Fachinformatiker für Anwendungsentwicklung

Dokumentation zur betrieblichen Projektarbeit

Automatisierte Accounterstellung via AMQP-Messaging-System

mit Konsolidierung der Datenquellen (Übergabeprotokoll und Angebotssystem)

Auszubildender: Rico Krüger
Beermannstr. 14
12435 Berlin
Tel.: 01766-4775045
andie.biller@gmail.com

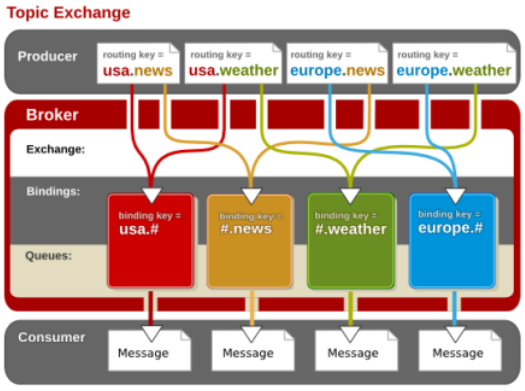

Abbildung 1: Interprozesskommunikation mit Messaging-Queues

Prüfungsausschuss: Tatjana Goebel, Ali Hafezi, Ralf Merettig
Abgabetermin: Berlin, den 15.06.2018

Berliner Synchron GmbH
EUREF-Campus 10-1, 10829 Berlin

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

HAML HTML abstraction markup language

HTML Hypertext Markup Language

IHK Industrie- und Handelskammer

1 Einleitung

1.1 Projektumfeld

Ausbildungsbetrieb: Die Berliner Synchron GmbH (BSG) ist ein mittelständisches, international-tätiges Unternehmen, dessen Kerngebiet die deutsche Synchronisation englischsprachiger Filme und Serien ist. 1949 als erstes Synchronunternehmen Deutschlands gegründet, hält die BSG nach wie vor eine Spitzenstellung innerhalb der Synchronbranche. Seit 2016 ist das Unternehmen Teil der S&L Medien Gruppe GmbH, die es sich zum Ziel gesetzt hat, ein allumfassendes Unterhaltungsmarketing zu bieten. Neben den rund 60 festen Mitarbeitern, greift die BSG auf einen Pool aus über 3 000 Synchronschauspielern, Autoren, Regisseuren, Cuttern und Takern zurück. Die interne IT-Abteilung des Unternehmens beschäftigt derzeit eine Vollzeitkraft und einen Auszubildenden. Unterstützt wird diese durch die IT-Abteilung der S&L Medien Gruppe sowie durch einen technischen Dienstleister vor Ort. Das Aufgabenfeld der IT-Abteilung umfasst den Aufbau und die Wartung des firmeninternen Netzwerks, die Administration der eigenen Server, die Entwicklung und Pflege der in der Firma eingesetzten Systeme sowie die alltägliche Benutzerunterstützung.

Auftraggeber: Der Auftraggeber ist in diesem Projekt der Ausbildungsbetrieb – die BSG.

Porjecktbeschreibung: Derzeit befindet sich eine neue Software – Copper in der Testphase. Diese wird von einem externen Dienstleister in Zusammenarbeit mit der BSG programmiert. Mit Copper sollen alle Tätigkeiten von der Projekterstellung über die Projektumsetzung bis hin zur Projektauswertung gesteuert und alle betrieblichen Prozesse über ein Webinterface zentralisiert dargestellt werden. Eine zusätzliche Webanwendung soll für die neue Software entwickelt werden. Diese Webanwendung soll ein Eingabeformular bereitstellen und die vom Benutzer eingegebenen Daten in eine MySQL-Datenbank speichern.

1.2 Projektziel

Projekthintergrund: Die Synchronisation von Filmen und Serien wird jeweils als ein Projekt bezeichnet. Die Akte eines Kinofilms bzw. die Episoden einer Serie bilden dabei einzelne Teilprojekte. Um diese Teilprojekte in deutscher Sprache aufzunehmen, erhält die Berliner Synchron GmbH von seinen Kunden, meist ausländische Produktionsstudios, sowohl das zu synchronisierende Video- als auch diverse Audiomaterialien. Dabei handelt es sich um unterschiedliche Audiotypen wie Originalton, Musik, Effekte und andere, von denen es wiederum unterschiedliche Versionen gibt. Bevor das gelieferte Material in den Produktionsprozess integriert werden kann, muss dieses analysiert werden. Dabei wird es auf Fehler überprüft und die Qualität des angelieferten Materials wird bewertet. Diese Bewertung kann dazu führen, dass das Audiomaterial überarbeitet oder sogar erneut vom Kunden zugesandt werden muss. Bisher wird das Ergebnis dieser Prüfung formfrei, meist in Word Dateien,

geschrieben. Danach werden diese in den entsprechenden Projektordnern gespeichert und die betreffenden Projektbeteiligten per E-Mail informiert.

Ziel des Projekts: Ziel dieses Projektes ist die Entwicklung einer einfachen und leicht bedienbaren Webanwendung zur Erfassung, Darstellung und zentralen Speicherung der Daten der Materialeingangsprüfung und deren direkter Zuordnung zu einem Teilprojekt. Alle Mitarbeiter sollen nach einer kurzen Einführung in der Lage sein, ein solches Formular auszufüllen.

1.3 Projektbegründung

Nutzen des Projekts: Durch ein standardisiertes Formular mit zum Großteil vorgegebenen Werten über Dropdown-Listen soll die Erstellung eines solchen Meldeberichtes erleichtert und damit beschleunigt werden. Dieses Projekt dient zudem einen weiteren wichtigen Aspekt: dem Controlling. Neben der Kostenerfassung für den Materialeingangsbericht durch Copper, lässt die zentralisierte Erfassung der Daten und deren direkter Zuordnung zu den Projekten und Kunden zu, eine übersichtliche und gefilterte Darstellung aller Materialeingangsberichte aus den gewonnen Daten zu generieren. Damit lassen sich Rückschlüsse auf anfallende Kosten für zukünftige Projekte ziehen. Das Erstellen eines solchen Berichts ist allerdings nicht Teil dieses Projekts und wird erst in einem Nachfolgeprojekt entwickelt. Die Erstellung und Speicherung der Daten eines Materialeingangs stellt lediglich die Grundlage dar.

Motivation: Der Auftraggeber ist daran interessiert, ein allumfassendes System für die Projektabwicklung zu haben, welches alle Projektschritte dokumentiert und darstellt. Dazu wurde die Software Copper in Auftrag gegeben, welche durch einen externen Dienstleister entwickelt und nach erfolgreicher Einführung von der eigenen IT-Abteilung erweitert werden soll.

1.4 Projektschnittstellen

Copper ist eine Webapplikation, die mit der Programmiersprache Ruby und dem Framework Rails erstellt wurde. Da diese Anwendung später auf einen firmeneigenen Server ausgeführt und hauptsächlich im lokalen Netzwerk eingesetzt wird, soll diese vor allem über den in der Berliner Synchron eingesetzten Webbrowser – Google Chrome ausführbar sein. Zur Erstellung der Webseite wird das Framework Bootstrap verwendet. Die browserbasierten Nutzereingaben werden dann mittels http von einem Apache-Webserver entgegen genommen und an die Rails-Applikation übermittelt. Dort werden die Anfragen bearbeitet und beantwortet. Die Daten werden mittels eines in Rails integrierten MySQL-Connectors in eine lokale relationale MySQL-Datenbank gespeichert und abgerufen. Die Anwendung ist vor allem für die QC-Abteilung im Unternehmen gedacht. Die Verantwortliche Mitarbeiterin sowie der Ausbilder sind für die Projektabnahme verantwortlich.

1.5 Projektabgrenzung

Was dieses Projekt nicht bietet: Eine automatisierte Auswertung sowie sonstige Zusammenfassungen außerhalb eines Teilprojektes werden nicht vorgenommen. Dieses Projekt stellt lediglich die Grundlage dar. Zudem wird die Software während des Projektes noch nicht in Copper integriert. (Siehe Abweichung vom Projektantrag).

2 Projektplanung

Der Projektplanung ging ein Meeting zwischen der Geschäftsführung, einer Mitarbeiterin, welche zum Großteil Materialeingangsberichte erfasst und der IT-Abteilung voraus. Nachdem die Geschäftsführung sich nach einer Zusammenfassung der Materialeingangsprüfungsberichte für einen bestimmten Kunden erkundigte bzw. nach der Möglichkeit einer solchen, wurde schnell klar, dass dies auf Basis der aktuellen Arbeitsweise nicht gewährleistet werden kann. Für eine übersichtliche, gefilterte Darstellung solcher Berichte würde eine gewisse Standardisierung und zentrale Speicherung der Daten benötigt. Schnell kam man dann auf die Idee, dies gleich mit der Einführung der neuen Software – Copper und meiner Projektarbeit zu verknüpfen. Nach Absprache mit dem Entwickler von Copper wurde dann der Projektantrag bei der IHK eingereicht.

2.1 Projektphasen

Von der Industrie- und Handelskammer (IHK) wurden für die Bearbeitung des Projektes 70 Arbeitsstunden vorgegeben. Daher wurde das Projekt in ein Teilprojekt gewandelt. Nach der positiven Rückmeldung auf mein Projektantrag wurde das Projekt in den Tagesablauf meiner betrieblichen Arbeit integriert und mit ca. 30 Wochenstunden bearbeitet.

2.2 Zeitplanung

Für die Umsetzung des Projektes stehen Seitens der Anforderungen der IHK 70 Stunden zur Verfügung. Diese wurden zur Antragstellung auf die einzelnen Phasen verteilt. Die grobe Zeitplanung der Hauptphasen kann der Tabelle 1 Zeitplanung aufs dieser Seite entnommen werden. Eine ausführlichere Zeitplanung findet sich im Anhang A.2: Zeitplanung auf Seite ii.

2.3 Abweichungen vom Projektantrag

Bei der Erstellung des Projektantrags wurde geplant, das Projekt auf einer bereits vorhandenen Testumgebung aufzusetzen und in Absprache mit dem Entwickler von Copper anzufertigen. Da der Entwickler von Copper während der Projektarbeit weder erreichbar, noch ein Zugriff auf die Testumgebung vorhanden war, musste das Projekt mit den zur Verfügung stehenden Daten zur Zeit der Erstellung

Projektphase	Geplante Zeit
Analyse	6 h
Entwurf	14 h
Implementierung	37 h
Abnahme und Einführung	2 h
Dokumentation	11 h
Gesamt	**70 h**

Tabelle 1: Zeitplanung

des Projektantrags nachgebaut werden. Somit ergaben sich erhebliche Änderungen im Vergleich zum Projektantrag. Ferner konnte aufgrund der am Ende fehlenden Zeit - durch die vorhergehenden, nicht eingeplanten Entwicklungsarbeiten - keine automatisierten Tests des entwickelnden Materialeingangsberichts mehr durchgeführt werden. Dies hätte das zusätzliche Aufsetzen einer Testumgebung bedurft, welche am Ende nicht einmal den tatsächlichen Produktionseinsatz der Anwendung sichergestellt hätten. Daher wurde in Absprache mit dem Ausbilder beschlossen lediglich ein Testprotokoll zu erstellen. Weiterhin wurde auf die Erstellung einer Entwicklerdokumentation verzichtet.

2.4 Ressourcenplanung

Alle benötigten Ressourcen zur Durchführung des Projekts sind bereits im Unternehmen vorhanden bzw. können online heruntergeladen werden. Eine Auflistung befindet sich im Anhang A.1: Projekt-Ressourcen auf Seite i.

2.5 Entwicklungsprozess

Die Entwicklung des Projekts erfolgt nach Rücksprache mit der verantwortlichen Mitarbeiterin für Qualitätsprüfung im Unternehmen. Ansonsten wird das Projekt weitestgehend nach dem Wasserfallmodell in Eigenregie erstellt.

3 Analysephase

Die erste Analysephase wurde bereits zum Zeitpunkt der Antragstellung des IHK-Projektes durchgeführt. Dort wurde bereits mit dem externen Entwickler gesprochen und auf der Grundlage eines SQL-Scripts ein vorläufiges Datenbankmodell für das Projekt erstellt und eingegrenzt.

3.1 Ist-Analyse

In der Berliner Synchron GmbH wird sowohl das eingehende als auch das ausgehende Audio- und Videomaterial analysiert und auf Fehler überprüft. Bei der Analyse werden neben möglichen Fehlern auch Meta-Daten wie das Materialeingangsdatum, Audioformat oder die Version gesammelt. Zudem wird entschieden, ob das Material nochmal überarbeitet oder gar vom Kunden neu übermittelt werden muss, bevor es in die Produktion geht. Eine solche Prüfung wird bisher formfrei, meist in Word-Dateien geschrieben. Danach werden die für das Projekt verantwortlichen Mitarbeiter per Email über die Fertigstellung und auf mögliche Mängel hingewiesen. Bei einem solchen Bericht gibt es viele Informationen, welche nur einen begrenzt möglichen Datensatz aufweisen. Weiterhin werden dann die Berichte in den entsprechenden Netzwerkordnern der einzelnen Teilprojekte gespeichert. Das Erhalten eines Überblicks über die Materialeingänge und deren Fehler für ein Teilprojekt ist somit äußerst schwierig. Will man einen solchen Überblick über ein komplettes Projekt oder gar für einen Kunden erhalten, ist dies nahezu unmöglich. Dies erschwert die Kostenberechnung und das Ziehen von Rückschlüssen für künftige Projekte.

3.2 Wirtschaftlichkeitsanalyse

Die Wirtschaftlichkeit des neuen standardisierten Berichts zur Qualitätssicherung steht außer Frage. Die derzeitige Darstellung und Bearbeitung ist unübersichtlich und ineffizient. Die Vielzahl der Erstellung von Berichten würde sich in jedem Fall mittelfristig gesehen, refinanzieren. Mit der Inbetriebnahme der neuen Software sollen zudem alle übrigen Projektprozesse dargestellt und deren Kosten ermittelt werden. Warum also die Erstellung eines solchen Berichts ausschließen? Das würde wiederum das Controlling erschweren und auch für diese Kostenstelle einen zusätzlichen Zeit- und damit Kostenaufwand bedeuten.

3.2.1 „Make or Buy“-Entscheidung

Da die Synchronbranche sehr speziell ist und es keine standardisierte Arbeitsweise oder Projektsoftware für diesen Bereich gibt, entschied sich die BSG einen externen Dienstleister mit der Entwicklung einer Software, zugeschnitten auf die Bedürfnisse des Unternehmens, zu engagieren. Von Anfang an war klar, dass dieses Projekt, selbst nach Einführung, von der unternehmenseigenen IT-Abteilung stetig weiterentwickelt werden soll und muss. Da die neue Software noch in den Händen des Entwicklers ist, stellt sich nur die Frage, ob dieser den neuen Bericht erstellt oder der firmeneigene Mitarbeiter/Auszubildende. Aus Kostengesichtspunkten ist der Auszubildende für das Unternehmen am geeignetsten.

3.2.2 Projektkosten

Die Kosten zur Durchführung des Projekts ergeben sich aus den Gehältern der beteiligten Mitarbeiter, die für die Mitarbeiter aufzuwendenden Sozialabgaben, sowie die Gemeinkosten sonstiger Res-

sourcen(Footnote). Sonstige Ressourcen wie Internet und Strom werden dabei vernachlässigt, da diese Kosten ohnehin anfallen. Bei 20 Arbeitstagen monatlich erhält der Auszubildende 884 € brutto. Der Arbeitgeberanteil zur Sozialversicherung beträgt dabei 229,53 €. Für den Personalaufwand der übrigen Mitarbeiter wird mit einer Pauschale von 30 € / Stunde gerechnet.

$$8\ \text{h/Tag} \cdot 20\ \text{Tage/Monat} = 160\ \text{h/Monat} \tag{1}$$

$$\frac{884{,}00\,€ + 229{,}53\,€/\text{Monat}}{160\ \text{h/Monat}} = \frac{1113{,}53\,€/\text{Monat}}{160\text{h/Monat}} = 6{,}96\,€/\text{h} \tag{2}$$

Es ergibt sich ein Stundenlohn von 6,96 €. Die Durchführungszeit des Projekts beträgt 70 Stunden. Die Nutzung von Ressourcen[1] wird hier mit einem pauschalen Stundensatz von 10 € berücksichtigt. Eine Aufstellung der Kosten befindet sich in Tabelle 2. Die Kosten belaufen sich auf insgesamt 1397,20 €.

Vorgang	Zeit	Kosten pro Stunde	Kosten
Entwicklungskosten	70 h	6,96 €	388,08 €
Fachgespräche	1 h	30 €	30 €
Meeting QC - Mitarbeiterin	2 h	30 € · 2	60 €
Benutzerschulung	0,5 h	4 x 30 €	60 €
Ressourcen	70 h	10 €	700 €
Kosten			**1238,08 €**

Tabelle 2: Projektkosten

Es ergibt sich demnach ein Stundensatz von 6,96 €. Die Durchführungszeit des Projekts beträgt 70 Stunden. Für die Nutzung der Ressourcen wird ein pauschaler Stundensatz von 10€ herangezogen. Nachfolgender Tabelle lässt sich eine detaillierte Kostenübersicht entnehmen.

3.2.3 Amortisationsdauer

Geht man von einer Zeitersparnis bei der Erstellung von 10 Minuten pro Bericht aus, bei zwei Berichten pro Tag, ergibt sich eine durchschnittliche monatliche Zeitersparnis von 400 Minuten über 6,6 Stunden. Bei der veranschlagten Mitarbeiterpauschale ergibt sich eine monatliche Kostenersparnis von 200€. Damit wäre dieses Projekt nach knapp über 6 Monaten amortisiert. Die tatsächliche Amortisationsdauer des Projektes ist zum jetzigen Zeitpunkt noch nicht absehbar, da es nur den ersten Teil des Projekts zur Qualitätssicherung darstellt. Dies stellt wiederum auch nur einen Teil des viel größeren Projektes – Copper dar.

[1] Räumlichkeiten, Arbeitsplatzrechner, Lizenzen, etc.

3.3 Nutzwertanalyse

Neben der Zeitersparnis und Minimierung der Fehleranfälligkeit liegt der größte Nutzen des Projekts in der Standardisierung der Berichte, sowie der späteren Integration in Copper. Auf dieser Grundlage lassen sich die tatsächlich aufgetreten Projektkosten noch genauer kalkulieren und es ist mögliche Rückschlüsse auf Kosten zukünftiger Projekte zu ziehen.

3.4 Qualitätsanforderungen

Die Webanwendung soll intuitiv und leicht bedienbar für die im Tonbereich qualifizierten Mitarbeiter sein. Schlüsseldatensätze für spätere Auswertungen sollen auswählbar sein. Zudem muss es für etwaige Fehler Kommentarfelder für genauere Erläuterungen geben. Die Berichte müssen sich eindeutig unterscheiden und einem Teilprojekt zugeordnet sein. Es muss möglich sein, sich die erstellten Berichte anzusehen und diese gegebenenfalls zu editieren. Zudem muss es eine Übersicht über die erstellten Berichte eines Teilprojekts geben. Die komplette Ansicht ist über Google-Chrome skalierbar.

3.5 Lastenheft

Nachfolgend werden die wichtigsten Anforderungen für das Projekt dargestellt.

Anforderungen an die Benutzungsoberfläche:

LB10: Die Berichte müssen online über den Chrome-Webbrowser erreichbar sein.
LB20: Die Ansicht muss skalierbar sein.
LB30: Die Bedienung und Formularfelder müssen intuitiv und selbsterklärend sein.
LB40: Sich wiederholende standardisierte Datensätze müssen auswählbar sein.
LB50: Es muss möglich sein zu diversen Datensätze Kommentare zu schreiben.
LB60: Erstellte Berichte müssen editierbar sein.
LB70: Für die erstellten Berichte muss es eine Anzeige geben.
LB80: Es muss eine Übersicht aller Berichte eines Teilprojekts geben.

Funktionelle Anforderungen:

LF10: Die Berichte müssen eindeutig voneinander unterschieden werden können.
LF20: Die Navigation zu den Berichten muss vorhanden sein.

Sonstige Anforderungen:

LS10: Die Anwendung muss anhand der vorhandenen Information in Copper integrierbar sein.
LS20: Das Projekt muss für das Anschlussprojekt erweiterbar sein.

3.6 Zwischenstand

Tabelle 3 zeigt den Zwischenstand nach der Analysephase.

Vorgang	Geplant	Tatsächlich	Differenz
Durchführung einer Ist-Analyse	2 h	3 h	+1 h
Wirtschaftlichkeitsanalyse und Amotisationsrechnung	1 h	1 h	
Erstellung eines Lastenhefts	3 h	3 h	
Analysephase	**6 h**	**7 h**	**1 h**

Tabelle 3: Zwischenstand nach der Analysephase

4 Entwurfsphase

Die meisten, jedoch nicht alle, Entscheidungen über verwendete Technologien stehen bereits mit Projektbeginn fest. Denn sie sind von der sich in der Entwicklung befindenden Software – Copper abhängig. In der Entwurfsphase ist es das Ziel, anhand der zur Verfügung stehenden Informationen ein Designplan für die Anwendung zur späteren Integration in Copper zu entwickeln.

4.1 Zielplattform

Die Ruby on Rails Webanwendung soll unter dem Apple-Betriebssystem OS X laufen. Die Daten werden eine lokal installierte MySQL-Datenbank gespeichert. Die Anbindung zur Datenbank erfolgt über den mysql2 gem, einer Ruby Bibliothek. Zum lokalen Aufrufen und Testen der Applikation über den Chrome-Browser wird der Thin-Webserver verwendet. Die Realisierung der skalierbaren Webanwendung wird über die Auszeichnungssprachen Haml, CSS und dem CSS-Framework Bootstrap sichergestellt.

4.2 Architekturdesign

Als Architekturdesign wird das in Rails üblicherweise eingesetzte Model View Controller (MVC) – Pattern genutzt. Dabei repräsentieren die sich im Model befindlichen Klassen die Datenbanktabellen. Die interne Kommunikation zwischen den Modell-Klassen, der Datenhaltungsschicht, und der MySQL-Datenbank wird dann vom mysql2 gem übernommen. Die browserbasierten Darstellungen und Formulare, die GUI-Schicht, werden in der View repräsentiert. Die Kommunikation und Logik zwischen View und dem Model übernimmt der Controller. Die ankommenden HTTP-Anfragen und Parameter werden über entsprechende Routen an den Controller weitergeleitet, welcher dann anhand dieser die sogenannten Create Read Update Delete (CRUD) – Operationen durchführt.

4.3 Entwurf der Benutzungsoberfläche

Da die Anwendung primär von ausgebildetem Fachpersonal in der Tonbranche verwendet wird, können auch Fachbegriffe oder Abkürzungen verwendet werden. Die Bewertung des Materials sollte jedoch für jedermann verständlich sein. Die Skalierbarkeit der Anwendung wird durch Bootstrap realisiert. Da die Anwendung von internem Personal genutzt und in ein anderes Projekt eingepflegt werden soll, wird auf ein ausgeklügeltes, repräsentatives Design verzichtet. Für Eingabefelder mit begrenzten Werten werden Dropdown-Menüs erzeugt. Zur eindeutigen Kennzeichnung jeder Eingabe werden Label verwendet. Die Datumseingabe erfolgt mithilfe eines Datumswählers. Layout Siehe Anhang Screenshots der Mockups für die neuen Felder in der Eingabemaske findet sich im Anhang A.3: Mockup auf Seite iii.

4.4 Datenmodell

Während der Vorbereitung des Projektantrags ließ mir der externe Entwickler einen kleinen Ausschnitt der Datenbank in Form eines MySQL-Skripts zukommen. Auf dieser Grundlage muss der bestehende Ausschnitt der Datenbank mit den entsprechenden Namenskonventionen erweitert werden. Dabei werden die benötigten Tabellen über Rails migriert. Es werden Tabellen für alle auszufüllenden Felder mit begrenzten Datensätzen erstellt. Eine Tabelle für den Materialeingangsbericht sammelt die eingegebenen Daten und ordnet diese der Tabelle des Teilprojekts zu, indem sie die entsprechenden Datensätze der anderen Tabellen referenziert. Für jede Tabelle muss in Rails eine Klasse angelegt werden. Dort müssen die Eingaben dann validiert werden, bevor sie tatsächlich in die Datenbank geschrieben werden.

4.5 Fachkonzept

Für die Umsetzung der Logik muss jeweils ein Controller für ein Projekt und ein angelegt werden, um die grundlegende Struktur von Copper zu simulieren. Dabei beinhalten diese nur die grundlegendsten Methoden, welche es mir gestatten sollen, mich über den Browser zu meinen Views zu navigieren und den zu erstellenden Controller für die Berichte anzusprechen. Auf diesen werden dann die CRUD-Operationen definiert.

Secretary, Practice, Agenda, Doctor

4.6 Maßnahmen zur Qualitätssicherung

Während der Bearbeitung des Projekts wurde die Funktionalität der Anwendung kontinuierlich getestet. Dabei wurden die Links ausgeführt, die Routen getestet und nach Eingaben die Datenbank auf die Richtigkeit der Werte geprüft. In Zusammenarbeitet mit der QM wird ein für die Abnahme erforderliches Testprotokoll erstellt.

4.7 Pflichtenheft

Nachfolgend werden die wichtigsten vereinbarten Pflichten des Projekts aufgelistet, um eine Standardisierung der Eingangsberichte, die Integrierbarkeit in Copper sowie die Erweiterbarkeit für die Erstellung späterer Auswertungen zu gewährleisten.

Pflichten der Benutzungsoberfläche:

PB10: Es muss eine Webanwendung programmiert und ständig mit Chrome getestet werden.
PB20: Es muss Bootstrap verwendet und die Skalierbarkeit ständig über den Webbrowser getestet werden.
PB30: Jede Eingabe muss ein Label erhalten. Die Namensbezeichnung erfolgt in Absprache mit der zuständigen Mitarbeiterin für Qualitätssicherung.
PB40: Es muss Dropdown-Menüs und Checkboxen geben.
PB50: Es muss Textfelder und Textareas geben. Diese müssen optisch zu den Dropdown-Menüs und Checkboxen zugeordnet werden.
PB60: Es muss eine initiale tabellarische Anzeige der Berichte nach Auswahl des Teilprojekts geben. Dort müssen der Audiotyp, die Version, die Bewertung, sowie Links zum Anzeigen, Bearbeiten und Löschen erscheinen.
PB70: Es muss ein Webformular mit den o.g. Konventionen geben.
PB80: Es muss einen Link zum Bearbeiten geben, welcher das unter PB70 definierte Formular mit den entsprechenden Datensätzen lädt.
PB90: Es gibt einen Link zur Anzeige, welches das unter PB70 definierte Formular mit deaktivierten Eingabefelder lädt.

Funktionelle Pflichten:

PF10: Berichte mit gleichem Teilprojekt, Audiotyp und Audioformat müssen eine unterschiedliche Versionsnummer haben.
PF20: Über die Auswahl der Teilprojekte muss die Auswahl eines jeden zugehörigen Berichts erfolgen.

Funktionelle Pflichten:

PS10: Alle bekannten Informationen über Copper vom Betriebssystem bis zu den Versionen der Frameworks und Bibliotheken werden verwendet.
PS20: Die Daten werden in eine relationale MySQL-Datenbank gespeichert.

4.8 Zwischenstand

Tabelle 4 zeigt den Zwischenstand nach der Entwurfsphase.

Vorgang	Geplant	Tatsächlich	Differenz
Entwurf der Projektumgebung	1 h	1 h	
Entwurf der Benutzeroberflächen	3 h	4 h	+1 h
Entwurf der erweiterten Datenbank	4 h	4 h	
Erstellung eines Testprotokolls	2 h	2 h	
Erstellung des Pflichtenhefts	4 h	4 h	
Entwurfsphase	**14 h**	**15 h**	**+1 h**

Tabelle 4: Zwischenstand nach der Entwurfsphase

5 Implementierungsphase

Bevor mit der eigentlichen Programmierung und Implementierung begonnen werden kann, muss direkt am Anfang eine dazugehörige Entwicklungsumgebung aufgesetzt werden. Dies war zum Zeitpunkt des Projektantrags nicht bekannt. Während der gesamten Dauer das Projekts wurde regelmäßig über die Versionsverwaltungssoftware Github gesichert.

5.1 Implementierung der Projektumgebung

Nach einem Backup des bisherigen Systems auf einem MacBook Pro, werden Ruby, Rails und alle weiteren benötigten Frameworks und Bibliotheken mit den bekannten Informationen eingebunden und deren Funktionstüchtigkeit getestet. Der MySQL-Server, Ruby Version Manager (rvm), über den Ruby installiert wird, sowie das Datenbank-Management System Sequel Pro werden mit Hilfe des OS X Paket Managers Homebrew installiert. Über den Bash rails new app wird ein neues Projekt erstellt. Als Entwicklungsumbegung (IDE) wird Rubymine verwendet.

5.2 Implementierung des Webservers und des Routings

Als Web Server wird Thin verwendet, welcher leicht als RubyGem über Ruby installiert werden kann und keiner weiteren Konfiguration bedarf. In der Projektdatei config/routes.rb werden die Routen zum Controller mit den entsprechenden Operationen definiert.

5.3 Implementierung der Datenbank

Nach der Erstellung eines Projekts wird über den simplen Befehl rake db:create die Datenbank erstellt werden und danach das vom externen Entwickler erhaltene MySQL-Skript importiert. Migrationen für die neuen Tabellen werden in Rails geschrieben und über rake db:migrate der Datenbank hinzugefügt. Das hat den Vorteil, dass man jederzeit seine selbst erstellten Migrationen zurücksetzen, diese editieren und der Datenbank wieder hinzufügen kann. Über ein erstelltes MySQL-Skript werden dann die Daten für die Tabellen der Dropdown Menüs importiert.

5.4 Implementierung der Models

Für jede Tabelle der Datenbank wird eine Klasse erstellt. Dies geschieht über den Befehl rails generate model [Tabellenname]. Dabei ist darauf zu achten, dass man sich an den Namenskonventionen von Rails orientiert. Rails kann so die entsprechenden Modelle der jeweiligen Tabelle automatisch zuordnen und die Datenbankabfragen generieren. Danach werden in den Model-Klassen die Relationen der Klassen nach dem Vorbild der Tabellen untereinander definiert und Operationen zur Gültigkeitsprüfung der an die Tabelle zu übergebenen Daten geschrieben.

5.5 Implementierung des Fachkonzepts

Über den Befehl rails generate controller [Controllername] wird eine neue Klasse für die entsprechenden Controller erstellt. Entsprechend der Namenskonventionen werden durch Rails zusätzliche Hilfsdateien, sowie einen Pfad zur View des Controllers angelegt. Anhand der Methodennamen erfolgt das Routing zu dessen Operationen. Des Weiteren wird anhand der Rails-Namenskonvention auch die entsprechende Ansicht festgelegt. Hier werden die entsprechenden CRUD – Operationen implementiert.

5.6 Implementierung der Benutzeroberfläche

Für die Ansicht der Projekte und Teilprojekte werden nur die nötigsten Dateien und Links für die Weiterleitung zum Materialeingangsbericht implementiert. Die Ansichten für den Bericht werden mit der HTML – Abstraktionssprache HAML geschrieben. Über die Index-Datei des Berichts werden alle zu dem Teilprojekt aufgelisteten gespeicherten Berichte angezeigt und zu den entsprechenden Ansichten (Anzeigen, Bearbeiten, Löschen) verlinkt. Zudem wird ein Link zur Erstellung eines neuen Berichts generiert. Für das Anzeigen, Bearbeiten und Erstellen wird jeweils das gleiche „Partial“ – Formular mit einem Parameter für das Aktivieren bzw. Deaktivieren geladen. Die Datenauswahl in den Dropdown-Menüs werden über Rails geladen.

5.7 Zwischenstand

Tabelle 5 zeigt den Zwischenstand nach der Implementierungsphase.

6 Qualitätssicherung und Abnahme

Die Abnahme erfolgt durch die zuständige Mitarbeiterin für Qualitätssicherung (QM) sowie den Ausbilder. Dabei werden erneut die Daten aus dem Testprotokoll eingegeben. Die Aus- und Eingabe sowie das äußere Design wird durch die QM-Verantwortliche geprüft, wohingegen die technischen Spezifikationen für Copper durch den Ausbilder geprüft werden.

Vorgang	Geplant	Tatsächlich	Differenz
Implementierung der Projektumgebung	6 h	9 h	+3 h
Implementierung der Datenbank und Migrationen	8 h	6 h	-2 h
Implementierung der Models	6 h	6 h	
Implementierung der Logik für Darstellung und Datenhaltung	8 h	7 h	-1 h
Implementierung der Views	9 h	10 h	+1 h
Implementierungsphase	**37 h**	**38 h**	**+1 h**

Tabelle 5: Zwischenstand nach der Implementierungsphase

6.1 Zwischenstand

Tabelle 6 zeigt den Zwischenstand nach der Abnahmephase.

Vorgang	Geplant	Tatsächlich	Differenz
Qualitätssicherung und Abnahme	1 h	1 h	
Abnahme	**1 h**	**1 h**	

Tabelle 6: Zwischenstand nach der Abnahmephase

7 Einführungsphase

7.1 Geplante Einführung

Die tatsächliche Inbetriebnahme der Anwendung kann derzeit noch nicht erfolgen, da dieses Projekt abhängig von der Integration in Copper ist.

7.2 Benutzerschulung

Einführung bei den Mitarbeitern Den Sounddesignern wird das Projekt demonstriert und die Bedienung erklärt.

7.3 Zwischenstand

Tabelle 7 zeigt den Zwischenstand nach der Einführungsphase.

Vorgang	Geplant	Tatsächlich	Differenz
Benutzerschulung	1 h	1 h	
Einführung	**1 h**	**1 h**	

Tabelle 7: Zwischenstand nach der Einführungsphase

8 Dokumentation

Diese Projektdokumentation wurde im Rahmen der IHK - Abschlussprüfung zum IT-Fachinformatiker für Anwendungsentwicklung mit LATEX erstellt.

8.1 Benutzerdokumentation

Eine vorläufige Benutzerdokumentation mit entsprechenden Hinweisen für die Bedienung wurde mit Microsoft-Word geschrieben und in einen für Handbücher vorgesehenen Netzwerkordner im PDF-Format gespeichert. Anhang A.5: Benutzerdokumentation auf Seite xii.

8.2 Zwischenstand

Tabelle 8 zeigt den Zwischenstand nach der Dokumentation.

Vorgang	Geplant	Tatsächlich	Differenz
Erstellung der Projektdokumentation	10 h	12 h	+2 h
Erstellen der Benutzerdokumentation	1 h	1 h	
Dokumentation	**11 h**	**13 h**	

Tabelle 8: Zwischenstand nach der Dokumentation

9 Fazit

9.1 Soll-/Ist-Vergleich

Auch wenn ich mir den Projektablauf und das Ergebnis zum Zeitpunkt der Antragsstellung gänzlich anders vorgestellt habe, sind die Kollegen mit dem neuen Formular sehr zufrieden und würden es gerne schon in Betrieb nehmen. Die Zeitplanung weicht, insgesamt betrachtet, geringfügig ab. Betrachtet man jedoch die einzelnen Teile, fällt auf, dass es nicht in Gänze möglich war vorher zu sehen wie groß der tatsächliche Aufwand für die einzelnen Teile ist. Tabelle 9 zeigt einen Vergleich der veranschlagten sowie tatsächlich aufgewendeten Zeit.

Phase	**Geplant**	**Tatsächlich**	**Differenz**
Analysephase	*6 h*	*7 h*	*+1 h*
Entwurfsphase	*14 h*	*15 h*	*+1 h*
Implementierungsphase	*37 h*	*38 h*	*+1 h*
Abnahme	*1 h*	*1 h*	
Einführung	*1 h*	*1 h*	
Dokumentation	**11 h**	**13 h**	
Gesamt	**70 h**	**75 h**	**+5 h**

Tabelle 9: Soll-/Ist-Vergleich

Eidesstattliche Erklärung

Ich, Rico Krüger, versichere hiermit, dass ich meine **Dokumentation zur betrieblichen Projektarbeit** mit dem Thema

> *Automatisierte Accounterstellung via AMQP-Messaging-System mit Konsolidierung der Datenquellen (Übergabeprotokoll und Angebotssystem)*

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, wobei ich alle wörtlichen und sinngemäßen Zitate als solche gekennzeichnet habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Berlin, den 15.06.2018

_____

Rico Krüger

A Anhang

A.1 Projekt-Ressourcen

Projekt-Ressourcen

1. Beteiligte Personen

Andreas Biller, Auszubildender, Software Developer, Doctena Germany GmbH
André Rauschenbach, Ausbilder, CTO, Doctena Germany GmbH
Philipp Vaßen, Onboarding-Manager, Doctena Germany GmbH

2. Hardware

Ein Development-Rechner (Intel NUC i5, 8 GB RAM, 500 GB SSD, OS: Ubuntu 16.04)
Ein Bildschirm, 24 Zoll (Dell)
Benötigte Peripherie (Eine Maus, eine Tastatur, Drucker, Fax, etc.)
Ein VOIP-Telefon

3. Software, Frameworks, Services

Github, Rubymine, Circle-Ci, Ruby, Rails, Jira, Confluence, Minitest, Bootstrap, Heroku, AWS, MongoLab, RabbitMQ, Okta, Google, Firefox, Ubuntu, Windows, LaTeX, TexStudio, Balsamiq, etc.

4. Sonstige Ressourcen

Ein Bildschirmarbeitsplatz in Büroräumen in der Prinzessinnenstraße 20 A, 10969 Berlin, Internet, Strom, Heizung, Wasser, eine Küche, Toiletten, Verkabelung, usw.

Azubi: Andreas Biller - Azubinummer: 3593566
Ausbilder: André Rauschenbach - Doctena Germany GmbH - Urbanstr. 116 - 10967 Berlin

1 / 1

A.2 Zeitplanung

Vorgang	Geplant	Tatsächlich	Differenz
Durchführung einer Ist-Analyse	2 h	3 h	+1 h
Wirtschaftlichkeitsanalyse und Amotisationsrechnung	1 h	1 h	
Erstellung eines Lastenhefts	3 h	3 h	
Analysephase	**6 h**	**7 h**	**+1 h**
Entwurf der Projektumgebung	1 h	1 h	
Entwurf der Benutzeroberflächen	3 h	4 h	+1 h
Entwurf der erweiterten Datenbank	4 h	4 h	
Erstellung eines Testprotokolls	2 h	2 h	
Erstellung des Pflichtenhefts	4 h	4 h	
Entwurfsphase	**14 h**	**15 h**	**+1 h**
Implementierung der Projektumgebung	6 h	9 h	+3 h
Implementierung der Datenbank und Migrationen	8 h	6 h	-2 h
Implementierung der Models	6 h	6 h	
Implementierung der Logik für Darstellung und Datenhaltung	8 h	7 h	-1 h
Implementierung der Views	9 h	10 h	+1 h
Implementierungsphase	**37 h**	**38 h**	**+1 h**
Qualitätssicherung und Abnahme	1 h	1 h	
Qualitätssicherung und Abnahme	**1 h**	**1 h**	
Einführungsphase	1 h	1 h	
Einführungsphase	**1 h**	**1 h**	
Erstellen der Projektdokumentation	10 h	12 h	+2 h
Erstellung der Benutzerdokumentation	1 h	1 h	
Dokumentation	**11 h**	**13 h**	**+2 h**
Gesamt	**70 h**	**75 h**	**+5 h**

A.3 Mockup

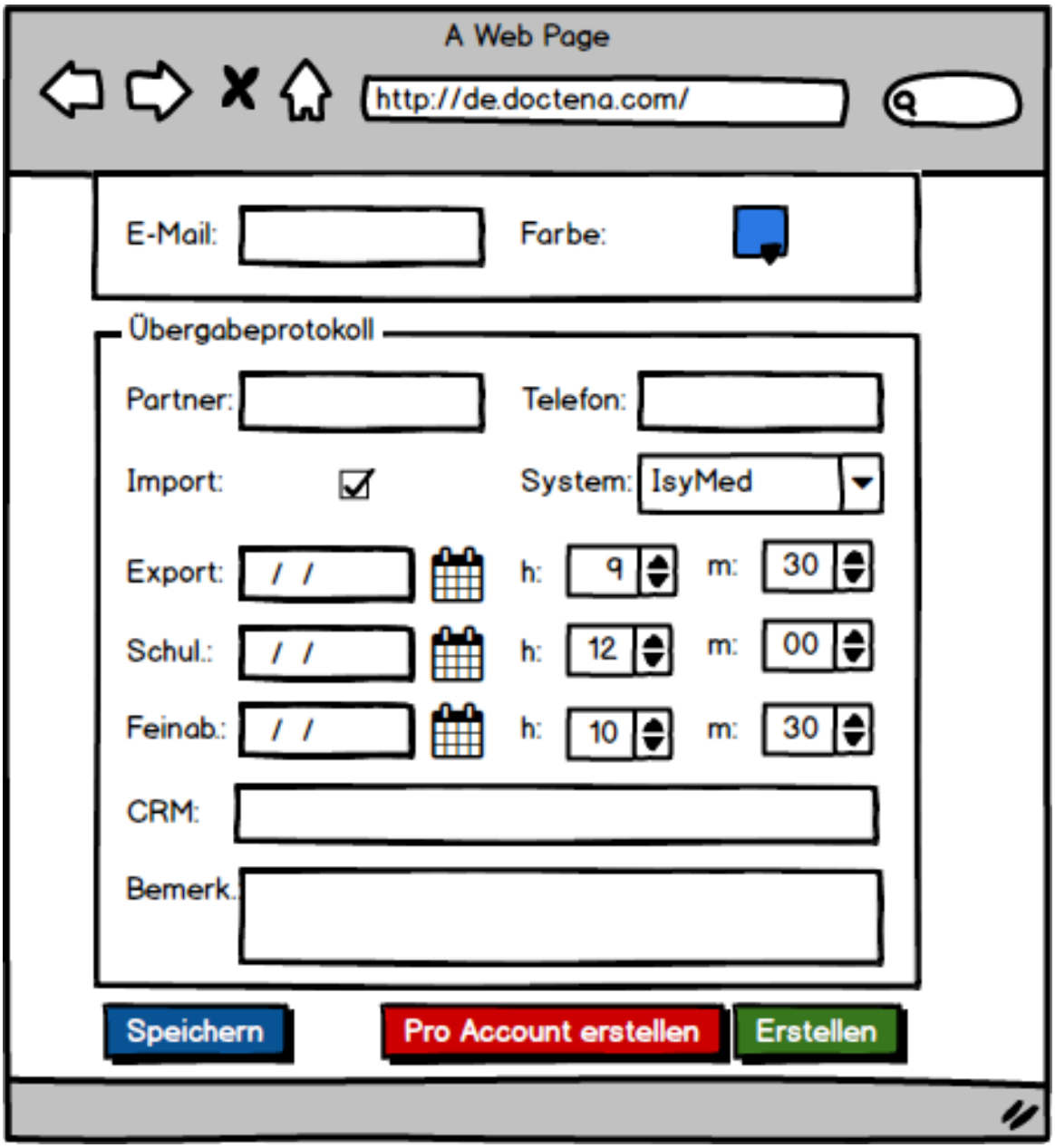

Abbildung 2: Neue Felder und Übergabeprotokoll

A.3.1 HAML-Partial

A.3.2 Contracts-Controller

```
class ContractsController < ApplicationController
  include ContractsHelper

  layout 'contract'

  before_action :authenticate_admin_user!, except: [:show, :update, :download_pdf]

  [...]

  # Vertragsabschluss
  def signup_now
    @contract.calculate_sums!
    @contract.contract_is_valid = @contract.valid?

    success = false
    if @contract.save
      success = if @contract.created_account_id.present?
                  convert_account_demo_to_full
                else
                  @contract.create_user_account
                end
    end

    if success
      # Account creation successful, mark contract:
      @contract.contract_state = Contract::SIGNED
      @contract.signup_date = Time.zone.now
      @contract.save
      UserMailer.salesform_signup_done(@contract.id.to_s).deliver_later

      issue = 'Vertragsabschluss durch Callagent'
      SupportMailer.salesform_support_issue(@contract.id.to_s, issue).deliver_later

      call_to_action_onboarding

      if @contract.reload.created_account_id.present?
        sign_in :user, Account.find(@contract.created_account_id).try(:owner) unless current_admin_user.present?
        redirect_to account_welcome_path(@contract.created_account_id)
      else
        render 'signup_done', layout: 'page'
      end
    else
      render 'edit'
    end

  [...]

  def call_to_action_onboarding
    issue = 'Vertrag bereit fr Doctena Pro Onboarding durch Onboarding Manager'
```

```
50      SupportMailer.doctena_pro_onboarding(@contract.id.to_s, issue).deliver_later
51    end
52
53     [...]
54
55    def send_user_to_bus(method, user)
56      routing_key = "doxter.secretary.#{user.eid}.#{method}.de"
57      puts routing_key
58      BusManager::Services::TranslationAssistant.to_bus(
59        user,
60        routing_key
61      ).translate
62    end
63
64    def send_practice_to_bus(method, practice)
65      routing_key = "doxter.practice.#{practice.eid}.#{method}.de"
66      puts routing_key
67      BusManager::Services::TranslationAssistant.to_bus(
68        practice,
69        routing_key
70      ).translate
71    end
72
73    def send_doctor_to_bus(method, doctor)
74      routing_key = "doxter.doctor.#{doctor.eid}.#{method}.de"
75      puts routing_key
76      BusManager::Services::TranslationAssistant.to_bus(
77        doctor,
78        routing_key
79      ).translate
80    end
81
82    def send_calendar_to_bus(method, calendar)
83      routing_key = "doxter.agenda.#{calendar.eid}.#{method}.de"
84      puts routing_key
85      BusManager::Services::TranslationAssistant.to_bus(
86        calendar,
87        routing_key
88      ).translate
89    end
90
91     [...]
92
93    def create_address_for_bus(contract, city)
94      Address.new(
95        city: contract.billing_city,
96        city_eid: city.eid,
97        latlng: city.latlng,
98        line1: contract.billing_street,
99        zip: contract.billing_zip
100     )
```

```
101   end
102
103   def update_address_for_bus(contract, city)
104     # TODO: add functionality for more than one practice per contract
105     address = contract.practices.first.address
106     address.city = contract.billing_city
107     address.city_eid = city.eid
108     address.latlng = city.latlng
109     address.line1 = contract.billing_street
110     address.zip = contract.billing_zip
111     address
112   end
113
114   def create_practice_for_bus(contract, address)
115     Practice.new(
116       address: address,
117       created_at: Time.zone.now,
118       eid: SecureRandom.uuid,
119       fax: '',
120       name: contract.practice_name,
121       philosophy: '',
122       phone: contract.practice_phone,
123       primary_email: contract.email,
124       secondary_email: '',
125       updated_at: Time.zone.now,
126       website: ''
127     )
128   end
129
130   def update_practice_for_bus(contract, address)
131     # TODO: add functionality for more than one practice per contract
132     practice = contract.practices.first
133     practice.address = address
134     practice.fax = ''
135     practice.name = contract.practice_name
136     practice.philosophy = ''
137     practice.phone = contract.practice_phone
138     practice.primary_email = contract.email
139     practice.secondary_email = ''
140     practice.updated_at = Time.zone.now
141     practice.website = ''
142     practice
143   end
144
145   def create_user_for_bus(practice, practitioner, password)
146     User.new(
147       confirmed_at: Time.zone.now, # confirmed_at equals status in pro
148       created_at: Time.zone.now,
149       email: practitioner.email,
150       encrypted_password: password,
151       first_name: practitioner.first_name,
```

```
152         gender: practitioner.gender,
153         last_name: practitioner.last_name,
154         practice_eid: practice.eid,
155         title: practitioner.title,
156         updated_at: Time.zone.now
157       )
158     end
159
160     def update_user_for_bus(practice, practitioner, user)
161       user.confirmed_at = Time.zone.now # confirmed_at equals status in pro
162       user.email = practitioner.email
163       user.first_name = practitioner.first_name
164       user.gender = practitioner.gender
165       user.last_name = practitioner.last_name
166       user.practice_eid = practice.eid
167       user.title = practitioner.title
168       user.updated_at = Time.zone.now
169       user
170     end
171
172     [...]
173
174     def signup_onboarding
175       # create the same initial password for all users
176       password = ENV["INITIAL_ONBOARDING_PASSWORD"]
177       new_hashed_password = User.new(:password => password).encrypted_password
178
179       admin_user_mail = "team+" + @contract.email
180
181       city = ::City.find_by(zip_codes: @contract.billing_zip)
182
183       # practice
184       if !@contract.practices.present?
185         method = 'created'
186         address = create_address_for_bus(@contract, city)
187         practice = create_practice_for_bus(@contract, address)
188         @contract.practices << practice
189       else
190         method = 'updated'
191         address = update_address_for_bus(@contract, city)
192         practice = update_practice_for_bus(@contract, address)
193       end
194       send_practice_to_bus(method, practice)
195
196       # user
197       if !@contract.users.present?
198         method = 'created'
199
200         @contract.practitioners.each do |practitioner|
201           user = create_user_for_bus(practice, practitioner, new_hashed_password)
202           @contract.users << user
```

```
203           send_user_to_bus(method, user)
204         end
205
206         if @contract.practitioners.count > 1
207           admin_user = create_admin_user_for_bus(@contract, practice, admin_user_mail, new_hashed_password)
208           @contract.users << admin_user
209           send_user_to_bus(method, admin_user)
210         end
211       else
212         @contract.practitioners.each_with_index do |practitioner, i|
213           user = @contract.users[i]
214
215           if !user.nil?    # as long as there are the same amount of practitioners as before
216             method = 'updated'
217             user = update_user_for_bus(practice, practitioner, user)
218             @contract.users[i] = user
219           else      # when there are more users than before
220             method = 'created'
221             user = create_user_for_bus(practice, practitioner, new_hashed_password)
222             @contract.users << user
223           end
224
225           send_user_to_bus(method, user)
226         end
227
228         if @contract.practitioners.count > 1
229           admin_index = @contract.users.each_with_index do |user, i|
230             return i if user.email.starts_with? 'team+'
231             nil
232           end
233
234           if !admin_index.nil?
235             method = 'updated'
236             admin_user = update_admin_user_for_bus(@contract, admin_index, admin_user_mail, practice)
237             @contract.users[admin_index] = admin_user
238           else
239             method = 'created'
240             admin_user = create_admin_user_for_bus(@contract, practice, admin_user_mail, new_hashed_password
                  )
241             @contract.users << admin_user
242           end
243
244           send_user_to_bus(method, admin_user)
245         end
246       end
247
248       # account
249       account = get_account_for_bus(@contract)
250
251       # doctor and calendar
252       if !@contract.doctors.present? && !@contract.calendars.present?
```

```
253         method = 'created'
254         @contract.practitioners.each do |practitioner|
255           doctor = create_doctor_for_bus(account, practice, practitioner)
256           @contract.doctors << doctor
257
258           calendar = create_calendar_for_bus(account, practice, practitioner, doctor)
259           @contract.calendars << calendar
260
261           send_doctor_to_bus(method, doctor)
262           send_calendar_to_bus(method, calendar)
263         end
264       else
265         @contract.practitioners.each_with_index do |practitioner, i|
266           method = 'updated'
267
268           doctor = @contract.doctors[i]
269           calendar = @contract.calendars[i]
270
271           # as long as there are the same amount of practitioners as before
272           if !doctor.nil? && !calendar.nil?
273             @contract.doctors[i] = update_doctor_for_bus(doctor, account, practice, practitioner)
274             @contract.calendars[i] = update_calendar_for_bus(calendar, account, doctor, practice, practitioner)
275           else  # if another practitioner was added
276             method = 'created'
277
278             doctor = create_doctor_for_bus(account, practice, practitioner)
279             @contract.doctors << doctor
280
281             calendar = create_calendar_for_bus(account, practice, practitioner, doctor)
282             @contract.calendars << calendar
283           end
284
285           send_doctor_to_bus(method, doctor)
286           send_calendar_to_bus(method, calendar)
287         end
288       end
289
290       # TODO: set id from Doctena Pro after account creation is implemented
291       @contract.created_pro_account_id = "some-fake-id"
292
293       @contract.save(validate: false)
294
295       flash.now[:notice] = "Die Daten aus Vertrag-Nr. #{@contract.contract_nr} wurden zur Doctena Pro
              Accounterstellung an den Bus gesendet."
296       render 'offer_info', layout: 'page'
297     end
298
299     [...]
300
301 end
```

A.4 Testausgabe

```
nd@nd-780:~/Work/weise$ be rake test TEST=test/models/contract_test.rb
Expected string default value for '--serializer'; got true (boolean)
Started with run options --seed 25375

  4/4: [==========================================================================================] 100%

Finished in 1.60993s
4 tests, 4 assertions, 0 failures, 0 errors, 0 skips
nd@nd-780:~/Work/weise$
nd@nd-780:~/Work/weise$
nd@nd-780:~/Work/weise$
nd@nd-780:~/Work/weise$
nd@nd-780:~/Work/weise$ be rake test TEST=test/controllers/contracts_controller_test.rb
Expected string default value for '--serializer'; got true (boolean)
Started with run options --seed 29016

  9/9: [==========================================================================================] 100%

Finished in 3.82818s
9 tests, 117 assertions, 0 failures, 0 errors, 0 skips
nd@nd-780:~/Work/weise$
nd@nd-780:~/Work/weise$
nd@nd-780:~/Work/weise$
nd@nd-780:~/Work/weise$
nd@nd-780:~/Work/weise$ be rake test TEST=test/mailers/onboarding_mailer_test.rb
Expected string default value for '--serializer'; got true (boolean)
Started with run options --seed 37909

  2/2: [==========================================================================================] 100%

Finished in 0.33726s
2 tests, 9 assertions, 0 failures, 0 errors, 0 skips
nd@nd-780:~/Work/weise$
nd@nd-780:~/Work/weise$
nd@nd-780:~/Work/weise$
nd@nd-780:~/Work/weise$
nd@nd-780:~/Work/weise$ be rake test TEST=test/integration/contracts_integration_test.rb
Expected string default value for '--serializer'; got true (boolean)
Started with run options --seed 9943

You're running an old version of PhantomJS, update to >= 2.1.1 for a better experience.=---=---=---=---=-] 0%
  7/7: [==========================================================================================] 100%

Finished in 6.91049s
7 tests, 7 assertions, 0 failures, 0 errors, 0 skips
```

Abbildung 3: Testausgabe der Unit- und Integration-Tests

A.5 Benutzerdokumentation

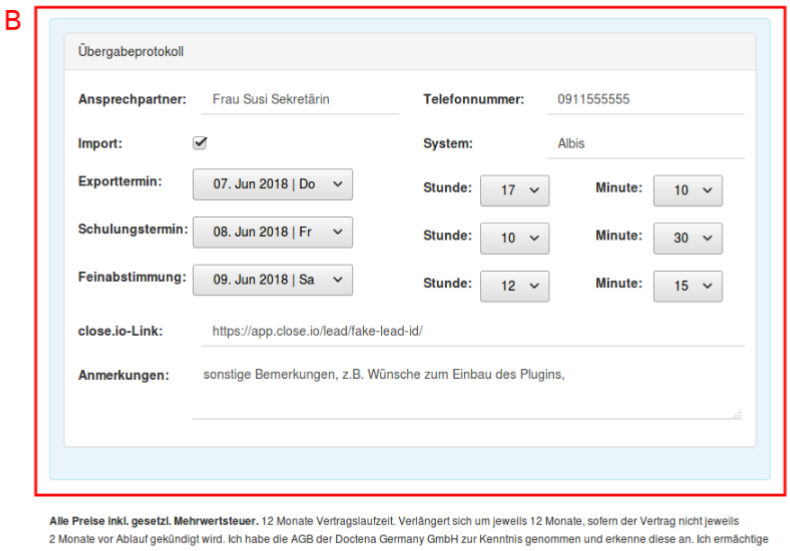

Abbildung 4: Auszug aus der Benutzerdokumentation