERTSON, James: *Mastering the requirements process*. 2nd ed. Upper Saddle River and NJ [u.a.] : Addison-Wesley, 2006

[SAK11] SWELLER, John ; AYRES, Paul L. ; KALYUGA, Slava: *Cognitive load theory*. New York : Springer, 2011 (Explorations in the learning sciences, instructional systems and performance technologies)

[TH13] TACY, Adam ; HANSON, Robert: *GWT in action: Easy Ajax with the Google Web Toolkit*. 2nd ed. Greenwich and Conn : Manning, 2013

[Wor13a] WORLD HEALTH ORGANIZATION ; WORLD HEALTH ORGANIZATION (Hrsg.): *WHO | Background to Safe Surgery Saves Lives*. `http://www.who.int/patientsafety/safesurgery/issue/en/index.html`. Version: 2013

[Wor13b] WORLD HEALTH ORGANIZATION ; WORLD HEALTH ORGANIZATION (Hrsg.): *WHO | Safe Surgery Saves Lives*. `http://www.who.int/patientsafety/safesurgery/en/`. Version: 2013

[wun13] WUNDERLIST ; WUNDERLIST (Hrsg.): *Organisiere dein Leben - Wunderlist*. https://www.wunderlist.com/de/. Version: 2013

Name: Norman Thiel Matrikelnummer: 702041

Erklärung

Ich erkläre, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Norman Thiel