Fachinformatiker für Anwendungsentwicklung

Dokumentation zur betrieblichen Projektarbeit

Automatisierte Accounterstellung via AMQP-Messaging-System

mit Konsolidierung der Datenquellen (Übergabeprotokoll und Angebotssystem)

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

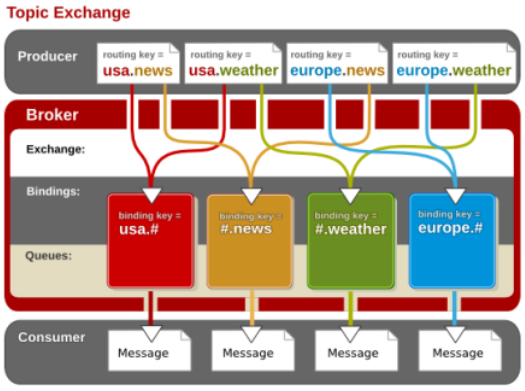


Abbildung 1: Interprozesskommunikation mit Messaging-Queues

Prüfungsausschuss: Tatjana Goebel, Ali Hafezi, Ralf Merettig
Abgabetermin: Berlin, den 24.05.2018

Doctena Germany GmbH
Urbanstr. 116, 10967 Berlin

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

AMQP	Advanced Message Queuing Protocol
AWS	Amazon Web Services
CI	Continuous Integration
CIO	Chief Information Officer
CPP	Central Patient Portal
CRM	Customer Relationship Management
CRUD	Create Read Update Delete
CTO	Chief Technology Officer
JSON	JavaScrip Open Notation
MVC	Model View Controller
HAML	HTML abstraction markup language
HTML	Hypertext Markup Language
IHK	Industrie- und Handelskammer
TDD	Test-Driven Development

1 Einleitung

Pläne sind nichts, Planung ist alles.
Dwight D. Eisenhower, ehemaliger US-Präsident

Kein Plan überlebt die erste Feindberührung.
Helmuth von Moltke, preußischer Generalfeldmarschall

1.1 Projektumfeld

Das Projekt wird als Teil der Abschlussprüfung im Rahmen meiner Ausbildung zum Fachinformatiker für Anwendungsentwicklung bei der Doctena Germany GmbH umgesetzt. Doctena ist ein internationales Unternehmen mit Hauptsitz in Luxemburg und Niederlassungen in 5 weiteren europäischen Ländern. Doctena bietet Patienten eine internationale Plattform zur Online-Terminbuchung an. "2013 (...) wurde die Plattform Doctena in Luxemburg ins Leben gerufen. (...) Aufgrund des anhaltenden Erfolges und des schnellen Wachstums des Projekts dehnte das Unternehmen seine Aktivitäten auf Belgien, die Niederlande, [Österreich,] die Schweiz und Deutschland aus. Seit 2016 hat sich Doctena mit sechs Wettbewerbern zusammengeschlossen: DocBook (BE), Doxter (DE), Terminland (DE), Sanmax (BE), Mednanny (AT) und Bookmydoc (LU). Doctena ist heute die führende medizinische Buchungsplattform in Europa."[1] Doctena beschäftigt momentan um die 80 Mitarbeiter, ca. 30 davon hier in Berlin. Hauptprodukt neben der Terminbuchungs-Plattform für Patienten ist die cloudbasierte Terminverwaltungs-Lösung Doctena Pro für Ärzte und Praxen. "Doctena hat das Ziel, den Zugriff von Patienten auf verfügbare Termine von Ärzten und Praktikern zu vereinfachen. Patienten können mit Hilfe der Onlineplattform oder der Handy-App verfügbare Termine sehen und buchen. (...) Die Lösung ist mit vielen medizinischen Buchungssoftwares kompatibel und kann deshalb leicht in die Struktur von Ärzten und ihren Praxen integriert werden."[2] So können Ärzte praxisintern ihre Verfügbarkeiten managen und gleichzeitig freie Termine über die Plattform oder über die eigene Internetseite anbieten. Eigentlicher Kunde des Projektes sind die Abteilungen Verkauf und Onboarding der Doctena Germany GmbH. Die Abnahme des Projekts erfolgt durch den Chief Technology Officer (CTO) der Doctena Germany GmbH, André Rauschenbach.

1.2 Projektziel

Ziel des Projektes ist die Erweiterung des zur Angebots- und Vertragserstellung benutzten Angebotssystemes. Den Onboarding-Managern soll damit ermöglicht werden, nach Annahme des Angebotes durch den Kunden, in unserem System die automatische Accounterstellung auf dem luxemburger System Doctena Pro über den bestehenden Advanced Message Queuing Protocol (AMQP) Message-Bus auszulösen. Zusätzlich soll die Maske des Angebotsformulars um Eingabefelder für das in Folge zu

[1] Über Doctena - Unsere Firma - Eine Erfolgsgeschichte, **?**
[2] Über Doctena - Der Doctena Update Catalog, **?**

erstellende Übergabeprotokoll erweitert und so eine Konsolidierung der bisher für den Onboarding-Prozess verwendeten Datenquellen erreicht werden.

1.3 Projektbegründung

Mit diesem Projekt soll die bisherige manuelle Erstellung der neuen Kunden-Accounts durch die Onboarding-Manager automatisiert werden, was zu einer Zeit- und somit auch Kostenreduzierung im Onboarding-Prozess führt. Gleichzeitig sollen mögliche Fehlerquellen bei der im bisherigen Prozess hierzu verwendeten doppelten Datenhaltung eliminiert werden. Hauptmotivation hinter dem Projekt ist somit die Prozessoptimierung bei der bisherigen Accounterstellung. Neben dem reinen Arbeitsaufwand sollen hiermit auch Fehler reduziert werden, die durchdie vielen repetitive Aufgaben beim Onboarding und anspruchslose Copy-und-Paste-Tätigkeiten leicht entstehen können.

1.4 Projektschnittstellen

Das Angebotssystem im Backend des deutschen Systems Doctena Standard wurde mit Ruby on Rails erstellt. Es besteht eine Verbindung zu einer nicht-relationalen MongoDB Datenbank über einen in Rails eingebundenen Data-Connector, den MongoDB Driver. Die browserbasierte Eingabemaske des Views kann so unabhängig vom verwendeten Betriebsystem benutzt werden. Die externe Kommunikation zwischen den Objekten in Rails bei Doctena Standard und den Objekten im Zielsystem Doctena Pro, mit denen der neue Benutzeraccount nach Vertragserstellung angelegt werden soll, erfolgt über einen AMQP Message-Bus mit RabbitMQ. Dieser wird bereits zur Synchronisation der Verfügbarkeiten der Ärzte aus den verschiedenen Backend-Bereichen der angeschlossenen Systeme mit dem Central Patient Portal (CPP) von Doctena benutzt. Die verwendete Datenstruktur zum Versand der Objekte ist ßacJSON. Das Projekt wurde von Doctenas internationalem Chief Information Officer (CIO), Alain Fountain, in Absprache mit dem CTO von Doctena Germany, André Rauschenbach, genehmigt. Doctena stellt somit als Kunde im Rahmen der Projektarbeit und Ausbildung alle zur Umsetzung benötigten Mittel zur Verfügung. Benutzer der Anwendung sind die Mitarbeiter von Doctena Germany in den Abteilungen Verkauf und Onboarding. Diesen soll nach Abnahme durch den Auftraggeber, vertreten durch den CTO von Doctena Germany, das fertige Produkt präsentiert werden. Zusätzlich soll für die Benutzer eine Benutzerdokumentation für die Accounterstellung im Firmeninternen Wiki erstellt werden.

1.5 Projektabgrenzung

Das Aktivieren von Features über den Bus ist Seitens Luxemburg noch nicht möglich. Deswegen wurde diese Funktionalität aus dem Projekt ausgeklammert. Sie wird erst zu einem späteren Zeitpunkt umgesetzt. Da das Einrichten der Testumgebungen für zwei komplette Systeme sehr aufwendig ist und den Projektrahmen übersteigt, wurde auf die Korrektheit der Messages auf dem Bus getestet.

2 Projektplanung

2.1 Projektphasen

Das Projekt soll im Zeitraum zwischen der schriftlichen Abschlussprüfung und dem Abgabetermin des Projekts während den täglichen Arbeitszeiten von 10:00 bis 19:00 Uhr umgesetzt werden. Die Arbeitszeit kann während diesem Zeitraum neben der von der Erledigung dringender oder anderweitig notwendiger Aufgaben beanspruchten Zeit frei für das Projekt genutzt werden. Entsprechend dem in Kapitel 2.5 beschriebenen Entwicklungsprozess wurde der Projektablauf in die entsprechenden Phasen unterteilt. Deren nähere Planung bzw. Durchführung kann den entsprechenden Kapiteln dieser Dokumentation entnommen werden.

2.2 Zeitplanung

Für die Umsetzung des Projektes stehen Seitens der Anforderungen der Industrie- und Handelskammer (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.

Projektphase	Geplante Zeit
Analyse	6 h
Entwurf	11 h
Implementierung	39 h
Abnahme und Deployment	5 h
Dokumentation	9 h
Gesamt	**70 h**

Tabelle 1: Zeitplanung

2.3 Ressourcenplanung

Die benötigten Mittel zur Durchführung des Projektes werden vom Auftraggeber Doctena zur Verfügung gestellt. Eine detaillierte Auflistung der zur Durchführung benötigten Ressourcen findet sich im Anhang A.1: Projekt-Ressourcen auf Seite i. Die Benutzung dieser Ressourcen wird mit pauschalen Werten für die in Kapitel 3 angestellten Berechnungen berücksichtigt.

2.4 Entwicklungsprozess

Der Projektablauf wurde vom Wasserfallmodell ausgehend in die folgenden Projektphasen unterteilt:

Definition und Projektantrag, Planung, Analyse, Entwurf, Implementierung, Qualitätssicherung und Abnahme, Einführung, Dokumentation

Diese werden in sequentieller Reihenfolge mit zwischenzeitlichen Projektbesprechungen mit den beteiligten Stellen zum aktuellen Projektstand bis zum Abgabetermin abgearbeitet. Die Entwicklung während der Implementierungsphase wird nach Test-Driven Development (TDD)-Prinzipien durchgeführt, wodurch auch die Tests zur Sicherstellung der Einhaltung der vereinbarten Anforderungen aus dem Pflichtenheft teilweise während dieser Phase erstellt werden. Da das Angebotssystem an unsere Continuous Integration (CI)-Pipeline angebunden ist, kann eine erfolgreiche Abnahme und anschließende Einführung erst nach einem bestehen aller Tests der Qualitätssicherung erfolgen. Artefakte für die Dokumentation werden, wo möglich, bereits während der gesamten Durchführung gesammelt.

3 Analysephase

Eine erste Analyse wurde bereits während der Projektdefinition zur Antragstellung durchgeführt. Darauf aufbauend folgt in diesem Kapitel eine genauere Analyse der Situation. Die hierbei gewonnenen Erkenntnisse werden dann als Anforderungen im Lastenheft genauer definiert.

3.1 Ist-Analyse

Unsere Verkäufer erstellen täglich Angebote an Ärzte aus ganz Deutschland. Die Daten, die zur Erstellung dieser Angebote im Formular des Angebotssystems eingegeben werden, können bei Doctena Standard bereits zum automatischen Erstellen eines Accounts für den zukünftigen Benutzer verwendet werden. Um einen neuen Account bei Doctena Pro anzulegen werden die gleichen Daten als Grundlage benutzt. Die im Angebotssystem bereits in digitaler Form vorliegenden Daten zu Ärzten und Praxis werden im momentanen Onboarding-Prozess bei Doctena Pro von einem Onboarding-Manager per Copy-und-Paste aus dem Angebot in ein Übergabeprotokoll im von Doctena verwendeten Customer Relationship Management (CRM), Close.io, kopiert. Hier kommen einige zusätzliche Informationen, wie Termine für Schulung, Feinabstimmung und Datenimport hinzu. Dann werden in einem weiteren manuellen Schritt die Accounts im Backend von Doctena Pro vom Onboarding-Manager zusammengeklickt und die Daten zu Ärzten und Praxis wieder aus dem Übergabeprotokoll per Copy-und-Paste in die Eingabemaske des Backends transferiert. Dieser Prozess ist repetitiv, nimmt unnötig Zeit in Anspruch, erzeugt in seinem Verlauf redundante Daten und ist somit fehleranfällig. Gleichzeitig existiert zum Datenaustausch der verschiedenen internationalen Systeme ein AMQP-Message-Queue-Bus, über den die Verfügbarkeiten der Ärzte mit dem CPP synchronisiert werden. Um den Onboarding-Prozess zu optimieren, sollen zum einen die Daten des Übergabeprotokolls direkt im Angebotsformular eingegeben werden können, zum anderen sollen die Daten zu Praxis und den Ärzten dazu verwendet werden, die Objekterstellung in Doctena Pro über den Bus auszulösen. Hierzu müssen zusätzliche Felder im Formular der Benutzungsoberfläche und im verwendeten Datenmodell hinzugefügt werden.

In der Geschäftslogik im Controller müssen die Daten dann zum auf dem Message-Queue-Bus verwendeten AMQP-Exchange-Type konvertiert werden. Es soll ein Button zum Auslösen des Versandes über den Bus erstellt werden, welcher nur für Onboarding-Manager sichtbar ist. Bei einem Vertragsabschluss für Doctena Pro soll automatisch eine E-Mail an den verantwortlichen Onboarding-Manager versendet werden, damit dieser die Daten des Angebots im Formular überprüfen und ggf. korrigieren kann und dann die automatische Account-Erstellung auslösen kann.

3.2 Wirtschaftlichkeitsanalyse

3.2.1 „Make or Buy"-Entscheidung

Die „Make or Buy"-Entscheidung ist in diesem Fall leicht getroffen. Zum einen existiert bereits eine bestehende AMQP-Busverbindung zum Datentransfer zwischen dem Ruby on Rails Backend von Doctena Standard, in welchem das Angebotssystem eingebettet ist, und dem Java Zielsystem Doctena Pro in Luxemburg. Zum anderen kann über kommerziell erhältliche Lösungen zur Angebotserstellung die automatische Account-Erstellung nicht realisiert, und so auch der bisherige manuelle Onboarding-Prozess nicht optimiert werden.

3.2.2 Projektkosten

Die realen Kosten für die Durchführung des Projekts setzen sich sowohl aus Personal-, als auch aus Ressourcenkosten zusammen. Gerechnet wird hier bei den Personalkosten lediglich mit dem fiktiven Gehalt eines Auszubildendem im dritten Lehrjahr von ungefähr 884 € Brutto bei 20 Arbeitstagen im Monat. Der Arbeitgeberanteil zur Sozialversicherung bemisst sich auf monatlich 229,53 €. Für das Gehalt aller übrigen am Projekt beteiligten Mitarbeiter wird ein pauschaler Stundensatz von 30 € angenommen.

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

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

Es ergibt sich ein Stundenlohn von 6,96 €. Die Durchführungszeit des Projekts beträgt 70 Stunden. Die Nutzung von Ressourcen[3] 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 €.

[3] Räumlichkeiten, Arbeitsplatzrechner, Lizenzen, etc.

Vorgang	Zeit	Kosten pro Stunde	Kosten
Entwicklungskosten	70 h	6,96 €	388,08 €
Fachgespräche	2 h	30 €	60 €
Meeting mit 2 Abteilungsvertretern	2 h	30 € · 2	120 €
Benutzerschulung	1 h	30 €	30 €
Ressourcen	70 h	10 €	700 €
Kosten			**1397,20 €**

Tabelle 2: Projektkosten

3.2.3 Amortisationsdauer

Mit dem pauschale Stundensatz von 30 € und den kalkulierten Kosten aus der vorherigen Rechnung ergibt sich für 3 Mitarbeiter im Onboarding, die pro Person bei jeweils nur einem angelegten Account pro Tag 30 Minuten einsparen, bei 20 Arbeitstagen pro Monat eine Amortisationsddauer von ca. 1,53 Monaten. Selbst wenn sich die Kosten und Gewinne durch Nachbesserungen oder anderweitigen Arbeitsaufwand im Onboarding verändern sollten, amortisiert sich das Projekt doch relativ schnell.

3.3 Nutzwertanalyse

Den größten Nutzen des Projektes stellt die Optimierung des bisherigen Arbeitsprozesses beim Onboarding neuer Kunden dar. Dadurch wird Zeit eingespart und so freie Kapazitäten für andere Tätigkeitsfelder im Onboarding-Prozess geschaffen. Durch die Reduzierung der sich wiederholenden manuellen Tätigkeiten und der dadurch möglichen Fehler erhöht sich die Kundenzufriedenheit, wodurch zusätzlich die Anzahl der Kundenanfragen an die Support-Abteilung reduziert wird.

3.4 Qualitätsanforderungen

Da das Angebotssystem als Webanwendung sowohl von den Mitarbeitern von Doctena Germany in den Abteilungen Verkauf und Onboarding, wie auch von den Kunden zum Abschließen von Verträgen online genutzt wird, wird bei den Qualitätsanforderungen besonderes Augenmerk auf die Abgrenzung der Funktionalität bei den verschiedenen Benutzerrollen gelegt. Die Funktionen zur Account-Erstellung sollen nur Onboarding-Managern zur Verfügung stehen, die Eingabefelder für system-interne Daten nur den Angestellten von Doctena, während die Kunden in der Lage sein sollen, ihre personenbezogenen Daten ggf. ändern zu können.

3.5 Lastenheft

Im folgenden werden die wichtigsten Anforderungen an das Projekt aus dem Lastenheft dargestellt.

Anforderungen an die Benutzungsoberfläche:

LB10: Die Daten des Übergabeprotokolls sollen im Angebotsformular integriert werden.
LB20: Es soll ein Button zur Account-Erstellung hinzugefügt werden.
LB30: benötigte zusätzliche Felder zur Account-Erstellung benötigten Daten hinzugefügt werden.

Funktionelle Anforderungen:

LF10: Der Button zur Account-Erstellung soll nur von Onboarding-Managern genutzt werden können. Die Formularfelder zur Account-Erstellung und des Übergabeprotokolls sollen nur für Verkauf und Onboarding zur Verfügung stehen.
LF20: Bei einem Vertragsabschluss durch einen Kunden soll die Onboarding-Abteilung eine Benachrichtigung per E-Mail bekommen.
LF30: Es sollen die für einen Account auf Doctena Pro benötigten Objekte erzeugt werden.
LF40: Die Objekte sollen über den AMQP Message-Queue-Bus an Doctena Pro übermittelt werden.
LF50: Wenn mehr als ein Arzt in der Praxis vorhanden ist, soll ein zusätzlicher Admin-User für die Praxis in Doctena Pro angelegt werden.

Sonstige Anforderungen:

LS10: Die Funktionalität soll durch Tests der Anforderungen gewährleistet werden, damit das Projekt im Produktivsystem genutzt werden kann.
LS20: Die Benutzer sollen im Umgang mit den neuen Funktionen geschult werden.

3.6 Zwischenstand

Tabelle 3 zeigt den Zwischenstand nach der Analysephase.

Vorgang	Geplant	Tatsächlich	Differenz
Durchführung einer Ist-Analyse	1 h	1 h	
Wirtschaftlichkeitsanalyse und Amotisationsrechnung	1 h	1 h	
Ermittlung von Use-Cases	1 h	0.5 h	-0.5 h
Qualitätsanforderungen und Lastenheft	3 h	4 h	+1 h
Analysephase	**6 h**	**6.5 h**	**+0.5 h**

Tabelle 3: Zwischenstand nach der Analysephase

4 Entwurfsphase

Da das Projekt die Erweiterung des Funktionsumfanges eines bereits bestehenden Systems darstellt, sind viele Entscheidungen zu den verwendeten Technologien bereits im Vorfeld getroffen. Im Folgenden wird deshalb detaillierter beschrieben, wie die geplanten Erweiterungen im existierenden Angebotssystem realisiert werden sollen.

4.1 Zielplattform

Die Angebotsformular ist als Webanwendung mit den drei gängigsten Webbrowsern unabhängig vom verwendeten Betriebssystem ausführbar. Diese wurde mit Ruby und Rails geschrieben, bindet über Bibliotheken JQuery und Bootstrap ein und wird über Heroku und Amazon Web Services (AWS) zur Verfügung gestellt. Als Datenbanksystem für das Projekt ist die nicht-relationale Datenbank MongoDB von MongoLab angebunden. Zum Datenversand aus dem Angebotsystem zu Doctena Pro wird RabbitMQ verwendet.

4.2 Architekturdesign

Das Angebotsystem besteht aus einer im Framework Rails üblichen Model View Controller (MVC)-Struktur. Somit erfolgt die interne Kommunikation zwischen der in der Datenhaltungsschicht angebundenen nicht-relationalen Datenbank MongoDB und den Modell-Klassen über einen in Rails als Bibliothek eingebundenen Data-Connector, den MongoDB Driver. Die browserbasierte Eingabemaske des Views in der Schicht der Benutzungsoberfläche interagiert intern über die im Controller vorhandene Geschäftslogik mit den üblichen Create Read Update Delete (CRUD)-Operationen und Formular-Parametern über die in den Routes eingestellten Pfade mit dem Modell und so der Datenschicht.

4.3 Entwurf der Benutzungsoberfläche

Als Webanwendung, die auch von Kunden zum Vertragsabschluss genutzt wird, ist das Angebotsformular bereits durch die Verwendung von Bootstrap selbst für die mobile Benutzung optimiert. Aus repräsentativen Gründen muss das Formulardesign klar und übersichtlich sein und allen Anforderungen an die Corporate Identity gerecht werden. Da täglich von technisch nicht geschultem Personal damit gearbeitet wird, muss jedes neue Feld zu Accounterstellung und Übergabeprotokoll möglichst eindeutig beschriftet sein. Diese neuen Felder können in Rails mittels der vereinfachten Auszeichnungssprache HAML geschrieben werden, welche daraus den gewünschten HTML-Code für die Seite erzeugt. Ein Screenshot des Mockups für die neuen Felder in der Eingabemaske findet sich im Anhang A.3: Mockup auf Seite iii.

4.4 Datenmodell

Das Datenmodell lässt sich dank einem einfachen Mapping der Datenstruktur bei Rails und MongoDB über die entsprechende Modell-Klasse verwirklichen. Durch die Benutzung eines Connectors (MongoDB Driver) können die neuen Datenfelder einfach auf dem Modell Vertrag hinzugefügt werden. Zum persistieren der Objekte im Vertrag werden diese beim Speichern des Vertrages über diesen verknüpft. Beim Versand über den Bus werden in Doctena Standard die folgenden Objekte benötigt:

Account, User, Practice, Doctor, Calendar

4.5 Geschäftslogik

Bei der Umsetzung der Geschäftslogik muss im Controller eine neue Methode erstellt werden, über welche die Accounterstellung geroutet wird. Hier ist beim Erstellen der Objekte darauf zu achten, ob sich nach dem Speichern mehr oder weniger Behandler im Vertragsformular finden. Je nach Fall ist dann ggf. ein Admin-User zu erzeugen und zu speichern oder muss wieder entfernt werden. Die Objekte müssen für den Versand von Doctena Standard zum Zielsystem in das auf dem AMQP-Bus benutzte Format JavaScrip Open Notation (JSON) übersetzt werden. Auf Doctena Pro werden aus den Messages der Queues die neuen Objekte erzeugt und so werden die Objekte von Doctena Standard auf die von Doctena Pro gemappt. Die entsprechenden Objekte im Zielsystem sind:

Secretary, Practice, Agenda, Doctor

4.6 Maßnahmen zur Qualitätssicherung

Damit das Projekt überhaupt im Produktivsystem genutzt werden kann, müssen für die wichtigsten Anforderungen entsprechende Tests mit dem in Rails eingebundenen Test-Framework Minitest geschrieben werden. Diese sorgen bei Änderungen am Quellcode dafür, dass man im Fall eines unbeabsichtigten Fehlers nicht dringend benötigte Funktionalitäten unbrauchbar macht. Die Verwendete CI-Pipeline über das Repository auf Github und circle.ci sorgt bei einem grünen Master nach den Tests für ein Deployment auf dem Produktivsystem. Deshalb müssen die Prinzipien des TDD in unserem Entwicklungsprozess beachtet werden.

4.7 Pflichtenheft

Im folgenden die wichtigsten vereinbarten Pflichten des Projektes bezüglich Qualitätssicherung, um die Funktionalität und Erweiterbarkeit während der aktiven Nutzung im Produktivsystem gewährleisten zu können.

Pflichten der Benutzungsoberfläche:

PB10: Die Daten des Übergabeprotokolls müssen persistiert und ggf. wieder geladen und erneut persistiert werden.
PB20: Der Button zur Account-Erstellung muss den Datenversand über den AMQP Message-Queue-Bus auslösen.
PB30: Die Daten zur Account-Erstellung müssen persistiert und ggf. wieder geladen und erneut persistiert werden.

Funktionelle Pflichten:

PF10: Für Benutzer mit der Rolle Manager muss der Button zur Account-Erstellung sowie die Formularfelder zur Account-Erstellung und das Übergabeprotokoll sichtbar sein. Für Benutzer mit der

Rolle Verkäufer müssen die Formularfelder zur Account-Erstellung und das Übergabeprotokoll sichtbar sein. Für einen Benutzer ohne Rolle dürfen weder der Button zur Account-Erstellung noch die Formularfelder zur Account-Erstellung oder das Übergabeprotokoll vorhanden sein.
PF20: Bei einem Vertragsabschluss muss eine E-Mail an die Adresse onboarding@doctena.com gesendet werden.
PF40: Die Daten aus dem Vertrag müssen zur Account-Erstellung in das auf dem AMQP Message-Queue-Bus verwendete Datenformat übersetzt werden.
PF50: Bei mehr als einem Arzt pro Praxis muss ein zusätzlicher User als Admin-User ohne Arzt und Kalender erstellt und dieses zusätzliche Objekt über den Bus gesendet werden.

4.8 Zwischenstand

Tabelle 4 zeigt den Zwischenstand nach der Entwurfsphase.

Vorgang	Geplant	Tatsächlich	Differenz
Entwurf des überarbeiteten Eingabeformulars	2 h	1 h	-1 h
Klassendiagramm für die verteilten Systeme	2 h	3 h	+1 h
Zustandsdiagramm für die Vertragszustände	2 h	1 h	-1 h
Planung manueller Tests auf verteilten Systemen	2 h	4 h	+2 h
Testfälle für das Lastenheft definieren	3 h	2.5 h	-0.5 h
Entwurfsphase	**11 h**	**11.5 h**	**+0.5 h**

Tabelle 4: Zwischenstand nach der Entwurfsphase

5 Implementierungsphase

5.1 Implementierung der Datenstruktur

Das Hinzufügen der benötigten Datenfelder auf dem Model in Rails wurde über den Data-Connector zur nicht-relationalen Datenbank MongoDB umgesetzt und die erstellten Objekte wurden über den Vertrag persistiert.

5.2 Implementierung der Benutzeroberfläche

Die Anzeige der Felddaten im Übergabeprotokoll in der Benutzungsoberfläche wurde wie geplant über ein HAML-Partial realisiert. Auch die Sichtbarkeit für Benutzerrollen konnte mit einfachen Abfragen auf den entsprechend zu erzeugenden Elementen realisiert werden. Ein Codebeispiel des Partials zur Erzeugung der HTML-Seite findet sich im Anhang A.3.1: HAML-Partial auf Seite iv. Ein Screenshot

der fertigen Anwendung befinden sich unter anderem im Anhang A.5: Benutzerdokumentation auf Seite xiv.

5.3 Implementierung der Geschäftslogik

Ein Codebeispiel der Umsetztung der Geschäftslogik mit den objekterzeugenden Methoden aus dem Controller ist im Anhang A.3.2: Contracts-Controller auf Seite vii zu finden. Durch die Anwendung der Prinzipien des TDDs wurden Tests bereits während der Implementierungsphase geschrieben. Diese wurden, wenn möglich, bereits vor Umsetzung einer Anforderung erstellt, damit neben deren Funktionalität auch sichergestellt werden konnte, dass die Arbeit am zu bearbeitenden Feature auch beendet war, wenn die Tests von rot auf grün wechselten.

5.4 Zwischenstand

Tabelle 5 zeigt den Zwischenstand nach der Implementierungsphase.

Vorgang	Geplant	Tatsächlich	Differenz
Formularfelder des Übergabeprotokolls	4 h	2 h	-2 h
Button zur Accounterstellung, Sichtbarkeit für Benutzergruppen	3 h	1 h	-2 h
Unit-Tests für Formular und Benutzereingabe	4 h	3 h	-1 h
Persistieren der Daten	1 h	1 h	
Unit-Tests für die Datenhaltung	2 h	2 h	
Objekte für Datenversand auf Doctena Standard	10 h	18 h	+8 h
Unit-Tests für die Objekterstellung	5 h	5 h	
Versand über den -Messaging-Bus	4 h	7 h	+3 h
E-Mail-Versand bei Vertragsabschluss	3 h	1 h	-2 h
Unit-Tests für den Objektversand	3 h	4 h	+1 h
Implementierungsphase	**39 h**	**44 h**	**+5 h**

Tabelle 5: Zwischenstand nach der Implementierungsphase

6 Qualitätssicherung und Abnahme

6.1 Testing

Es wurden zu den in der Entwurfsphase spezifizierten Testanforderungen umfangreiche Unit-Tests mithilfe des Minitest-Frameworks für Rails geschrieben und das gewünschte Verhalten durch Integration-Tests mittels des Headless-Browsers PhantomJS sichergestellt. Zusätzlich wurden manuelle Tests bezüglich des Sendens und Lesens der Daten auf das Test-Amqp-System durchgeführt. Um in der verwendeten CI-Pipeline mit Github und circle.ci im Produktivsystem eingesetzt zu werden, müssen alle Tests grün sein. Ein Screenshot der Testausgabe findet sich im Anhang A.4: Testausgabe auf Seite xiii.

6.2 Abnahme

Die Abnahme erfolgte am 23.05.2018 durch den CTO von Doctena Germany, André Rauschenbach. Hierzu wurden die Tests über die CI-Pipeline auf circle.ci für den Github-Branch des Projektes ausgeführt um sicherzugehen, das sich das Projekt ohne Probleme mit dem Master auf Github zusammenführen und so im Produktivsystem nutzen lässt. Die vereinbarte Funktionalität wurde zusätzlich im Formular vor Ort demonstriert und getestet.

6.3 Zwischenstand

Tabelle 6 zeigt den Zwischenstand nach der Abnahmephase.

Vorgang	Geplant	Tatsächlich	Differenz
Tests der -Pipeline ausführen	1 h	0.5 h	-0.5 h
Durchführung manueller Tests	2 h	1 h	-1 h
Abnahme	**3 h**	**1.5 h**	**-1.5 h**

Tabelle 6: Zwischenstand nach der Abnahmephase

7 Einführungsphase

7.1 Geplante Einführung

Da benötigte Funktionalität auf Doctena Pro zum endgültigen Deployment noch nicht umgesetzt wurde, muss die finale Einführung auf einen späteren Zeitpunkt verschoben werden. Eine Benutzerschulung mit den bisher vorhandenen Funktionen ist für nächsten Monat angesetzt. Hierzu wurde bereits ein Eintrag im Firmenwiki (siehe Anhang A.5: Benutzerdokumentation auf Seite xiv) erstellt.

7.2 Zwischenstand

Tabelle 7 zeigt den Zwischenstand nach der Einführungsphase.

Vorgang	Geplant	Tatsächlich	Differenz
Deployment über die -Pipeline	1 h	0.5 h	-0.5 h
Schulung der Benutzer	1 h	0 h	-1 h
Einführung	**2 h**	**0.5 h**	**-1.5 h**

Tabelle 7: Zwischenstand nach der Einführungsphase

8 Dokumentation

Die Projektdokumentation wurde, soweit dies möglich war, bereits während der Planung und Implementierung mit angelegt. Sie wurde in LaTeX erstellt, um den Anforderungen an die betriebliche Projektarbeit im Rahmen der Abschlussprüfung zum Anwendungsentwickler Seitens der IHK gerecht zu werden. Für die Benutzerschulung wurde eigens eine Dokumentation der Funktionen im neuen Prozess im Firmenwiki erstellt, um auch bei zukünftigen Funktionsänderungen am Angebotssystem schnell erweiterbar zu sein.

8.1 Benutzerdokumentation

Die Benutzerdokumentation wurde zur Unterstützung der Mitarbeiter beim neuen Onboarding-Prozess im unternehmenseigenen Wiki erstellt und soll den Benutzern jederzeit einen einfachen Überblick über die Bedeutung der neuen Eingabefelder und den neuen Prozess im allgemeinen bieten. Sie wird zusammen mit der neuen Funktionalität während des Termines der geplanten Benutzerschulung vorgestellt. Ein Screenshot des Wiki-Eintrages befindet sich im Anhang A.5: Benutzerdokumentation auf Seite xiv.

8.2 Zwischenstand

Tabelle 8 zeigt den Zwischenstand nach der Dokumentation.

Vorgang	Geplant	Tatsächlich	Differenz
Erstellen der Projektdokumentation	7 h	8.5 h	+1.5 h
Erstellen der Benutzerdokumentation	2 h	0.5 h	-1.5 h
Dokumentation	**9 h**	**9 h**	

Tabelle 8: Zwischenstand nach der Dokumentation

9 Fazit

9.1 Soll-/Ist-Vergleich

An der folgenden Tabelle 9 kann die Abweichung von der geplanten Projektzeit abgelesen werden. So wurde besonders in den Teilbereichen, bei denen die Datenübertragung zwischen den Systemen umgesetzt werden sollte mehr Zeit beansprucht, als geplant.

Phase	Geplant	Tatsächlich	Differenz
Analysephase	*6 h*	*6.5 h*	*+0.5 h*
Entwurfsphase	*11 h*	*11.5 h*	*+0.5 h*
Implementierungsphase	*39 h*	*44 h*	*+5 h*
Abnahme	*3 h*	*2.5 h*	*-0.5 h*
Einführung	*2 h*	*0.5 h*	*-1.5 h*
Dokumentation	**9 h**	**9 h**	
Gesamt	**70 h**	**75 h**	**+5 h**

Tabelle 9: Soll-/Ist-Vergleich

9.2 Lessons Learned

Die Ursachen für diese Zeitabweichung von der geplanten Zeit stellen auch eine der am nachhaltigsten gelernten Lektionen dar. Besonders schwierig sind Systeme zu testen, wenn man währenddessen immer wieder auf Antworten von Kollegen mit eigenem Zeitplan angewiesen ist. Deshalb sollte in solchen Situationen immer eine gewisse Pufferzeit für Abweichungen mit einkalkuliert werden. Auch ist die fertige Umsetzung des Projektes zu großen Teilen den im Vorfeld definierten Testfällen zu verdanken, die beim programmieren nach TDD-Prinzipien bereits während der Erstellung eines Features mehr Freiheiten beim Refactoring von bestehender Logik verschaffen.

9.3 Ausblick

Sobald die benötigten Features umgesetzt sind und das Projekt im Produktivsystem zur Accounterstellung genutzt wird, werden neben evtl. Bugfixes wahrscheinlich weitere Funktionsanfragen aus den Abteilungen in das Formular integriert werden. Das setzen sonstiger buchbaren Optionen eines Vertrages im Account stellt eine bereits angedachte Erweiterungsmöglichkeit dar. Auch eine zusätzliche Erstellung der Einträge zu den Kunden in unserem internen Billing-System zur automatischen Integration in unseren Abrechnungsprozess ist denkbar.

Eidesstattliche Erklärung

Ich, Andreas Biller, 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 24.05.2018

_____

Andreas Biller

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	1 h	1 h	
Wirtschaftlichkeitsanalyse und Amotisationsrechnung	1 h	1 h	
Ermittlung von Use-Cases	1 h	0.5 h	-0.5 h
Qualitätsanforderungen und Lastenheft	3 h	4 h	+1 h
Analysephase	**6 h**	**6.5 h**	**+0.5 h**
Entwurf des überarbeiteten Eingabeformulars	2 h	1 h	-1 h
Klassendiagramm für die verteilten Systeme	2 h	3 h	+1 h
Zustandsdiagramm für die Vertragszustände	2 h	1 h	-1 h
Planung manueller Tests auf verteilten Systemen	2 h	4 h	+2 h
Testfälle für das Lastenheft definieren	3 h	2.5 h	-0.5 h
Entwurfsphase	**11 h**	**11.5 h**	**+0.5 h**
Formularfelder des Übergabeprotokolls	4 h	2 h	-2 h
Button zur Accounterstellung, Sichtbarkeit für Benutzergruppen	3 h	1 h	-2 h
Unit-Tests für Formular und Benutzereingabe	4 h	3 h	-1 h
Persistieren der Daten	1 h	1 h	
Unit-Tests für die Datenhaltung	2 h	2 h	
Objekte für Datenversand auf Doctena Standard	10 h	18 h	+8 h
Unit-Tests für die Objekterstellung	5 h	5 h	
Versand über den AMQP-Messaging-Bus	4 h	7 h	+3 h
E-Mail-Versand bei Vertragsabschluss	3 h	1 h	-2 h
Unit-Tests für den Objektversand	3 h	4 h	+1 h
Implementierungsphase	**39 h**	**44 h**	**+5 h**
Tests der CI-Pipeline ausführen	1 h	0.5 h	-0.5 h
Durchführung manueller Tests	2 h	1 h	-1 h
Abnahme	**3 h**	**1.5 h**	**-1.5 h**
Deployment über die -Pipeline	1 h	0.5 h	-0.5 h
Schulung der Benutzer	1 h	0 h	-1 h
Einführung	**2 h**	**0.5 h**	**-1.5 h**
Erstellen der Projektdokumentation	7 h	8.5 h	+1.5 h
Erstellen der Benutzerdokumentation	2 h	0.5 h	-1.5 h
Dokumentation	**9 h**	**9 h**	
Gesamt	**70 h**	**75 h**	**+5 h**

A.3 Mockup

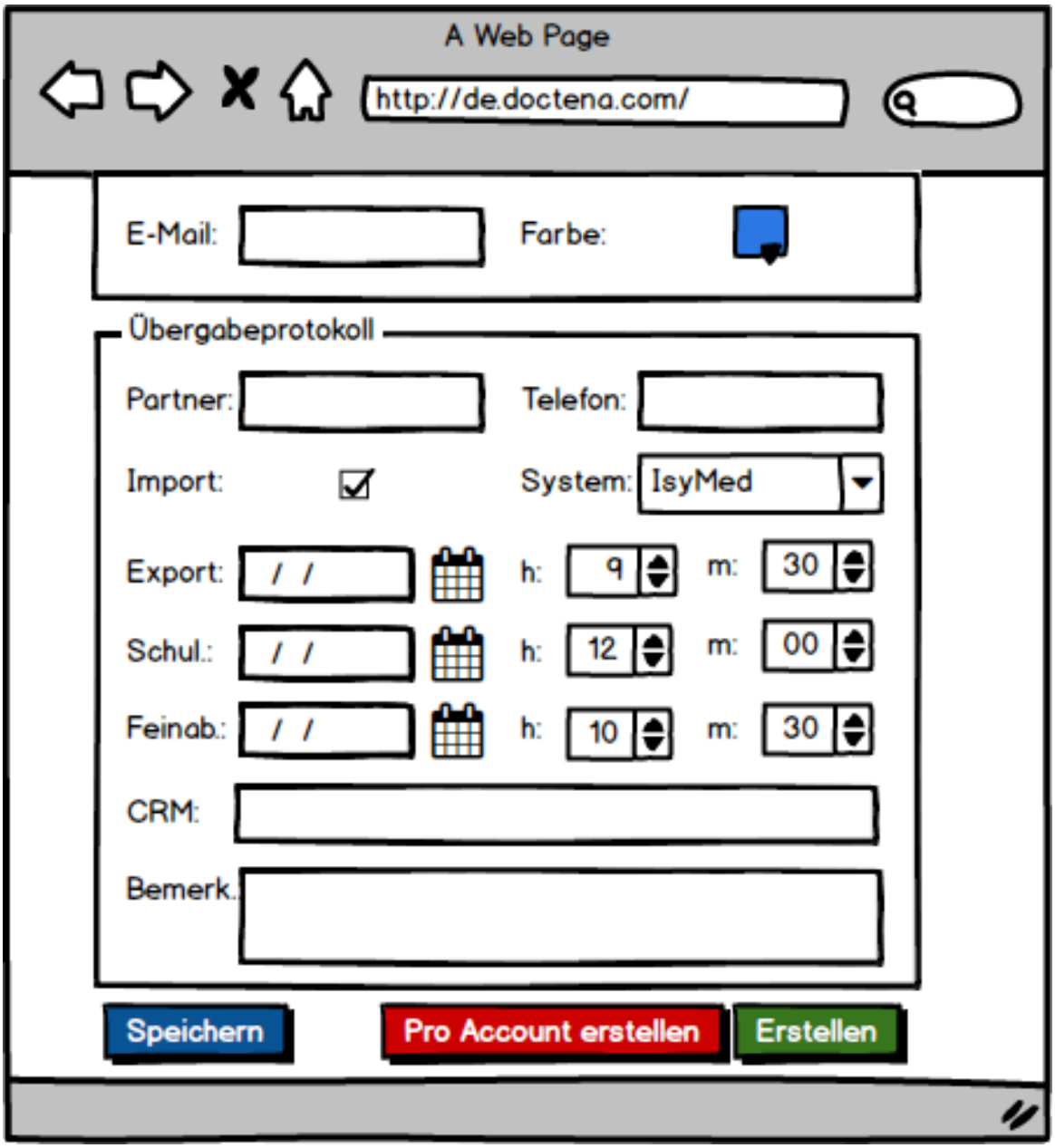


Abbildung 2: Neue Felder und Übergabeprotokoll

A.3.1 HAML-Partial

```
.col-xs-12.customer-data.onboarding.noprint
  .form-horizontal.container-fluid
    .panel.panel-info
      .panel-body
        .panel.panel-default
          .panel-heading
            bergabeprotokoll
          .panel-body.form-horizontal
            .row
              .col-xs-6
                .form-group
                  = f.label :contact_person, 'Ansprechpartner:', class: 'col-xs-4 control-label'
                  .col-xs-8
                    = f.text_field :contact_person, class: 'form-control', disabled: (customer_field && @contract.account?), required: false
                .form-group
                  = f.label :data_import_wanted, 'Import:', class: 'control-label col-xs-4', title: 'Wenn ein Import gewnscht wird, bitte Praxissoftware unter System mit angeben.'
                  .col-xs-8
                    = f.check_box :data_import_wanted, disabled: customer_field, class: 'control-checkbox col-xs-3', title: 'Wenn ein Import gewnscht wird, bitte Praxissoftware unter System mit angeben.'
                .form-group
                  = f.label :data_export_date, 'Exporttermin:', class: 'col-xs-4 control-label'
                  .col-xs-8
                    = f.select :data_export_date, options_for_select(generate_dates([Time.zone.now, @contract.subscription_start].compact.min, 125).map {|d| [l(d.in_time_zone.to_date, format: "%d. %b %Y | %a"), d.in_time_zone.to_date]}, selected: @contract.data_export_date.try(:to_date) ), { include_blank: !customer_field }
                .form-group
                  = f.label :training_date, 'Schulungstermin:', class: 'col-xs-4 control-label'
                  .col-xs-8
                    = f.select :training_date, options_for_select(generate_dates([Time.zone.now, @contract.subscription_start].compact.min, 125).map {|d| [l(d.in_time_zone.to_date, format: "%d. %b %Y | %a"), d.in_time_zone.to_date]}, selected: @contract.training_date.try(:to_date) ), { include_blank: !customer_field }
                .form-group
                  = f.label :setup_date, 'Feinabstimmung:', class: 'col-xs-4 control-label'
                  .col-xs-8
                    = f.select :setup_date, options_for_select(generate_dates([Time.zone.now, @contract.subscription_start].compact.min, 125).map {|d| [l(d.in_time_zone.to_date, format: "%d. %b %Y | %a"), d.in_time_zone.to_date]}, selected: @contract.setup_date.try(:to_date) ), { include_blank: !customer_field }
              .col-xs-6
                .form-group
                  = f.label :contact_person_phone, 'Telefonnummer:', class: 'col-xs-4 control-label'
                  .col-xs-8
                    = f.text_field :contact_person_phone, class: 'form-control', disabled: (customer_field && @contract.account?), required: false
```

```
36                  .form-group
37                    = f.label :data_import_source, 'System:', class: 'control-label col-xs-4'
38                    .col-xs-8
39                      = f.text_field :data_import_source, list: 'data_import_sources', class: 'form-control', disabled
                            : (customer_field && @contract.account?), required: false
40                      %datalist#data_import_sources
41                        %option
42                          Albis
43                        %option
44                          DocComfort
45                        %option
46                          DocConcept
47                        %option
48                          IndaMed
49                        %option
50                          Isynet
51                        %option
52                          MediStar
53                        %option
54                          Turbomed
55                  .form-group
56                    = f.label :data_export_date_hour, 'Stunde:', class: 'col-xs-2 control-label'
57                    .col-xs-4
58                      = f.select :data_export_date_hour, options_for_select([*0..23], selected: @contract.
                            data_export_date_hour), { include_blank: !customer_field }
59                    = f.label :data_export_date_minute, 'Minute:', class: 'col-xs-2 control-label'
60                    .col-xs-4
61                      = f.select :data_export_date_minute, options_for_select([*0..59], selected: @contract.
                            data_export_date_minute), { include_blank: !customer_field }
62                  .form-group
63                    = f.label :training_date_hour, 'Stunde:', class: 'col-xs-2 control-label'
64                    .col-xs-4
65                      = f.select :training_date_hour, options_for_select([*0..23], selected: @contract.
                            training_date_hour), { include_blank: !customer_field }
66                    = f.label :training_date_minute, 'Minute:', class: 'col-xs-2 control-label'
67                    .col-xs-4
68                      = f.select :training_date_minute, options_for_select([*0..59], selected: @contract.
                            training_date_minute), { include_blank: !customer_field }
69                  .form-group
70                    = f.label :setup_date_hour, 'Stunde:', class: 'col-xs-2 control-label'
71                    .col-xs-4
72                      = f.select :setup_date_hour, options_for_select([*0..23], selected: @contract.setup_date_hour
                            ), { include_blank: !customer_field }
73                    = f.label :setup_date_minute, 'Minute:', class: 'col-xs-2 control-label'
74                    .col-xs-4
75                      = f.select :setup_date_minute, options_for_select([*0..59], selected: @contract.
                            setup_date_minute), { include_blank: !customer_field }
76                .col-xs-12
77                  .form-group
78                    = f.label :crm_link, 'close.io-Link:', class: 'col-xs-2 control-label'
79                    .col-xs-10
```

```
80                    = f.text_field :crm_link, class: 'form-group-single form-control'
81                .col-xs-12
82                  .form-group
83                    = f.label :onboarding_notes, 'Anmerkungen:', class: 'col-xs-2 control-label'
84                    .col-xs-10
85                      = f.text_area :onboarding_notes, class: 'form-control form-group-single'
```

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?
```

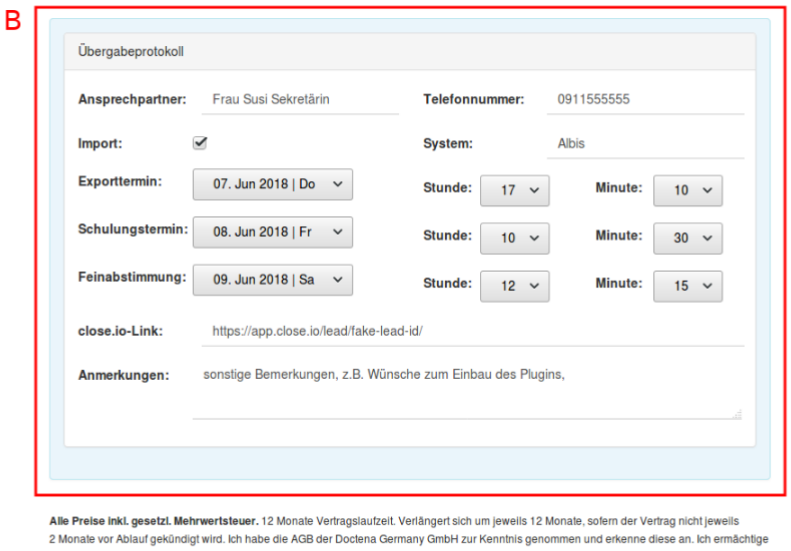
```
253         method = 'created'
254         @contract.practitioners.each do |practitioner|
255           doctor = create_doctor_for_bus(account, practice, practitioner)
256           @contract.doctors << doctor
257
258           calendar = create_calendar_for_bus(account, practice, practitioner, doctor)
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



Beschreibung der Funktionen, neuen Formularfelder und des automatischen Onboarding-Prozesses:

- Die zur automatischen Accounterstellung in Doctena Pro benötigten Daten können sowohl von Verkäufern wie auch Onboarding-Managern über das Angebotsformular eingegeben werden.
- Der Button zum Auslösen der Accounterstellung über den Bus ist generell nur für Onboarding-Manager sichtbar. Der Button ist nur dann sichtbar, wenn der Vertrag mit dem Kunden geschlossen und noch keine Accounterstellung in Doctena Pro vorgenommen wurde.
- Die neuen Eingabefelder (siehe Screenshot unter A und B) sind beim Anlegen eines neuen Angebots durch die Verkäufer nicht zwingend auszufüllen und können auch vom Onboarding-Manager nachträglich bearbeitet werden.
- Im Feld E-Mail unter Punkt A wird die E-Mail zum Login des Arztes in Doctena Pro angelegt.
- Im Feld Farbe unter Punkt A wird die Kalenderfarbe des Arztes in Doctena Pro festgelegt.
- Als Passwort wird jeweils das Standardpasswort für neue Benutzer verwendet, dieses kann nachträglich von Onboarding-Manager und Kunde nach Erstellung des Accounts auf Doctena Pro geändert werden.
- Bei einem Vertragsabschluss zu Doctena Pro über das Angebotssystem, sowohl Kundenseitig wie auch durch einen Verkäufer, erhält die Abteilung Onboarding eine Benachrichtigung per E-Mail. Diese enthält einen Link zum entsprechenden Vertrag.
- Die vorhandenen Daten des Angebots sowie die neuen Felder unter A müssen vom Onboarding-Manager vor Accounterstellung auf ihre Richtigkeit geprüft werden.
- Sind die Daten vom Onboarding-Manager kontrolliert und ggf. geändert worden, wird über den roten "Account erstellen"-Button am unteren Seitenende die Accounterstellung über den Bus ausgelöst.
- Die unter B aufgeführten Formularfelder für das Übergabeprotokoll sollen von Verkäufern und Onboarding-Managern dazu genutzt werden, die im weiteren Onboarding-Prozess zu planenden Termine an einer Stelle einzutragen.

Abbildung 4: Auszug aus der Benutzerdokumentation