HAW HAMBURG

BACHELORTHESIS
Robert Kossendey

Domänenspezifische Texterkennung am Beispiel der automatischen Rechnungsverarbeitung: Konzeption und Realisierung

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Robert Kossendey

Domänenspezifische Texterkennung am Beispiel der automatischen Rechnungsverarbeitung: Konzeption und Realisierung

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Bachelor of Science Wirtschaftsinformatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Olaf Zukunft
Zweitgutachter: Prof. Dr. Ulrike Steffens

Eingereicht am: 28. September 2020

Robert Kossendey

Thema der Arbeit

Domänenspezifische Texterkennung am Beispiel der automatischen Rechnungsverarbeitung: Konzeption und Realisierung

Stichworte

Domänenspezifische Informationsextraktion, Optische Zeichenerkennung, Prozess Automatisierung

Kurzzusammenfassung

Im Mittelpunkt dieser Arbeit steht die domänenspezifische Texterkennung am Beispiel der automatischen Rechnungsverarbeitung in der Versicherungsbranche. Im Rahmen der Abwicklung eines Schadensfalls werden von den Beteiligten Dokumente in verschiedenen Formaten, zum Beispiel EML oder PDF, ausgetauscht. Um die enthaltenen Daten weiter zu verarbeiten können, müssen sie aus den vorliegenden Dokumenten extrahiert werden. Um diesen, meist noch manuellen Prozess, zu automatisieren, wurde in dieser Arbeit ein System entwickelt, das die Möglichkeit zulässt, aus zwei verschiedenen Formaten von EML-Dateien die enthaltenen Daten zu extrahieren und Texterkennung auf PDF-Dateien laufen zu lassen. Die EML-Verarbeitung wurde an 1000 Sample-Dateien getestet und ergab bei der Extraktion der Daten eine Fehlerquote von 29 Prozent. Bei der Texterkennung wurden eine Open Source Lösung (Tesseract) und eine proprietäre Lösung (Amazon Textract) an verschiedenen Sample-Dateien getestet und die Testergebnisse miteinander verglichen. Amazon Textract schneidet dabei, bezogen auf den Anwendungsfall, mit einer Fehlerquote von 4 Prozent deutlich besser ab als Tesseract, das eine Fehlerquote von 17 Prozent aufweist.

Robert Kossendey

Title of Thesis

Automatic invoice recognition: Conception and implementation

Keywords

Domain specific information extraction, Optical character recognition, process automation

Abstract

This thesis focuses on domain-specific text recognition using the example of automatic invoice processing in the insurance industry. In the course of processing a claim, the parties involved exchange documents in different formats, for example EML or PDF. In order to automate this process, which is usually still manual, a system was developed that allows the data contained in two different formats of EML files to be extracted and text recognition to be run on PDF files. The EML processing was tested on 1000 sample files and resulted in an error rate of 29 percent when extracting the data. For text recognition, an open source solution (Tesseract) and a proprietary solution (AmazonTextract) were tested on different sample files and the test results were compared. Amazon Textract scored significantly better than Tesseract, with an error rate of 4 percent and an error rate of 17 percent.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation

In der Versicherungswirtschaft wurden im Jahr 2018 im Bereich der Sachversicherungen, Leistungen im Wert von über 15 Milliarden Euro ausgezahlt [13]. Da Versicherungen größtenteils gewinnorientierte Unternehmen sind, liegt es in ihrem Interesse, Ausgaben so gering wie möglich zu halten.

Ein wichtiger Faktor, welcher die Höhe der ausgezahlten Leistung beeinflusst, ist die Bewertung des Schadens durch einen Gutachter. Dieser fährt zu dem vom Schaden betroffenen Kunden und überprüft ob die vom Kunden angegebenen Informationen korrekt sind oder ob sich der Schaden unter Umständen anders darstellt als beschrieben. Die Beauftragung eines Gutachters lohnt sich jedoch erst ab einer bestimmten Schadenhöhe.

Viele Versicherungen greifen in einem solchen Falle auf Gutachter von Drittanbietern zurück, die zum Schadenort fahren und einen Bericht erstellen. Ein Gutachter benötigt hierfür bestimmte Informationen, wie zum Beispiel die Adresse des Versicherungsnehmers oder die geschätzte Schadenshöhe. Daher muss die Versicherung dem Drittanbieter die benötigten Informationen zukommen lassen. Diese können per Schnittstelle, aber auch per E-Mail oder PDF übersandt werden.

Meist bieten die Drittanbieter ihren Experten eine App oder ein Webportal an, auf welchem diese die Schadendaten einsehen, Berichte schreiben und weitere Bearbeitungen durchführen können. Die vom Versicherer vermittelten Daten müssen jedoch erst in dieses System eingepflegt werden. Informationen, welche die Versicherungen per E-Mail oder PDF übermitteln, werden meist per Hand in die Applikation der Drittanbieter eingespeist. Diese Vorgehensweise ist sehr zeitaufwendig und dementsprechend kostenintensiv. Aufgrund dessen ist es vorteilhaft, diesen Prozess zu automatisieren. Deshalb ist die automatische Rechnungsverarbeitung und die dazu gehörige domänenspezifische Texterkennung das Thema meiner Bachelorarbeit.

Problematisch ist, dass weder die E-Mails, noch die dazugehörigen PDF-Anhänge, standardisiert sind. Das Format der E-Mails ist von Versicherung zu Versicherung unterschiedlich. Hinzukommt, dass der Text, den die PDFs enthalten, nicht direkt, sondern nur durch optische Zeichenerkennung extrahiert werden kann.

1.2 Vorstellung claimsforce

Diese Bachelorarbeit ist in Zusammenarbeit mit der claimsforce GmbH entstanden. Diese bietet Software as a Service Lösungen für Versicherer und Schadenregulierer an. Gleichzeitig stellt sie den Versicherungen ein Gutachternetzwerk zur Verfügung, das mit den eigenen Systemen arbeitet.

1.3 Aufgabe und Ziele

Das zu implementierende fertige System soll, sobald eine E-Mail eintrifft, den Inhalt und die zugehörigen Anhänge verarbeiten und in das System pflegen. Da die eintreffenden E-Mails auf Grund von unterschiedlichen Systemen auf der Seite der Versicherer viele verschieden Formate haben können, wird sich auf Grund von zeitlichen Einschränkungen bei dem System auf zwei dieser Formate konzentriert.

Aus den PDFs soll mit optischer Zeichenerkennung Text extrahiert werden. Diesbezüglich werden zwei verschiedene Softwarelösungen, eine Open-Source, die andere proprietär, getestet und dem Anwendungsfall entsprechend verglichen und bewertet. Der Vergleich wird sich auf Performance, Funktionalität und Kosten beziehen.

Ziel des zu implementierenden Systems ist die Reduzierung von, für die Schadenanlage benötigte, Arbeitszeit und die damit verbundene Reduzierung von Kosten. Darüber hinaus wird die manuelle Schadenanlage von den verantwortlichen Mitarbeitern als unangenehm und lästig empfunden. Daher wird das System zusätzlich die Mitarbeiter von störenden Prozessen entlasten.

Auf genaue Anforderungen wird später eingegangen.

1.4 Gliederung

Im Folgenden werde ich die Gliederung meiner Arbeit erläutern.

Im zweiten Kapitel werden die Grundlagen, die zum Verstehen der Arbeit erforderlich sind, erläutert, um ein einheitliches Verständnis der Begriffe zu schaffen. Zuerst wird auf Daten, deren jeweilige Struktur und die für den Anwendungsfall relevanten Dateiformate eingegangen. Im Anschluss wird der Begriff der Optischen Zeichenerkennung dargestellt. Außerdem wird die Levenshtein Distanz erläutert, die für die spätere Auswertung der Ergebnisse der optischen Zeichenerkennung genutzt wird. Da ein Großteil des Systems in der Cloud realisiert werden soll, wird zum Abschluss des Grundlagenkapitels auf Cloud Computing generell und die Amazon Web Services eingegangen.

Das dritte Kapitel wird sich mit den verschiedenen Anforderungen, die an das fertige System gestellt werden, beschäftigen.

Im vierten Kapitel wird die konkrete Infrastruktur vorgestellt, welche für die Realisierung des Systems genutzt wurde.

Das fünfte Kapitel beinhaltet den Entwurf. An dieser Stelle werden die Architekturentscheidungen, die getroffen wurden, vorgestellt und begründet. Darüber hinaus werden Konzepte wie Infrastructure as Code, Continuous Integration und Continuous Deployment vorgestellt. Abschließend wird in diesem Kapitel auf die Tests eingegangen.

Das sechste Kapitel beschreibt den Prozess der Realisierung und welche Schwierigkeiten sich dabei ergaben. Das realisierte System wird in Bezug auf die gestellten Anforderung untersucht.

Im siebten Kapitel wird das Systems getestet. Hierfür wird auf den Testaufbau und die Testergebnisse eingegangen. Außerdem wird ein Überblick über eine mögliche methodische Abstraktion gegeben.

Den Abschluss der Arbeit bildet das Fazit, das die Bewertung der Ergebnisse, eine Zusammenfassung und einen Ausblick enthält.

2 Grundlagen

In diesem Kapitel werden Grundlagen, die für das Verständnis des Systems von Nöten sind, erläutert.

2.1 Datenarten und -formate

Die Unterschiede zwischen strukturierten und unstrukturierten Daten sind zu überprüfen, und wie im Falle der Informationsextraktion mit ihnen umgegangen werden muss. Außerdem werden die beiden Dateiformate PDF und EML dargestellt, welche von dem entwickelten System verarbeitet werden. Wichtig ist, den Unterschied zwischen strukturierten Daten an sich und strukturiertem Inhalt von Daten klar aufzuzeigen.

2.1.1 Strukturierte Daten

Bei strukturierten Daten handelt es sich um Daten, die in einer bestimmten Struktur vorliegen. Durch die Struktur können die einzelnen Werte identifiziert werden.

Ein Beispiel für eine solche Struktur ist das Datenbankmodell einer relationalen Datenbank. Eine relationale Datenbank enthält eine Menge von Relationen, welche die Daten enthalten. Relationen sind in Tabellenform angeordnet, wobei die Spalten als Attribute und die Reihen als Tupel bezeichnet werden. Zu jeder Kombination aus Attribut und Tupel existiert eine Zelle, die einen Wert in einem bestimmten Datentyp enthält. Der Datentyp wird durch das Attribut bestimmt. Die Tupel können durch einen eindeutigen Schlüssel identifiziert werden. Dadurch kann in einer relationalen Datenbank mit dem Namen der Relation, dem Schlüssel und dem Attribut auf jeden einzelnen Wert zugegriffen werden [27]. Durch die starke Typisierung werden bestimmte Operationen auf Attributen, wie die Summierung, möglich.

2.1.2 Semistrukturierte Daten

Bei dem Begriff 'semistrukturiert' liegt in Bezug auf Daten eine gewisse Ambiguität vor. Manche semistrukturierte Daten haben ein loses Schema, andere gar keins. Peter Buneman schreibt Folgendes über semistrukturierte Daten:

"In semistrukturierten Daten sind die Informationen, welche normalerweise mit einem Schema verbunden sind, in den Daten enthalten, was manchmal als 'selbstbeschreibend' bezeichnet wird."[26]

Ein Beispiel dafür sind JSON-Dateien. In diesem Format unterliegen Dateien keinem externen Schema, sondern beschreiben ihre Struktur selber. Die Informationen, die eine JSON-Datei enthält, sind an sich unstrukturiert. Ein weiteres Beispiel für ein semistrukturiertes Dateiformat ist das EML-Format, auf welches in 2.1.4 näher eingegangen wird.

2.1.3 Unstrukturierte Daten

Unstrukturierte Daten unterliegen keiner Struktur, wodurch es unmöglich, ist a priori Schlüsse über solche Daten zu ziehen. Beispiele dafür sind der Body einer E-Mail oder Audiodateien. Unstrukturierte Daten machen mit ungefähr 95 Prozent den größten Teil von Big Data aus [28], wodurch die Verarbeitung von unstrukturierten Daten zu einer grundlegenden Herausforderung für Data Scientists wird.

2.1.4 Electronic Mail Format

Das Electronic Mail Format, kurz Electronic Mail Format (EML), ist eine Dateierweiterung für die Speicherung von E-Mail-Nachrichten gemäß des Internet Message Format Protokolls. EML-Dateien können von den meisten E-Mail Clients geöffnet werden.

Das EML hat keine eigene Spezifikation und wird laut Library of Congress durch die Einhaltung des RFC 2322 als eine Erweiterung des Internet Message Formats (IMF) angesehen [11]. Durch die Einhaltung dieses Standards liegen EML-Dateien in semistrukturierter Form vor. Eine EML-Datei kann in zwei Teile zerlegt werden, den Header und den Body. Der Header enthält zwei notwendige Felder, das Ursprungsdatum der Nachricht und den Absender. Weitere Felder wie der Betreff sind optional [22].

Alle laut RFC 5322 möglichen Header-Felder sind in der folgenden Abbildung zu sehen.

Field	Min number	Max number	Notes
trace	0	unlimited	Block prepended - see 3.6.7
resent-date	0*	unlimited*	One per block, required if other resent fields are present - see 3.6.6
resent-from	0	unlimited*	One per block - see 3.6.6
resent-sender	0*	unlimited*	One per block, MUST occur with multi-address resent-from - see 3.6.6
resent-to	0	unlimited*	One per block - see 3.6.6
resent-cc	0	unlimited*	One per block - see 3.6.6
resent-bcc	0	unlimited*	One per block - see 3.6.6
resent-msg-id	0	unlimited*	One per block - see 3.6.6
orig-date	1	1	
from	1	1	See sender and 3.6.2
sender	0*	1	MUST occur with multi-address from - see 3.6.2
reply-to	0	1	
to	0	1	
cc	0	1	
bcc	0	1	
message-id	0*	1	SHOULD be present - see 3.6.4
in-reply-to	0*	1	SHOULD occur in some replies - see 3.6.4
references	0*	1	SHOULD occur in some replies - see 3.6.4
subject	0	1	
comments	0	unlimited	
keywords	0	unlimited	
optional-field	0	unlimited	

Abbildung 2.1: Notwendige und optionale Header-Felder für Nachrichten im IMF [22]

Der Body enthält gemäß RFC 5322 simplen Text in ASCII-Zeichen. Um im Body andere Objekte als Text in ASCII-Zeichen zu verschicken, gibt es die Multipurpose Internet Mail Extensions (MIME) welche in RFC 2045 definiert werden. Das RFC 2045 definiert das Format des IMF-Bodies neu, um folgendes zu ermöglichen:

1. Textnachrichten Bodies in anderen Zeichensätzen als US-ASCII
2. Einen erweiterbaren Satz verschiedener Formate für nicht-textuelle Bodies
3. Mehrteilige Bodies
4. Header in anderen Zeichensätzen als US-ASCII

[20]

Dadurch wird es möglich, Bilder oder die für den Anwendungsfall relevanten PDF Dateien als Anhänge einer E-Mail zu verschicken. Jeder Teil eines Bodies hält einen Content-Type-Header, welcher das Format, in dem der Body vorliegt, spezifiziert [21].

2.1.5 Portable Document Format

Das Portable Document Format (Portable Document Format (PDF)) ist ein Dateiformat, welches von Adobe 1993 mit dem Ziel, eine umgebungsunabhängigen Wiedergabe von elektronischen Dokumenten zu ermöglichen, veröffentlicht wurde. Aktuell wird das Format von der PDF Association weiterentwickelt und unterliegt in seiner 2.0 Version der ISO 32000-2 Norm. Die PDF 2.0 Dokumentation wurde nicht veröffentlicht, da diese ausschließlich von der ISO-Gruppe ISO/TC 171/SC 2 entwickelt wurde. Da die PDF-Dateien im Anwendungsfall in der Version 1.4 vorliegen, werde ich mich im weiteren Verlauf auf die Dokumentation dieser Version beziehen, welche von Adobe frei zugänglich gemacht wurde.

Die Ursprünge des PDFs liegen im 1991 von John Warnock ins Leben gerufenen 'Camelot Project'. John Warnock definiert das Ziel dieses Projektes folgendermaßen:

"Ziel dieses Projekts ist es, ein grundlegendes Problem zu lösen, mit dem die Unternehmen von heute konfrontiert sind. [...] Was die Industrie dringend benötigt, ist eine universelle Möglichkeit zur Übermittlung von Dokumenten über eine Vielzahl von Maschinenkonfigurationen, Betriebssystemen und Kommunikationsnetzwerken. Diese Dokumente sollten auf jedem Bildschirm angezeigt und auf jedem modernen Drucker ausgedruckt werden können. Wenn dieses Problem gelöst werden kann, dann wird sich die grundlegende Arbeitsweise der Menschen ändern."[44]

Zu diesem Zeitpunkt existierte schon ein bedeutender Ansatz zur Lösung dieses Problems, die von Adobe entwickelte PostScript-Sprache, welche eine umgebungsunabhängige Seitenbeschreibungssprache ist. PostScript war durch die Implementation des Interpreters in über hundert verschiedenen Druckern und den Umstand, dass über 4000 Applikationen diese Drucker unterstützten, ein ernstzunehmender Anwärter für die Lösung. Doch über die Notwendigkeit einen Drucker zu besitzen, der PostScript unterstützt, hinaus, erforderte die Nutzung PostScripts eine hohe Rechenleistung. Um die Vorteile, die PostScript ermöglichte, auch einer breiten Masse von Nutzern zur Verfügung zu stellen, wurde das PDF entwickelt [44].

Das PDF besteht aus drei unterschiedlichen Modulen: Das erste Modul ist eine abgeänderte Version von PostScript, um das Layout und die Grafiken zu generieren; das Zweite ist ein System, welches ermöglicht, Schriftarten mit dem Dokument zu verschicken; das Dritte schließlich ist ein Speichersystem, das die vorgenannten Module und den Inhalt des PDFs in einer einzigen Datei abspeichern kann. Der Markt reagierte zunächst verhalten auf die Einführung des PDFs und eine breite Adaption des Formats blieb aus. Da dies auch unter anderem dem Umstand geschuldet war, dass Adobe Acrobat, jenes Programm, um PDFs zu erstellen und zu lesen, nicht frei verfügbar war, entschloss sich Adobe 1994, Acrobat mit dem Release der Version 2.0 kostenlos zur Verfügung zu stellen, wobei das Format an sich vorerst proprietär blieb. Im Jahre 2008 entschied sich Adobe dazu, die Rechte am PDF abzugeben, wodurch im Juli 2008 die erste PDF Version, 1.7, die einem ISO-Standard unterliegt, erschien [14].

Im Folgenden wird näher auf das Format der Datei an sich eingegangen. Die PDF 1.4 Dokumentation definiert eine Datei im PDF Format folgendermaßen:

“Ein PDF Dokument besteht aus einer Sammlung von Objekten, die zusammen das Erscheinungsbild einer oder mehrerer Seiten beschreiben, möglicherweise begleitet von zusätzlichen interaktiven Elementen und übergeordneten Anwendungsdaten. Eine PDF Datei enthält die Objekte, aus denen sich ein PDF-Dokument zusammen mit den zugehörigen Strukturinformationen zusammensetzt, welche alle als eine einzige in sich geschlossene Folge von Bytes dargestellt werden.” [18]

Die Seiten eines Dokuments können Text, Grafiken und Bilder enthalten. Diese werden inklusive ihrer Layout- und Formatierungskonfiguration im sogenannten ’content stream’ der Seite beschrieben. Der ’content stream’ enthält eine Folge von Grafikobjekten, welche dann auf einer Seite dargestellt werden. Ein Dokument kann auch interaktive Elemente, wie Videos, Töne oder Hyperlinks enthalten. Darüber hinaus können Aktionen durch Tastatur- oder Mausinput ausgelöst werden. Diese Aktionen reichen vom Abspielen eines Tones bis zu der Ausführung von JavaScript Code. Außerdem kann ein PDF Formularfelder enthalten, welche entweder eine Eingabe vom Nutzer erwarten und diese exportieren, oder Daten aus externen Applikationen importieren kann. Zusätzlich dazu kann ein Dokument auch noch übergeordnete Daten enthalten wie strukturelle Informationen, wodurch die Suche in Dokumenten ermöglicht wird.

2.2 Optical Character Recognition

An dieser Stelle wird erklärt, was Optical Character Recognition (OCR) ist, ohne auf die genaue Funktionsweise solcher Software oder Hardware einzugehen, da diese in der Tiefe im vierten Kapitel, wenn die verwendeten Implementationen erläutert werden, behandelt werden.

Unter optischer Zeichenerkennung wird die Übertragung von auf Bildern gedrucktem oder handgeschriebenem Text in eine von Maschinen lesbare Form verstanden. Hierbei kann es sich um eingescannte Dokumente, aber auch um Fotos handeln, welche Text, zum Beispiel Straßenschilder, enthalten. Im Folgenden wird auf die Geschichte der optischen Zeichenerkennung eingegangen:

Die optische Zeichenerkennung hat ihre Anfänge im späten neunzehnten und frühen zwanzigsten Jahrhundert. Zwischen 1875 und 1930 wurden die ersten Grundsteine für die optische Zeichenerkennung erfunden, wie die Nipkow-Scheibe. Die ursprüngliche Intention für die Erfindung solcher Maschinen war die Unterstützung von sehbehinderten Menschen [32]. 1929 gelang es Gustav Tauschek, die erste Maschine zur optischen Zeichenerkennung zu erfinden, die 'Reading machine' [43]. Dieses Gerät ermöglichte mit Hilfe eines Photodetektors die Erkennung von Buchstaben und Zeichen. Licht fällt durch eine mechanische Maske und wird von dem Photodetektor erfasst. Bei einem exakten Match würde den Photodetektor kein Licht erreichen, womit das Zeichen erkannt wäre. Bei dieser Methode wird eines der großen Probleme der optischen Zeichenerkennung deutlich. Ein Mensch würde 'ε' als 'E' erkennen, die Maschine jedoch nicht [37].

Mit dem Aufkommen elektronischer Rechner konnten große Fortschritte in der Entwicklung der optischen Zeichenerkennung erzielt werden. 1974 gelang dem bekannten Erfinder Ray Kurzweil ein Durchbruch. Er entwickelte eine schriftart-agnostische OCR Software, die 1978 mit der Hilfe einer Finanzierung des Technologieunternehmens Xerox auf den Markt gebracht wurde [15].

Seitdem hat die optische Zeichenerkennung rasante Fortschritte erzielt, sie wird inzwischen in vielen verschiedenen Gebieten angewendet. Beispiele hierfür sind das Projekt Gutenberg, welches lizenzfreie Bücher digitalisiert und bereitstellt [19] oder die automatische Nummernschilderkennung, welche bei Geschwindigkeitsmessungen zum Einsatz kommt. Inzwischen gibt es auch OCR-Software die Open-Source ist, zum Beispiel OCRopus [17] oder Tesseract, auf die in Abschnitt 4.9 näher eingegangen wird. Darüber hinaus existieren auch OCR-Implementationen in der Cloud, wie Amazons Textract oder Googles

Cloud-Vision-API. Die optische Zeichenerkennung ist auch nicht vom Hype um Machine Learning verschont geblieben. Viele OCR-Systeme, wie auch Tesseract oder Amazon Textract, implementieren inzwischen Machine Learning Ansätze, um die Fehlerrate bei der Identifikation von Zeichen zu reduzieren.

2.3 Levenshtein-Distanz

Die Levenshtein-Distanz, auch Editierdistanz genannt, ist eine Metrik für den Vergleich von zwei Zeichenketten. Sie wurde 1965 von dem russischen Wissenschaftler Wladimir Lewenstein eingeführt, nach dem sie auch benannt wurde [34]. Die Levenshtein-Distanz repräsentiert die minimale Anzahl von Operationen um die erste Zeichenkette in die zweite zu konvertieren. Operationen können hierbei das Löschen, das Einfügen oder das Ersetzen eines Buchstabens sein. Jede Operation inkrementiert die Levenshtein-Distanz zwischen den beiden Zeichenketten um 1. Dies bedeutet, dass, umso höher die Levenshtein-Distanz zwischen zwei Zeichenketten ist, desto größer ist der Unterschied.

Um die Levenshtein-Distanz zu berechnen, wird zuerst eine Matrix M vom Typ (m+1,n+1) aufgestellt, wobei m für die Länge der ersten und n für die Länge der zweiten Zeichenkette steht. Der Algorithmus iteriert durch alle Spalten und Reihen, was in einer Laufzeit von $O(mn)$ resultiert.

2.4 Cloud Computing

Die Definitionen von Cloud Computing variieren von Quelle zu Quelle. Eine der bekanntesten Definitionen ist die des National Institute of Standards and Technology (NIST). Diese wird vom Bundesamt für Sicherheit in der Informationstechnik folgendermaßen übersetzt:

"Cloud Computing ist ein Modell, das es erlaubt bei Bedarf, jederzeit und überall bequem über ein Netz auf einen geteilten Pool von konfigurierbaren Rechnerressourcen (z. B. Netze, Server, Speichersysteme, Anwendungen und Dienste) zuzugreifen, die schnell und mit minimalem Managementaufwand oder geringer Serviceprovider-Interaktion zur Verfügung gestellt werden können." [10]

Dazu charakterisieren gemäß NIST die folgenden fünf Eigenschaften einen Cloud Service: [36] [10]

1. **On-demand Self Service:** Die Allokation von Ressourcen wie Speicherplatz oder Rechenleistung an die Nutzer geschieht nicht durch Interaktion mit dem Service Provider, sondern automatisch.

2. **Broad Network Access:** Die Kommunikation mit den Services geschieht über das Internet durch standadisierte Netzwerk-Mechanismen, und nicht mit Hilfe von speziellen Clients.

3. **Resource Pooling:** Die Ressourcen, welche von dem Anbieter zur Verfügung gestellt werden, befinden sich in einem Pool, aus welchem sich die einzelnen Nutzer bedienen können. Hierbei kann der Nutzer lediglich entscheiden, in welcher Region oder in welchem Rechenzentrum sich die Ressourcen befinden. Genaue Kenntnis über den Ort dieser Ressourcen hat der Nutzer jedoch nicht.

4. **Rapid Elasticity:** Die jeweiligen Services werden dem Nutzer flexibel und schnell zur Verfügung gestellt, unter Umständen auch automatisch. Im Falle einer unerwartet hohen Nachfrage des Nutzers, können die Ressourcen schnell hochskaliert werden. Somit kann auf der Nutzerseite der Eindruck entstehen, dass die Ressourcen unendlich sind.

5. **Measured Services:** Die Nutzung der Ressourcen kann überwacht und gemessen werden. Die Ergebnisse der Messungen werden den einzelnen Nutzern, der Inanspruchnahme der Services entsprechend, zur Verfügung gestellt.

NIST unterscheidet einzelne Clouds anhand ihres Bereitstellungsmodells (Deployment Model). Grob wird zwischen diesen vier verschiedenen Modellen unterschieden: [36] [10]

1. **Private Cloud:** Hier wird die Cloud-Infrastruktur nur einer einzigen Institution zur Verfügung gestellt. Dabei kann es vorkommen, dass die jeweilige Institution gleichzeitig auch der Betreiber der Cloud ist.

2. **Public Cloud:** Bei der Public Cloud werden die einzelnen Services der Allgemeinheit zugänglich gemacht. Beispiele hierfür sind Microsoft Azure oder Amazon Web Services.

3. **Community Cloud:** Von einer Community Cloud spricht man, wenn mehrere Institutionen mit ähnlichen Interessen und Anforderungen an eine Cloud sich dieselbe Infrastruktur teilen. Diese Community Cloud kann von einer dieser Institutionen betrieben werden oder von außerhalb zur Verfügung gestellt werden.

4. **Hybrid Cloud:** Bei einem Zusammenschluss mehrerer Cloud-Infrastrukturen über standardisierte Schnittstellen, spricht man von einer Hybrid Cloud.

Weiterhin unterscheiden sich Cloud-Systeme auch in ihrem jeweiligen Servicemodell. Hierfür definiert NIST die folgenden drei Servicemodelle: [36] [10]

1. **Infrastructure as a Service (IaaS):** Von Infrastructure as a Service spricht man, wenn der Cloud-Provider dem Nutzer nur Ressourcen wie Datenspeicher, Arbeitsspeicher oder Ähnliches zur Verfügung stellt. Der Kunde kann, darauf aufbauend, auf diesen virtualisierten Ressourcen ein selbstgewähltes Betriebssystem inklusive eigenen Anwendungen laufen lassen.

2. **Platform as a Service (PaaS):** Bei PaaS werden dem Nutzer standardisierte Schnittstellen bereitgestellt. Diese kann er mit seiner Anwendungslogik, die dann in der Cloud läuft, benutzen. Ein Beispiel hierfür wären Datenbankzugriffe. Im Gegensatz zu IaaS hat der Kunde in diesem Fall keinen direkten Zugriff mehr auf das Betriebssystem oder die Hardware, da diese eine Abstraktionsebene tiefer liegen.

3. **Software as a Service (SaaS):** SaaS bietet dem Nutzer die größtmögliche Abstraktion. Hierbei interagiert der Nutzer nur noch mit einer vom Provider bereitgestellten Software. Er muss sich nicht um Datenspeicherung oder Ähnliches kümmern. Im Gegenzug verliert man als Nutzer ein gewisses Maß an Autonomie.

2.4.1 Amazon Web Services

Amazon Web Services (Amazon Web Services (AWS)) ist ein Cloud-Provider und ein Tochterunternehmen des Online-Versandhändlers Amazon. AWS bietet den Nutzern ein Angebot aus über 200 verschiedenen Services (Stand März 2020) in diversen Bereichen wie zum Beispiel Datenspeicherung, Künstliche Intelligenz, Analytics und vielen mehr. AWS ist zum März 2020 laut der Synergy-Research-Group, der größte Cloud-Infrastruktur-Provider, mit einem Marktanteil von 33 Prozent [23].

3 Anforderungen

In diesem Kapitel wird näher auf die Anforderungen an das zu entwickelnde System eingegangen. Die bestehenden Anforderungen werden in drei Bereiche unterteilt. Funktionale Anforderungen beschreiben das Verhalten oder die zu erbringende Leistung des Systems. Qualitätsanforderungen behandeln qualitative Eigenschaften des Systems, wie Änderbarkeit, Wartbarkeit oder Benutzbarkeit. Randbedingungen sind Anforderungen, welche den Rahmen der Entwicklung beschreiben. Beispiele hierfür wären Vorgaben vom Kunden oder Gesetzgeber.

Die Anforderungen wurden in Gesprächen mit einem zuständigen Product-Manager, dem CEO und den anderen Entwicklern der claimsforce GmbH erhoben. Da das Produkt durch die erhobenen Anforderungen auf die Anwendung bei claimsforce zugeschnitten ist, wird zum Ende des Kapitels auf mögliche Veränderungen der Anforderungen eingegangen, sodass das entwickelte System in mehreren Firmen zum Einsatz kommen kann.

3.1 Funktionale Anforderungen

Im Folgenden werden die an das System gestellten funktionalen Anforderungen aufgeführt.

1. funktionale Anforderung: Automatische Identifikation der für die Verarbeitung relevanten E-Mails

Das System soll automatisch E-Mails, welche an ein bestimmtes Postfach gesendet werden, erfassen und auf ein bestimmtes Kriterium im Betreff hin untersuchen. Falls dieses Kriterium erfüllt ist, soll die E-Mail weiterverarbeitet werden. Ansonsten soll die Verarbeitung abgebrochen werden.

2. funktionale Anforderung: Verarbeitung von E-Mails in zwei verschiedenen Formaten

Die für die Verarbeitung geeigneten E-Mails können in unterschiedlichen Formaten verfasst sein. Das System soll zwei ausgesuchte Formate identifizieren und verarbeiten können. Falls eine E-Mail ein anderes Format hat, soll die Verarbeitung abgebrochen werden. Die Ergebnisse sollen in einer Datenbank abgespeichert werden.

3. funktionale Anforderung: Erkennung und Speicherung von fehlenden Werten

Wenn in einer E-Mail Werte nicht ausgefüllt sind, soll das System dies erkennen und vermerken, welcher Wert fehlt.

4. funktionale Anforderung: Erkennung und Speicherung von falschen Werten

Das System soll, nach Möglichkeit, falsche Werte erkennen und diese vermerken.

5. funktionale Anforderung: Verarbeitung und Interpretation von Anhängen

Falls eine eingetroffene E-Mail einen Anhang enthält, soll die angehängte Datei auf ihren Typ überprüft werden. Wenn es ein PDF ist, soll mit Hilfe von optischer Zeichenerkennung so viel Information wie möglich aus dem PDF extrahiert und in einer Datenbank gespeichert werden.

6. funktionale Anforderung: Automatischer Abgleich der Ergebnisse mit von Menschen erzeugten Ergebnissen

Das System soll parallel zu dem in Produktion befindlichen System laufen, bei welchem die Schadendaten von Sachbearbeitern aus den E-Mails extrahiert und abgespeichert werden. Sobald im Altsystem ein Schaden von Menschenhand angelegt wird, soll das System sich die Daten holen, und mit den entsprechenden automatisch extrahierten Daten abgleichen. Die Resultate dieses Abgleichs sollen in einer Datenbank gespeichert werden. Weiterhin soll es eine Kennzahl geben, welche die Durchschnittsperformance des Systems angibt und von der jedes Mal ein Update durchgeführt werden soll, sobald ein neuer Vergleich vollzogen wurde.

3.2 Qualitätsanforderungen

An dieser Stelle werde ich die Qualitätsanforderungen, anhand der in der DIN ISO 9126 definierten Qualitätsmerkmale, beschreiben [33].

Funktionalität:
Das entwickelte System soll allen zuvor genannten funktionalen Anforderungen entsprechen.

Zuverlässigkeit:
Die Funktionalität des Systems soll jederzeit gewährleistet sein und nicht durch unerwartete Eingaben eingeschränkt werden. Falls dem System eine E-Mail übergeben wird, die in einem nicht unterstützten Format ist, soll das System die Verarbeitung abbrechen und eine Fehlermeldung ausgeben.

Benutzbarkeit:
Die Software soll um eine ausführliche und verständliche ReadMe-Datei ergänzt werden, damit andere Entwickler schnell in das Projekt einsteigen können. Darüber hinaus soll das System durch Logging der zu verarbeitenden E-Mails und der Fehlermeldungen größtmögliche Transparenz bieten. Das System wirft eigene Exceptions, sodass die Fehlerursache schnellstmöglich aus den Logs ersichtlich wird.

Effizienz:
Das System soll möglichst effizient arbeiten. Ohne Anhänge soll eine Ausführung nicht länger als zehn Sekunden dauern. Im Falle von beigefügten Anhängen soll es keine konkrete Obergrenze geben, da die Ausführungsdauer in Abhängigkeit zu der Größe der PDFs steht, die variieren kann.

Änderbarkeit:
Neue funktionale Anforderungen, wie zum Beispiel die Verarbeitung eines weiteren E-Mail-Formats, sollen auf Basis von klar definierten Schnittstellen schnell und einfach implementiert werden können.

Übertragbarkeit:
Durch die Entkopplung von Geschäftsregeln und unterliegender Infrastruktur sollen die Geschäftsregeln im Falle eines Cloudanbieter- oder Cloudservice-Wechsels unberührt bleiben. Die erzeugten Daten sollen über ein klar definiertes Schema in einer Datenbank anderen Services zur Verfügung gestellt werden können.

3.3 Randbedingungen

Im Folgenden werden die vom Product-Owner gesetzten Randbedingungen erläutert.

Randbedingung 1:
Als Product-Owner möchte ich, dass das System im Amazon Web Service Ökosystem entwickelt wird, damit es in die Menge der schon bestehenden Services integriert werden kann.

Randbedingung 2:
Als Product-Owner möchte ich, dass die Amazon Web Services Infrastruktur mit Cloudformation und Infrastructure as Code umgesetzt wird, damit Zeit gespart wird und es anderen Entwicklern ermöglicht, die benutzte Infrastruktur besser nachzuvollziehen.

Randbedingung 3:
Als Product-Owner möchte ich, dass das System einen automatisierten Build- und Deploymentprozess hat. Dies soll durch einen CI / CD Prozess in Amazon CodePipeline umgesetzt werden.

Randbedingung 4:
Als Product-Owner möchte ich, dass es eine Staging-Umgebung und eine Production-Umgebung gibt. Die Staging-Umgebung soll für Tests benutzt werden.

3.4 Abstraktion

Im Folgenden wird abschließend darauf eingegangen, inwiefern von den vorangegangenen Anforderungen abstrahiert werden kann, um das System in anderen Umgebungen zu realisieren. Hierbei wird nur auf die Randbedingungen eingegangen, die funktionalen Anforderungen und die Qualitätsanforderungen gelten weiterhin wie bisher.

Ein hypothetischer Product-Owner einer anderen Firma könnte, um Abhängigkeiten zu vermeiden, die Bedingung aufstellen, dass das System ohne jegliche Cloud-Dienste realisiert werden soll. Dieses hypothetische System würde von dem tatsächlich zu implementierenden System stark abweichen, da hierbei nicht auf Software-as-a-Service Lösungen wie DynamoDB oder aber auch Textract zurückgegriffen werden könnte. Diese Entscheidung hebt zwar die Abhängigkeit von Amazon auf, erfordert jedoch die Bereitstellung von eigener Infrastruktur, die mit hohen Anschaffungskosten verbunden ist.

Um dies zu verhindern und sich dennoch nicht von einem Anbieter abhängig zu machen, könnte eine mögliche Bedingung die Verteilung der Cloud-Services auf verschiedene Anbieter sein. Hierdurch wird das Risiko eines Vendor Lock-Ins reduziert. Jedoch steigt bei

diesem Ansatz die Komplexität und der Aufwand, da mehrere Cloud-Anbieter eingebunden werden müssen.

Eine weitere Bedingung könnte die Nutzung eines Infrastructure as Code Framework wie Serverless sein, damit eine weitere Abstraktionsebene zwischen System und unterliegender Infrastruktur geschaffen wird.

4 Infrastruktur

In diesem Teil folgt die Vorstellung der einzelnen Software-Komponenten, die bei der Realisierung des Systems genutzt wurden, damit im weiteren Verlauf die Architektur erklärt werden kann.

4.1 Amazon Simple Email Service

Amazon Simple Email Service (SES) ist ein cloudbasierter Service, der einem das Empfangen und Verschicken von E-Mails ermöglicht. Hierbei können Nutzer eigene Domains benutzen. Da sich dafür entschieden wurde, das System, soweit möglich, in die Cloud zu verlagern, wird SES für die Annahme und Weiterleitung ankommender E-Mails, inklusive der zugehörigen Anhänge, benutzt. SES wird (Stand Februar 2020) nur in der AWS Region Ireland angeboten und verursacht dadurch eine höhere Latenz. Da der Anwendungsfall keine Real-Time-Verarbeitung erfordert, überwiegt die Verminderung der Komplexität durch Nutzung des Services. [6]

4.2 Amazon Simple Storage Service

Der Amazon Simple Storage Service (S3) bietet die Möglichkeit, Objekte abzuspeichern und sie sich bei Belieben wieder ausgeben zu lassen. Dies geschieht in sogenannten Buckets, welche von einer Menge von Objekten repräsentiert werden. Amazon wirbt mit einer Verfügbarkeit von über 99,99%. S3 Buckets bieten eine breite Vielfalt an Einstellungsmöglichkeiten, eine davon ist die Replikation des Inhalts eines Buckets, in einen anderen Bucket. Diese spielt für das System eine große Rolle, da Services nur auf Buckets ihrer eigenen Region zugreifen können. Die Replikation ermöglicht an dieser Stelle überregionalen Datentransfer. [8]

4.3 Amazon Simple Notification Service

Amazon Simple Notification Service (SNS) bietet dem Nutzer die Möglichkeit Event-Driven-Computing im Amazon-Ökosystem umzusetzen, durch die Bereitstellung einer Cloud-Infrastruktur. SNS ermöglicht es, sogenannte Subscriber oder Abonnenten zu benachrichtigen, woraufhin diese auf die Nachricht reagieren können. Sogenannte Publisher können Events zu einem bestimmten Thema auslösen und alle Abonnenten des Themas werden benachrichtigt. Viele der Amazon Web Services bieten eine native Schnittstelle zu SNS an. [7]

4.4 Amazon DynamoDB

Amazon DynamoDB ist ein Key-Value Speichersystem, welches dem Nutzer darüber hinweg erlaubt, komplexere Werte abzuspeichern, bis hin zu Dokumenten. DynamoDB zeichnet sich laut Amazon durch Skalierbarkeit, Sicherheit sowie einer Zugriffszeit von unter 300 Millisekunden bei 99,9 Prozent aller Anfragen aus [29].

DynamoDB hält Daten in Tabellen, welche über einen Namen referenziert werden. Einer Tabelle werden bei der Erstellung von DynamoDB genügend Partitionen zugewiesen. Wenn der Nutzer eine höhere Durchsatzrate einstellt oder der belegte Speicherplatz auf den einzelnen Partitionen eine bestimmte Schwelle überschreitet, werden zusätzliche Partitionen zugewiesen. Partitionen werden auf sogenannten Storage Nodes gespeichert. Jede Tabelle enthält entweder keine oder mehr Elemente, welche jeweils aus einer Gruppe von Attributen bestehen. Ein Attribut ist das basale Datenelement DynamoDBs. Die meisten Attribute sind skalar, sie können aber auch komplexere Datenstrukturen wie Listen oder Maps enthalten. Solche verschachtelten Attribute können bis zu 32 Ebenen tief gehen. Jedes Element ist eindeutig identifizierbar, was durch einen eindeutigen Primärschlüssel ermöglicht wird. Dieser muss, neben dem Tabellennamen, bei der Erstellung einer Tabelle festgelegt werden. Primärschlüssel können ausschließlich Zeichenfolgen, Zahlen oder Binärwerte annehmen [3]. DynamoDB bietet dem Nutzer zwei verschiedene Arten von Primärschlüsseln zur Benutzung an:

1. **Partitionsschlüssel:** Ein Partitionsschlüssel ist ein einfacher Primärschlüssel, bestehend aus einem einzigen Attribut. Dieses muss für jedes Element eindeutig sein.

Intern nutzt DynamoDB dieses Attribut als Input für eine Hash-Funktion, deren Output bestimmt, auf welcher Partition das Element gespeichert wird [3].

2. **Zusammengesetzte Primärschlüssel:** Ein zusammengesetzter Primärschlüssel besteht aus zwei Attributen, einem Partitionsschlüssel und einem Sortierschlüssel. Auch hierbei werden die Partitionsschlüssel gehasht, und basierend darauf einer Partition zugeordnet. Es ist möglich, dass mehrere Elemente den gleichen Partitionsschlüssel haben, solange die Sortierschlüssel sich unterscheiden. Elemente, die den gleichen Partitionsschlüssel vorweisen, werden zusammen abgespeichert und nach den Sortierschlüsseln geordnet [3].

DynamoDB bietet dem Nutzer grundlegende Anfragen wie GetItem, PutItem, UpdateItem und DeleteItem, aber darüber hinaus auch SCAN, um alle Elemente einer Tabelle abzurufen und gegebenenfalls zu filtern, und QUERY um alle Elemente eines bestimmten Partitionsschlüssels zu erhalten. Jede Anfrage an DynamoDB, die über die öffentliche API geschieht, wird zuerst von einem sogenannten Request Router verarbeitet. Dieser ist ein 'stateless' Service, der die Anfrage authentifiziert und überprüft, ob der Nutzer autorisiert ist, diese Anfrage zu stellen. Die Request Router sind nicht nur ausschließlich für DynamoDB konzipiert, sondern sie kommen bei jedem AWS Service zum Einsatz. Im DynamoDB Kontext sind sie außerdem für das Routing der Nutzeranfragen zuständig, was im Folgenden anhand von Put- und Get-Anfragen näher erläutert wird [24]:

- **PutItem:**Möchte der Nutzer ein Element mit einer PutItem-Anfrage in einer Tabelle speichern, muss zuerst die Partition gefunden werden, die für den Primärschlüssel des Elements zuständig ist. Dafür wendet sich der Request Router mit dem Primärschlüssel an das sogenannte Partition Metadata System. Dieses gibt den Storage Node, auf welchem sich die Partition befindet, zurück. Dieser Storage Node wird Leader genannt. Nun wird auf dem Leader das Element gespeichert. Daraufhin propagiert der Leader die Änderung an zwei weitere Storage Nodes, da jede Partition auf drei verschiedenen Storage Nodes redundant gespeichert wird. Der Leader wartet dann darauf, dass einer der beiden anderen Nodes die Änderung bestätigt. Dann erst antwortet er dem User, dass die PutItem-Anfrage erfolgreich war. Falls der Leader ausfällt, wird ein neuer, anhand des von Leslie Lamport entwickelten Paxos-Algorithmus gewählt [24].
- **GetItem:** Bei einer GetItem-Anfrage gibt der Nutzer einen Primärschlüssel an und erhält das zugehörige Element. Sobald der Request Router die Anfrage erhält,

wendet er sich an das Partition Metadata System, welches standardmäßig einen der drei die Partition enthaltenden Storage Nodes zurückgibt. Da dies nicht unbedingt der Leader sein muss, bedeutet dies, dass GetItem Anfragen ohne Konfiguration des Nutzers 'Eventually Consistent' sind und veraltete Daten zurückgeben werden können [24]. Der Nutzer kann für Lesevorgänge durch Parametrisierung festlegen, dass diese Strongly Consistent sind, was jedoch zu eine höheren Latenz führen kann [3].

DynamoDB unterstützt Transaktionen, welche die von Haerder und Reuters vorgestellten ACID-Eigenschaften erfüllen [31]. Hierbei werden mehrere Anfragen zu einer Transaktion aggregiert. Sobald eine der Anfragen fehlschlägt, wird die ganze Transaktion abgebrochen und jede vorausgegangene Änderung wird rückgängig gemacht. Weitere Features, welche DynamoDB dem Nutzer zur Verfügung stellt, sind On-Demand Backups, Zweitschlüssel, Point-in-Time Recovery, Caching für schnellere Zugriffszeiten und sogenannte Global Tables, welche Replikation über mehrere AWS-Regionen ermöglichen.

4.5 Amazon CodePipeline

Amazon CodePipeline ist ein Continous Delivery Service, der das Erstellen von sogenannten Pipelines erlaubt. Diese ermöglichen eine schnelle und sichere Veröffentlichung und Freigabe von Software. Der Nutzer kann die erforderlichen Schritte für eine Veröffentlichung nach seinem Belieben konfigurieren, welche dann von CodePipeline automatisiert werden. Die möglichen Schritte sind in verschiedene Aktionstypen kategorisiert. Ein Beispiel sind Quellenaktionstypen, welche es dem Nutzer für eine automatisierte Bereitstellung des Codes ermöglicht, verschiedene Versionierungssysteme an die Pipeline anzuschließen. Weitere Aktionstypen sind unter anderem automatisierte Build-, Test- und Deploymentprozesse. Für die einzelnen Aktionstypen können außer internen Amazon Services auch bestimmte externe Services eingebunden werden, wie zum Beispiel GitHub oder Jenkins. [2]

4.6 Amazon Cloudformation

Amazon Cloudformation erlaubt dem Nutzer die Umsetzung von Infrastructure as Code innerhalb des AWS Ökosystems. Somit können Entwicklerteams zusammen an einer,

nach Bedürfnissen versionierten, Datei arbeiten, welche die Infrastruktur definiert. Dies ermöglicht eine bessere Nachvollziehbarkeit, sowie - durch die mögliche Wiederverwendung des Codes - eine schnelle Einrichtung der Infrastruktur. Im folgenden Kapitel wird konkreter auf die Vorzüge von Infrastructure as Code eingegangen. [1]

4.6.1 AWS Serverless Application Model

Das AWS Serverless Application Model (SAM) ist ein Open-Source Framework und eine Erweiterung von Cloudformation. Mittels SAM können innerhalb des AWS Ökosystems, serverlose Anwendungen mit Infrastructure as Code entwickelt werden. Der Nutzer definiert seine Ressourcen in einer YAML Datei, welche von SAM in Cloudformation Syntax übersetzt wird. Außerdem ermöglicht SAM dem Nutzer, durch Bereitstellung einer Ausführungsumgebung, Lambda-Funktionen lokal zu testen. [5]

4.7 Amazon Lambda

Amazon Lambda ist ein Service, der Serverless Computing ermöglicht, das Ausführen von Code, ohne sich um Server-Infrastruktur kümmern zu müssen. Lambda fällt unter die Kategorie der Function as a Service (FaaS) Services. Hierbei muss der Nutzer dem Service-Provider lediglich den Programmcode zur Verfügung stellen, einen Entrypoint, also eine Funktion, die von dem Service aufgerufen wird, definieren und gegebenenfalls die Laufzeitumgebung spezifizieren. Die Allokation der Ressourcen für die Nutzung übernimmt der Provider. Dieser Prozess findet in sogenannten Containern statt, welche im Hintergrund in der Amazon Elastic Compute Cloud laufen.

Ein Lambda-Service kann auf zwei verschiedene Weisen aufgerufen werden:

1. **Push:** In diesem Fall wird die Funktion aufgerufen, sobald ein vorher spezifiziertes Event im AWS-Ökosystem auftritt, zum Beispiel eine SNS-Benachrichtigung oder auf den Anwendungsfall bezogen, sobald einem S3-Bucket ein Objekt hinzugefügt wird.

2. **Pull:** Lambda kann auch eine Datenquelle abfragen, zum Beispiel eine Queue oder einen Stream. Hierbei fragt Lambda die Datenquelle periodisch ab und verarbeitet dann etwaige Daten. Wichtig zu wissen ist, dass hier keine Realtimeverarbeitung vorliegt.

Weiterhin kann eine Lambda-Funktion mit zwei unterschiedlichen 'Invocation-Types' aufgerufen werden:

1. **RequestResponse:** Wenn man eine Lambda Funktion mit dem Typ Request-Response aufruft, wird diese synchron ausgeführt, dass heißt, dass AWS auf eine Antwort der Lambda-Funktion wartet. Ein Beispiel hierfür wäre der Aufruf über ein API-Gateway oder aber auch über das AWS Command Line Interface. Hierbei wird der Funktion die HTTP-Request übergeben und gewartet bis eine Response zurückkommt. Die Behandlung eines Runtime-Errors liegt in der Verantwortung desjenigen, der die Funktion aufgerufen hat.

2. **Event:** In vielen Fällen erwartet derjenige, der die Funktion aufruft, keine Antwort. Für solche Fälle bietet Lambda einen asynchronen Ausführungstyp, nämlich Event. Wenn eine Funktion bei einem asynchronen Aufruf einen Error zurück gibt, wird der Aufruf, je nach Konfiguration, gar nicht, ein- oder zweimal wiederholt.

Die Ausführung einer Lambda-Funktion benötigt den Source-Code, welcher als ZIP Datei in einem S3 Bucket liegt und ein Runtime-Environment. Bei der erstmaligen Ausführung wird der Source-Code heruntergeladen, das Environment initialisiert und die vorher definierte Handler-Funktion aufgerufen. Dies wird als ein 'Cold Start' bezeichnet. Dies kann bei Sprachen wie zum Beispiel Java aufgrund von Overhead (JVM) zu höheren Latenzen führen als bei anderen.

Da dies bei einer hohen Anzahl von Funktionsaufrufen nicht performant wäre, hält Lambda die Funktionen für eine unbestimmte Dauer "warm". Dabei werden die Ressourcen wie Arbeitsspeicher nicht neu allokiert, wodurch die Laufzeitumgebung bestehen bleibt. Somit muss bei einem erneuten Funktionsaufruf nur noch der Code ausgeführt werden, was zu einer immensen Optimierung der Performanz und der durch die Laufzeit bedingten Kosten führt. Eine solche Ausführung wird "Warm Start"genannt.

Wie lange ein Container warm gehalten wird, kann der Nutzer weder beeinflussen noch einsehen. Dies führt dazu, dass man sich als Nutzer des Services darüber im Klaren sein sollte, dass manche Lambda-Aufrufe gegebenenfalls deutlich länger brauchen als andere.

Es kann durch empirische Beobachtung der Frequenz eingehender, das heißt zu verarbeitender E-Mails davon ausgegangen werden, dass die Lambda-Funktion im Durchschnitt einmal alle zehn Minuten aufgerufen wird, wodurch aller Voraussicht nach die meisten

Ausführungen 'cold' sein werden. Deswegen ist es in Hinblick auf die Kosten sinnvoll, die Laufzeit und den Overhead bei der Erzeugung der Laufzeitumgebung möglichst gering zu halten. Aus diesem Grund wurde sich bei der Realisierung des Systems für Node.js entschieden, welches durch eine schnelle Initialisierung der Laufzeitumgebung und Techniken wie Non-Blocking I/O in Hinblick auf den Anwendungsfall - im Gegensatz zu Alternativen wie Java - große Vorteile bietet [4].

4.8 Amazon Textract

Amazon Textract ist Amazons Implementation für Text- und Datenerkennung aus eingescannten Dokumenten. Im Gegensatz zu anderen OCR Programmen stellt Textract dem Nutzer Features zur Verfügung, welche über die einfache Texterkennung hinausgehen, wie zum Beispiel der Erkennung von Formularen und Tabellen. Ein weiterer Unterschied ist, dass Textract keine Konfiguration oder Parameter erwartet, was durch den Einsatz von Machine Learning ermöglicht wird.

Textract bietet zwei verschiedene Operationen auf Textdokumenten an, Texterkennung und Textanalyse. Die Anfrage an den Service muss ein 'Dokument' enthalten, welches entweder S3 Objekte oder in Base-64 kodierte Bytes sein müssen. Textract gibt dem Nutzer als Response ein Array über 'Block' Objekte wieder. Jeder Block kann einen von mehreren Typen, wie Page, Line oder Word annehmen. Außerdem kann jeder Block Relationships haben, welche Referenzen zu Child Blöcken enthalten, wodurch die Response eine hierarchische Struktur annimmt. An der Spitze dieser Hierarchie stehen die Page Blöcke, welche jeweils eine Seite aus dem Dokument repräsentieren. Die Pages können Line-, Table- oder Key-Value Set Blöcke als Kinder haben, wobei Table und Key-Value Set Blöcke nur bei der Textanalyse vorkommen können und nicht bei der Texterkennung. Diese Blöcke können auch Kinder haben, wodurch man auf die unterste Ebene der Hierarchie gelangt. Line Blöcke können Words, Table Blöcke Cells und Key-Value Set Blöcke Keys und Values enthalten. Jeder Block enthält den Text der von Textract erkannt wurde, ein Geometry Object, welches die Position des Blockes auf der jeweiligen Seite wiedergibt und einen Confidence Score, der eine Zahl von null bis hundert annehmen kann. Dieser Confidence Score repräsentiert, wie sicher Textract bei der Erkennung des Textes ist [9].

Beide Operationen können entweder synchron, und zwar für Anwendungsfälle in denen Single-Page Dokumente möglichst schnell verarbeitet werden sollen, oder asynchron ver-

arbeitet werden. Sobald das zu verarbeitende Dokument mehr als eine Seite enthält, muss die Verarbeitung asynchron ausgeführt werden. Bei der synchronen Ausführung kann der Nutzer entweder die DetectDocumentText API, für Texterkennung, oder die AnalyzeDocument API für Textanalyse anfragen, und bekommt die Response unmittelbar zurück. Bei einer asynchronen Anfrage ist dies nicht der Fall, da die Verarbeitung, abhängig von der Größe des Dokuments und des Anfragentyps, länger brauchen kann. Da dies für den Anwendungsfall relevant ist, werde ich im Folgenden näher auf den Umgang mit Mutli-Page Dokumenten eingehen.

Für asynchrone Anfragen stehen die StartDocumentTextDetection API für Texterkennung und die StartDocumentAnalysis für Textanalyse zur Verfügung. In diesem Fall muss neben einem S3 Object auch ein SNS Topic mitgeben werden, welches sich in der gleichen Region wie der Textract Endpoint befinden muss. Daraufhin wird dem Nutzer eine JobId zurückgegeben und die Verarbeitung des Dokuments gestartet. Sobald das Dokument fertig verarbeitet ist, schickt es eine Meldung an das SNS Topic, welches die Subscriber benachrichtigt. Dieses Event enthält einen Status, der Auskunft darüber gibt, ob die Verarbeitung erfolgreich war oder nicht. Schließlich kann der Nutzer, falls die Verarbeitung erfolgreich war und in Abhängigkeit vom vorausgegangenen Operationstyp, GetDocumentTextDetection oder GetDocumentAnalysis mit der JobId als Parameter aufrufen, die ihm ein Array von Blöcken zurückgeben.

Abbildung 4.1 zeigt eine Beispielarchitektur für den Umgang von asynchronen Textract-Aufrufen. In diesem Fall werden die Events in eine AWS Simple Queue Service Queue gepusht, die überwacht wird. Die eintreffenden Events werden nun auf ihren Status hin untersucht. Falls die Analyse erfolgreich war, wird die Get-API aufgerufen und die Ausführung endet. Wenn die Verarbeitung fehlschlägt, endet die Ausführung direkt.

Textract ist zu diesem Zeitpunkt nur in den europäischen AWS-Regionen Irland und London verfügbar. Die Kosten bei reiner OCR belaufen sich auf 1,5 US-Dollar pro 1000 Seiten PDF. Bei der Analyse inklusive Tabellen und Formular-Erkennung steigen die Kosten auf 65 Dollar pro 1000 Seiten an.

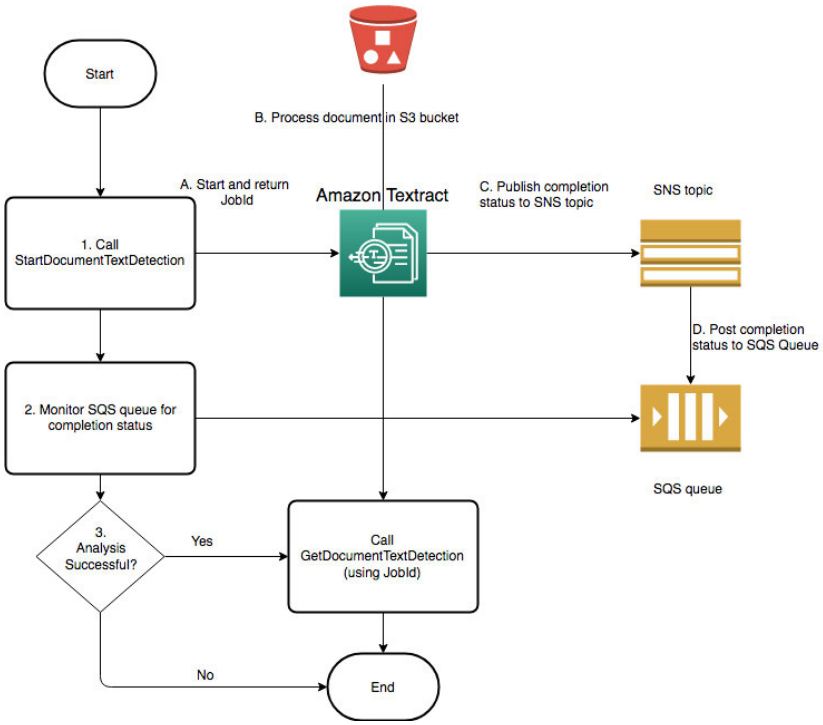


Abbildung 4.1: Beispiel Architektur für asynchrone Verarbeitung mit Textract [12]

4.9 Tesseract

Tesseract ist eine, bei HP entwickelte open-source Software für die optische Zeichenerkennung. Der Ursprung Tesseracts liegt in einem Promotionsforschungsprojekt des inzwischen bei Google angestellten Ray Smith [39]. Ursprünglich als mögliches Add-on für Scanner gedacht, wurde es 2005 von HP als open-source veröffentlicht und wird bis zum heutigen Stand von Google weiterentwickelt. Im Folgenden wird die Funktionsweise von Tesseract näher erläutert. Hierbei wird sich auf das 2007 von Ray Smith veröffentlichte Paper "An Overview of the Tesseract OCR Engine" bezogen. Etwaige Änderungen bezüglich des Aufbaus Tesseract werden nicht miteinbezogen.

Der erste Schritt in der Texterkennung von Tesseract, ist die physische Seitenlayoutanalyse. Hierbei werden die einzelnen Seiten in Text- und Nicht-Textbereiche eingeteilt. Methoden für die Seitenlayoutanalyse kann man grob in zwei Kategorien einordnen:

- **Top-Down:**
  Top-Down Methoden arbeiten rekursiv und unterteilen die Seiten in horizontale und vertikale Richtungen anhand von Leerraum, welcher als Grenze zwischen Absätzen und Spalten interpretiert wird. Dies birgt den Nachteil, dass nichtrechteckige Bereiche und andere Sonderfälle unvollständig oder gar nicht erkannt werden [42].

- **Bottom-Up:**
  Bei dem Bottom-Up-Ansatz bilden kleine Bereiche, wie Pixel oder Gruppen von Pixeln, denen durch den Einsatz eines Klassifikationsverfahren eine Ähnlichkeit zu geschrieben wird, einen größeren Bereich. Dadurch können auch Bereiche mit beliebiger Form erkannt werden. Dies kann dazu führen, dass übergeordnete Bereiche fragmentiert, und dadurch inkorrekt erkannt werden [42].

Tesseract wendet für die Seitenlayoutanalyse einen Hybrid aus Top-Down und Bottom-Up an. Zuerst werden Tabulatorstopps, die bei der Formatierung der Seite benutzt wurden, mit Hilfe eines Bottom-Up Ansatzes identifiziert. Dadurch wird das ursprüngliche Layout der Seite deduziert, welches dann mit Hilfe eines Top-Down Ansatzes auf die Seite angewendet wird, wodurch der Seite eine Struktur auferlegt wird [42].

Im nächsten Schritt wird eine Connected-Component-Analysis durchgeführt, bei welcher die Umrisse der als zugehörig identifizierten Bereiche gelagert werden. Dies war zur Zeit der Entwicklung von Tesseract eine rechenintensive Design-Entscheidung, die den Vorteil barg, dem System die Erkennung von weißer Schrift auf schwarzen Hintergrund zu ermöglichen. Am Ende dieses Schrittes befinden sich die Umrisse in Blobs [41].

Daraufhin werden die Blobs in Textzeilen gegliedert. Hierfür wird zuerst eine Untermenge an Blobs herausgefiltert, welche den Haupttext repräsentieren könnten. Dies wird dadurch erzielt, dass alle Blobs deren Höhe unter dem Zwanzig-Prozent-Perzentil sind, aussortiert werden. Die gefilterten Blobs werden anhand ihrer Platzierung auf der Seite sortiert. Schließlich wird durch die sortierten Blobs iteriert, aus denen entweder neue Textzeilen erstellt oder die zu bestehenden Textzeilen hinzugefügt werden [40].

Text kann in einer proportionalen oder nichtproportionalen Schriftart verfasst sein. Bei einer nichtproportionalen Schriftart haben Schriftzeichen eine feste Zeichenbreite, wohingegen sich die Breite bei Schriftzeichen einer proportionalen Schriftart an die tatsächliche Breite, die das Zeichen auf Grund seiner Eigenschaften benötigt, anpasst [41]. Deswegen untersucht Tesseract im nächsten Schritt die Textzeilen zuerst auf nichtproportionale Schrift, genannt Fixed Pitch Detection. Sobald nichtproportionale Schrift gefunden wird,

können die Wörter anhand der erkannten Breite direkt in Textzeichen zerteilt werden. Dies ist bei proportionaler Schrift deutlich anspruchsvoller. Hierbei müssen zuerst etwaige verbundene Zeichen erkannt und separiert werden, was insbesondere bei kursiven Schriften, wie zum Beispiel Schreibschrift, zu Unsicherheiten in der Zeichenerkennung führen kann. Tesseract sucht hierbei nach Stellen, an welchen Buchstaben unterteilt werden könnten und führt die Teilung in der Reihenfolge der von Tesseract berechneten Wahrscheinlichkeit, dass es sich um einen korrekten Schnitt handelt, aus. Für jedes Wort rechnet Tesseract ein sogenanntes Confidence-Level aus und weist es ihm zu. Diese Kennzahl repräsentiert, wie sicher das Programm bei der Erkennung des Wortes ist. Die Teilung von Buchstaben wird nur ausgeführt, sofern sich das Confidence-Level des Wortes, welches die Buchstaben enthält, erhöht [41].

Falls das Wort - nach der Ausführung aller dem Confidence-Level als förderlich betrachteten Teilungen - immer noch nicht einer von Tesseract angesetzten Mindest-Confidence entspricht, wird es dem sogenannten Associator übergeben. Dieser sucht mit dem A* Algorithmus die Menge der Möglichkeiten von maximal geteilten Versionen eines Zeichens durch. Diese 'kaputten' Zeichen werden dann ausgewertet. In der folgenden Abbildung ist ein Wort aus fehlerhaften Zeichen zu sehen, welches von Tesseract ohne Probleme erkannt wird [41].

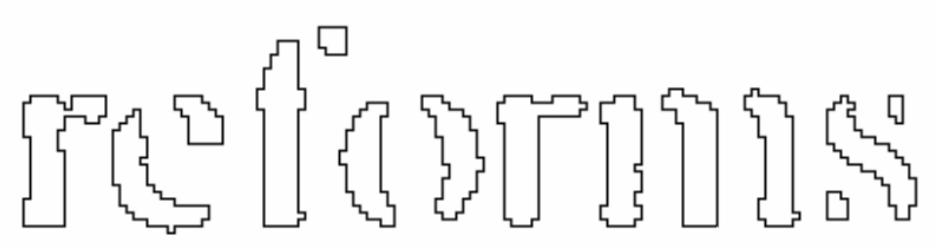


Abbildung 4.2: Beispielwort mit fehlerhaften Zeichen [41]

Im letzten Schritt der Prozesses werden die Zeichen klassifiziert. Eine Besonderheit des Klassifikators ist die verhältnismäßig geringe Anzahl der Trainingsdaten. Die Stichprobe umfasst gerade einmal 60160 Samples, was im Gegensatz zu anderen Klassifikatoren, wie zum Beispiel Baird's 100-Font Klassifikator, mit einer Stichprobengröße von über einer Million Samples [25] sehr gering ist.

Darüber hinaus verfügt Tesseract auch über die Möglichkeit einer linguistischen Analyse. Sobald eine Zeichenkette als ein zusammengehöriges Wort erkannt wird, sucht das Linguistik-Modul in verschiedenen Kategorien, wie 'Top frequent word' oder 'Top dictio-

nary word' nach einem passendem Wort. Dann wird mit einer Distanzmetrik das Wort mit der größten Übereinstimmung gesucht und bestimmt [41].

Neuere Versionen von Tesseract verfügen inzwischen über eine auf einem neuronalen Netz basierende optische Zeichenerkennung. Zu diesem Zeitpunkt ist diese Implementation jedoch noch nicht dokumentiert, weshalb nicht näher auf die Funktionsweise eingegangen werden kann.

5 Entwurf

In diesem Abschnitt werden die Entwurfsentscheidungen, die bei der Konzeption des Systems in Hinblick auf die Anforderungen getroffen wurden, näher dargestellt.

5.1 Systemarchitektur

Um die Qualitätsanforderungem hinsichtlich der Änderbarkeit und der Übertragbarkeit zu erfüllen, wurde das System in einer Clean Architecture geschrieben. Dies ist im Hinblick auf den Anwendungsfall vorteilhaft, da zwei verschiedene OCR-Schnittstellen angebunden und getestet werden sollen.

5.1.1 Clean Architecture

Durch die Einteilung von Klassen in verschiedene Komponenten, werden überflüssige Abhängigkeiten beseitigt und der Austausch von einzelnen Komponenten vereinfacht. Laut Robert C. Martin sind Softwaresysteme, welche folgende Charakteristika aufweisen, in Clean Architecture umgesetzt worden [35]:

1. **Framework-unabhängig:**
   Der Architektur ist es möglich, Frameworks und Libraries einzubinden, sie wird jedoch im Falle einer Implementierung nicht dazu gezwungen, sich den Gegebenheiten eines konkreten Frameworks anzupassen. Es darf keine Abhängigkeit gegenüber einem Framework existieren.

2. **Testfähig:**
   Der Domänenbereich soll ohne jedwede externen Elemente getestet werden können. Um einen Use-Case zu testen, darf keine Datenbank vorausgesetzt werden.

3. **UI-unabhängig:**
   Für die Architektur darf kein Unterschied darin bestehen, ob der Nutzer die Anwendung über eine Web-UI oder über die Konsole bedient. Änderung an der UI dürfen keine Änderung an den Geschäftsregeln nach sich ziehen.

4. **Datenbankunabhängig:**
   Die Geschäftsregeln dürfen nicht in Abhängigkeit zu der konkreten Implementierung der Datenbank stehen. Ob im Hintergrund ein SQL-Server oder eine DynamoDB läuft, muss für den Use-Case unerheblich sein.

5. **Unabhängig von jeglichen sonstigen externen Komponenten:**
   Generell dürfen Geschäftsregeln keinerlei Kenntnis von den konkreten Implementationen haben. Sie nutzen nur Interfaces.

Daraus resultiert, nach Martin die folgende Darstellung:

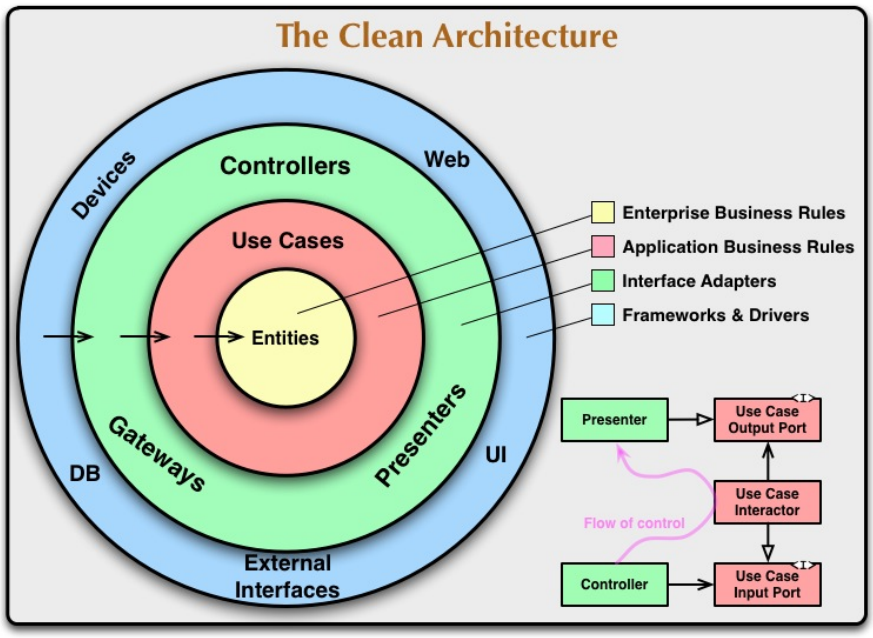


Abbildung 5.1: Die saubere Architektur [35]

Die schwarzen Pfeile spielen hierbei eine zentrale Rolle, da sie die erlaubten Abhängigkeiten angeben. Die Entitäten dürfen niemals von der Existenz der Use-Cases, geschweige denn der Infrastruktur wissen. Umgekehrt gilt dies nicht, Controller dürfen mit Entitäten arbeiten.

5.1.2 Architekturentwurf

Im folgenden Abschnitt wird auf den Entwurf der Systemarchitektur eingegangen. Um dem Paradigma der Clean Architcture zu folgen, wurde der Code in drei verschiedene Schichten unterteilt: Die Domainschicht, die über Entitäten und entitätspezifische Services verfügen soll, die Applikationsschicht, welche die anwendungsspezifischen Use Cases und Interfaces für Services und Repositories beinhalten soll und die Infrastrukturschicht, die die konkreten Datenbankrepositories und anwendungsunabhängigen Services enthalten soll. Im Folgenden werden die einzelnen Schichten und deren zugehörige Klassen und Interfaces näher erläutert:

Applikationsschicht:

Die Applikationsschicht soll die Use-Cases und Interfaces für Services und Repositories enthalten. In den Use-Cases befinden sich die applikationsspezifischen Geschäftsregeln. Die Use-Cases arbeiten mit den Interfaces der Services und Repositories, so dass es für einen Use-Case später keine Rolle spielt, ob hinter dem Interface eine DynamoDB oder eine MySQL-Anbindung steckt. Die Repositories und Services werden im Konstruktor der Use-Cases übergeben. Im Folgenden werden die zwei Use-Cases, welche implementiert werden sollen, näher erläutert:

- **ParseEmail:** Der Use-Case ParseEmail soll dafür zuständig sein, eine E-Mail mit etwaigen Anhängen zu verarbeiten und die Ergebnisse abzuspeichern. Dieser Use-Case soll die funktionalen Anforderungen 1 - 5 erfüllen.
- **AnalyzePerformance:** Der Use-Case AnalyzePerformance soll den Abgleich von per Hand eingegebenen Daten mit automatisch verarbeiteten Daten abbilden. Er soll jeweils einen Bericht für jede verarbeitete E-Mail erstellen und dann eine Übersicht der Gesamtperformance updaten. Hierdurch soll die 6. Funktionale Anforderung erfüllt werden.

Auf die Interfaces für Services und Repositories wird später in der Infrastruktur näher eingegangen (Siehe Abbildung 5.3).

Domainschicht:

Die Domainschicht soll Entitäten und Services enthalten, welche ausschließlich Geschäftslogik enthalten. Im Folgenden wird auf die Entitäten eingegangen:

- **Claim:** Der Claim ist die Core-Entität des Systems. Claim-Objekte werden im ParseEmail Use-Case erstellt und abgespeichert. Ein Claim hält das Interface 'ClaimAttributes', welches alle Bestandteile eines Claims enthält. Claim und alle Objekte, die im Claim aggregiert werden, implementieren das Interface Comparable. Um die Erläuterung des Claims kurz zu halten, wird für eine Verdeutlichung der Hierarchie auf Abbildung 5.2 verwiesen.

- **Result:** Ein Result ist das unmittelbare Ergebnis eines Vergleichs von zwei Claims. Es hält die beiden Claims, eine Liste von Strings, welche die abweichenden Variablen listet, sowie die summierte Anzahl aller unterschiedlichen Variablen.

- **Evaluation:** Eine Evaluation repräsentiert die Durchschnittsperformance des Systems. Sobald ein neues Result erzeugt wird, soll die Evaluation aktualisiert werden. Sie hält die Fehlerrate der Ergebnisse, sprich die Anzahl von Variablen eines Claims im Verhältnis zur durchschnittlichen Anzahl von Fehlern. Darüber hinaus enthält eine Evaluation die historische Anzahl aller geparsten Claims und eine Map über alle Variablen des Claims als Key und die historische Anzahl von Fehlern spezifisch für diese Variable.

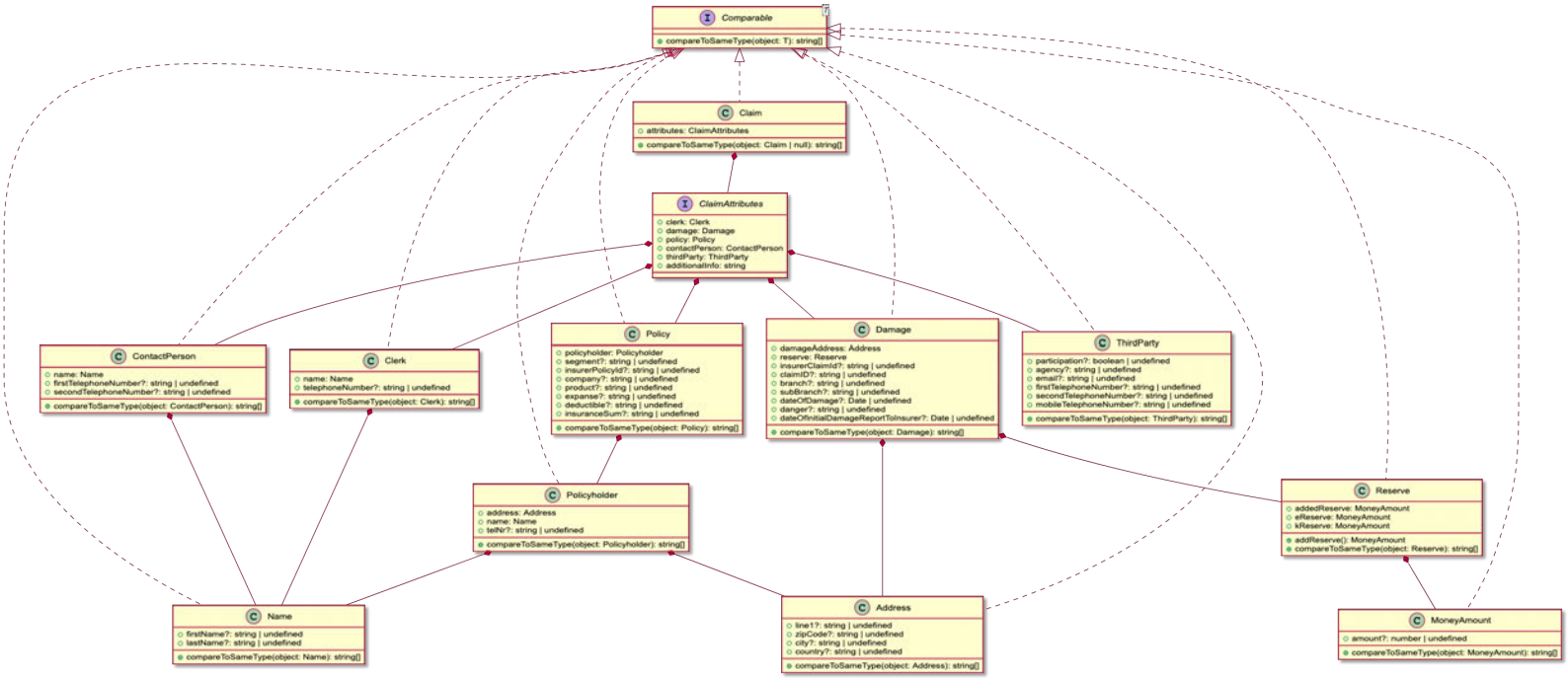


Abbildung 5.2: Core-Entitäten UML

Die Domainschicht enthält folgende Services:

- **FormatService:** Ein FormatService ist in diesem System ein Service, der aus einem konkret formatierten String ein Claim Objekt erzeugt. FormatService ist eine abstrakte Klasse von der alle konkreten FormatServices erben. FormatService enthält eine Methode 'getKeyword', die einen String an allen Leerzeichen teilt und den ersten Teil zurückgibt. Dies ist für die konkreten Format-Services später relevant. Außerdem hat FormatService die abstrakte Methode 'getClaim', welche einen String entgegen nimmt und ein Claim zurückgibt. Diese muss von allen Klassen, die von FormatService erben, implementiert werden.
- **FormatServiceFactory:** FormatServiceFactory soll eine statische Factory-Methode, nämlich 'createFormatService', zur Erzeugung von Format-Services anbieten. Der Methode wird ein String und den Betreff der E-Mail übergeben, woraufhin die Methode ein Objekt zurück gibt, das von FormatService erbt. Der String bestimmt den konkreten Typ dieses Objekts.
- **NessyFormatService:** Der NessyFormatService erbt von FormatService und ist für die Erzeugung von Claims aus Strings im 'Nessy'-Format zuständig.
- **GuidewireFormatService:** Der NessyFormatService erbt von FormatService und ist für die Erzeugung von Claims aus Strings im 'Guidewire'-Format zuständig.
- **EvaluationService:** Der EvaluationService enthält eine statische Methode, nämlich 'evaluateClaims'. Diese nimmt eine Evaluation und ein Array über Strings an und gibt eine Evaluation zurück. In der Methode wird die Input-Evaluation ausgelesen und auf dem Array basierend aktualisiert. Zurückgegeben wird die aktualisierte Version der Evaluation.

Darüber hinaus soll die Domainschicht die sogenannten 'Extractor-Services' beinhalten. Für alle Objekte, die in Claim enthalten sind, wird ein Extractor implementiert. Diese Extractor-Services werden von den Format-Services eingebunden und erstellen aus einem bestimmten Bereich der E-Mail ihr respektives Objekt. Der PolicyholderExtractor ist verantwortlich für die Erzeugung des PolicyHolders et cetera. Mit den erstellten Objekten der Extractor-Services sollen dann in den Format-Services Claims erzeugt werden.

Infrastrukturschicht:

Die Infrastrukturschicht wird Service- und Repositoryimplementationen, die Interfaces aus der Applikationsschicht implementieren, enthalten. Durch diese Abstraktion können konkrete Repositories oder Services ausgetauscht werden, ohne die Use-Cases oder Domänenlogik zu verändern. Folgende Services sind in der Infrastruktur enthalten:

- **GraphQLCapitolService:** Dieser Service soll die Anbindung über GraphQL an das bestehende System, in dem die Schadenanlage per Hand geschieht, ermöglichen. Der Service stellt eine Methode für den Abruf von Schadensfällen an.
- **CompareService:** Der CompareService soll eine bestimmte Equals-Methode anbieten, die auf Ungleichheit testet. Das Besondere an dieser Equals-Methode soll sein, dass sie null und undefined gleich behandelt.
- **DateFormatService:** Der DateFormatService soll die Formatierung von Strings eines Datumformates in ein anderes Format ermöglichen.
- **PhoneNumberParser:** Der PhoneNumberParser soll dem Nutzer eine Methode zum Parsen von deutschen Telefonnummern zur Verfügung stellen. Sonderzeichen sollen entfernt, und die vorangestellte '0' wird durch '+49' ersetzt werden.
- **S3FileHandle:** Dieser Service soll das Interface FileHandle aus der Applikationsschicht implementieren. Die FileHandles sind dafür gedacht, eine Abstraktionsebene zwischen FileRepositories und dem darunterliegenden Dateisystem zu erstellen. Konkrete FileRepositories kriegen ihre respektiven FileHandler über den Konstruktor injected. Dadurch soll verhindert werden, dass das Interface FileRepository Kenntnis darüber hat, wie auf die konkreten Dateisysteme zugegriffen wird.
- **PDFConverterService:** Der PDFConverService soll zwei statische Methoden anbieten, welche beide das Package 'imagemagick' nutzen. Die erste, 'convertPDFToTIFF', soll die zu verarbeitenden PDF-Dateien ins TIFF-Format konvertieren, damit Tesseract damit arbeiten kann. Da imagemagick bei der Konvertierung für jede PDF-Seite eine einzelne TIFF-Datei erstellt, soll die zweite Methode, 'bundleTIFFs', die ganzen TIFF-Dateien zu einer einzigen, großen Datei zusammenfassen.
- **TesseractService:** Der TesseractService soll als Wrapper um die Tesseract Binary fungieren. Der Service soll ein PDF und eine Confidence-Untergrenze erhalten und

einen Array von Strings, den extrahierten Worten, zurückgeben. Die Confidence-Untergrenze sorgt dafür, dass nur Worte, deren Confidence bei der Erkennung von Tesseract als höher eingeschätzt werden, zurückgegeben werden.

- **TextractService:** Der TextractService funktioniert analog zum TesseractService, außer dass die Textract-API abgerufen wird. Aber auch hier wird ein Array von Strings zurückgegeben, die dem als Parameter mitgegebenem Confidence-Level entsprechen.
- **SimpleEmailParser:** Dieser Service wird das Interface EmailParser implementieren und wird zu einem simplen Wrapper für eine Klasse aus dem npm-Paket 'mailparser'.
- **TextService:** Der TextService soll verschiedene Hilfsmethoden für die Arbeit mit Texten anbieten, wie zum Beispiel die Segmentierung eines Textes anhand von Keywords.

Das System soll außerdem insgesamt vier verschiedene Repositories enthalten:

- **DynamoDBClaimRepository:** Dieses Repository soll die Möglichkeit Schadensfälle in einer DynamoDB anzulegen, oder per ID auszulesen, anbieten.
- **DynamoDBResultRepository:** Das DynamoDBResultRepository soll die Persistierung eines Results in einer Dynamo Datenbank ermöglichen.
- **DynamoDBEvaluationRepository:** Dieses Repository soll dem Nutzer ermöglichen, eine Evaluation in einer DynamoDB zu speichern, zu aktualisieren oder auszulesen.
- **S3FileRepository:**
  Das S3FileRepository soll die Möglichkeit bieten, Objekte aus einem S3 Bucket zu holen.

In der folgenden Abbildung wird die Hierarchie zwischen den abstrakten und konkreten Repositories anhand des DynamoDBClaimRepository noch einmal verdeutlicht:

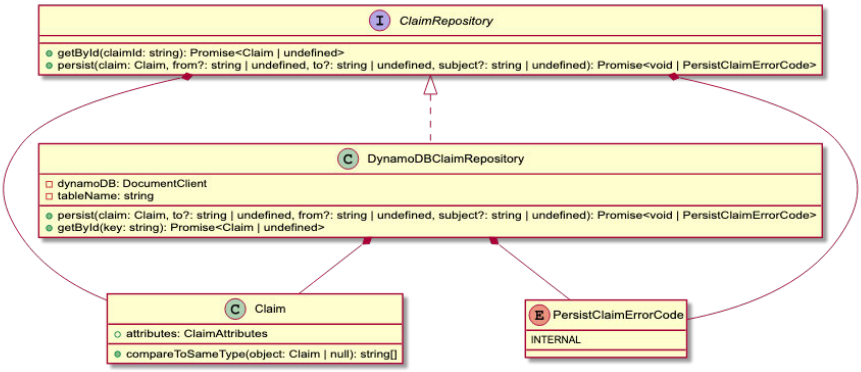


Abbildung 5.3: DynamoDBClaimRepository UML

5.1.3 AWS Architektur

Die Architektur der AWS-Services soll wie folgt realisiert werden.

Für beide Use-Cases soll es eine Lambda-Funktion geben. Jede E-Mail, die in einem bestimmten SES-Postfach ankommt, wird in einem S3-Bucket abgelegt. Da SES bis jetzt nur in Irland verfügbar ist, werden alle Objekte aus dem Bucket in einen anderen Bucket in Frankfurt repliziert. SNS wird ein Topic erstellen, welches den Bucket beobachtet. Sobald dem Frankfurter Bucket ein Objekt hinzugefügt wird, wird SNS eine Nachricht an alle Subscriber des Topics schicken. Die Lambda für den ParseEmail Use-Case hat das Topic abonniert und wird ausgeführt, sobald sie eine Nachricht erhält. Die Lambda holt sich aus dem Frankfurter S3-Bucket die E-Mail und verarbeitet sie. Bei etwaigen Anhängen wird entweder Textract oder Tesseract für die optische Zeichenerkennung benutzt. Das erstellte Claim-Objekt wird dann in einer DynamoDB-Tabelle, mit der Schadennummer als Primärschlüssel, gespeichert werden. Die Lambda für den Use-Case AnalyzePerformance wird auch ein SNS-Topic abonnieren, das außerhalb des zu entwickelnden Systems liegt. Dieses triggert die Lambda, sobald ein Claim manuell aus einer E-Mail erstellt wurde. Die Lambda holt sich per GraphQL den manuell erstellten Claim und extrahiert die Schadennummer. Mit der Schadennummer wird die Lambda dann den entsprechenden automatisiert verarbeiteten Claim aus der DynamoDB abrufen. Schließlich werden die Ergebnisse des Use-Cases in einer weiteren DynamoDB Tabelle abgespeichert. In der Abbildung 5.4 ist die geplante AWS-Architektur noch einmal als Diagramm dargestellt.

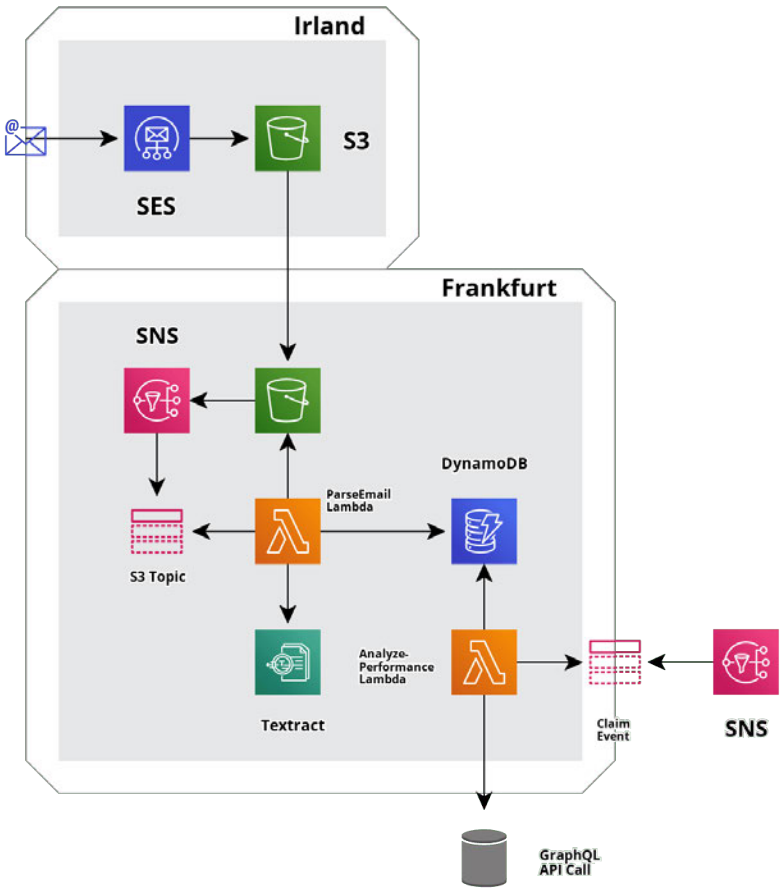


Abbildung 5.4: Architektur der AWS-Services

5.2 Infrastructure as Code

Damit Qualitätsanforderungen im Bereich der Übertragbarkeit und der Randbedingung 2 erfüllt werden können, ist es notwendig, bei der Umsetzung des Systems mit Infrastructure as Code (Infrastructure as Code (IaC)) zu arbeiten. IaC ist ein Ansatz für die automatische Einrichtung von Infrastruktur, der auf Praktiken aus der Softwareentwicklung basiert. Durch IaC wird es möglich, die gewünschte Infrastruktur in simplen Textdateien zu definieren und konfigurieren. Laut Kief Morris werden durch Infrastructure as Code folgende Praktiken möglich [38]:

Systeme können einfach reproduziert werden:
Durch IaC wird es möglich, einzelne Elemente einer Infrastruktur zu erstellen oder zu verändern. Änderungen der Laufzeitumgebungen oder Versionen von Software werden durch die Änderung einer einzelnen Zeile Code möglich. Dies reduziert Risiken im Falle von Änderungen an der Infrastruktur, da auftretende Fehler schnell zu behandeln sind.

'Wegwerfsysteme' werden ermöglicht:
Durch eine derart dynamische Infrastruktur können Ressourcen sehr einfach erstellt, aber auch wieder zerstört werden. Auch das schnelle Duplizieren von Infrastruktur ist möglich, was das Erstellen von Testumgebungen vereinfacht. Um dies ausnutzen zu können, sollen Softwaresysteme so unabhängig wie möglich von der Infrastruktur designt werden.

Systeme sind konsistent:
Wenn ähnliche Systeme innerhalb eines Projektes, wie zum Beispiel Server, inkonsistent sind, kann sich der Nutzer bei der Entwicklung von Software nicht auf seine Erfahrung verlassen. Wenn ein Server 32 Gigabyte Arbeitsspeicher hat und ein anderer nur 16, ermutigt dies die Entwicklung von Code, der speziell auf die unterliegende Infrastruktur abgestimmt ist. Um dies zu verhindern, sollten innerhalb eines Projektes oder einer Firma Teile der Infrastruktur standardisiert werden. Für das Erstellen von Datenbanken oder Server-Instanzen können Template-Dateien zur Verfügung gestellt werden, die somit die Infrastruktur vereinheitlichen.

Prozesse können wiederholt werden:
Systemadministratoren können wiederkehrende Prozesse, wie das Einrichten von Software oder anderen Tools, durch Skripte sehr einfach reproduzieren. Dies spart Zeit und verschafft Konsistenz.

Die Umsetzung des IaC-Ansatzes wird in diesem System größtenteils mit Amazon CloudFormation und dem AWS Serverless-Application-Model realisiert. Innerhalb der Projektstruktur wird eine Konfigurationsdatei existieren, in der die AWS-Ressourcen definiert werden. Diese Datei wird versioniert werden, so dass alle Änderungen jederzeit eingesehen werden können. Um die Ressourcen zu erstellen, wird die Konfigurationsdatei an CloudFormation geschickt. Die Konfiguration der CodePipeline wird auch mit IaC durchgeführt, näheres dazu im folgenden Abschnitt. Außerdem werden auch die Dependencies des Systems mit IaC definiert. Es soll eine 'package.json'-Datei geben, welche alle benötigten Packages und die zugehörige Version benennt. Dies soll das Aufsetzen des Systems vereinfachen und beschleunigen.

5.3 Continuous Integration / Continuous Deployment

Um die Randbedingung 3 und 4 zu erfüllen, wird das System mit der Continous Integration / Continous Deployment-Methode erstellt und bereitgestellt. Im Folgenden wird auf die Methode näher eingegangen und erläutert, wie es konkret für das System umgesetzt werden soll:

Der Begriff Continous Integration / Continous Deployment (Continous Integration / Continuous Development (CI/CD)) beschreibt übergeordnet einen Prozess, der schnelle Integration lokaler Änderungen in die Anwendung, automatisierte Tests und Builds und vieles mehr beinhalten kann.

Im Folgenden werden die laut Martin Fowler und Matt Foemmel wichtigsten CI/CD Praktiken erläutert [30]:

Single Source Repository
Software Projekte werden in einem einzigen Version Control System (VCS) Repository abgespeichert. Dieses Repository enthält nicht nur den Source Code, sondern auch Tests, Property Dateien, Datenbank Schemata, sowie auch Infrastructure as Code. Nach Fowler enthalten Repositories alle Dateien, die notwendig sind um die Software zu builden.

Build Automatisierung
Das Software-Building wird durch Tools wie make, Maven oder Amazon CodeBuild automatisiert. Bei jedem Commit wird ein Build erstellt. Der Buildprozess enthält außer dem Kompilieren und Linken auch das Erstellen einer Ausführungsumgebung. Jemand, der den Source Code neu erhält, ist somit anhand eines einzigen Commands in der Lage, das System zum Laufen zu bringen.

Zudem werden beim Builden die Tests automatisch durchlaufen. Zwar können Tests nicht die Abwesenheit von Bugs beweisen, dennoch sind, laut Fowler, unvollständige Tests, die regelmäßig ausgeführt werden, besser als gar keine Tests.

Staging-Umgebung
Um die Anzahl an Fehlern zu vermeiden, die in der Produktion auftreten, werden Softwaresysteme nach Änderung in einem Klon der Produktionsumgebung getestet, der Staging-Umgebung. Hierbei wird die Software dem Product Owner und den Kunden für die Abnahme zur Verfügung gestellt. Die Staging-Umgebung soll so wenig wie möglich von der Produktionsumgebung abweichen.

Automatisches Deployment:
Da man unter Umständen mehrere Ausführungsumgebungen hat, empfiehlt es sich, um Aufwand zu sparen, auch das Software Deployment zu automatisieren.

Für die Umsetzung von CI / CD wird Amazon CodePipeline genutzt. Die Pipeline wird, durch Infrastructure as Code, mit einer Konfigurationsdatei im YAML Format automatisch erstellt. Im Folgenden werden die fünf Phasen der Pipeline näher erläutert:

1. **CodeCommit**

   Sobald ein Entwickler ein Softwareupdate zu einem in der Konfigurationsdatei spezifizierten CodeCommit Repositoriy schickt, startet die Pipeline. In dieser ersten Phase wird das Repository geklont, und das dadurch entstandene Artefakt als Output zur Verfügung gestellt.

2. **CodeBuild**

   Die zweite Phase besteht aus einem CodeBuild-Container, der in diesem Falle ein NodeJS-Environment enthält. Dieser erhält als Input das Output-Artefakt der vorangegangen Phase. Innerhalb des geklonten Systems werden vorher konfigurierte Befehle ausgeführt, wie die Installation von externen Packages, das Ausführen von Tests und die Kompilation und Erstellung von lauffähiger Software aus dem Code. Im letzten Schritt wird die Konfigurationsdatei der AWS-Infrastruktur an CloudFormation geschickt.

3. **CloudFormation Staging Environment**

   CloudFormation erstellt daraufhin in der dritten Phase einen Stack, welcher alle Amazon-Services der Konfigurationsdatei enthält. Jetzt läuft das System inklusive aller Services in einer Staging-Umgebung. Diese ist, abgesehen von AWS-Ressourcen-Namen, identisch mit der, die später tatsächlich genutzt wird. Hier können die Entwickler das System testen und auf Bugs hin untersuchen.

4. **User Approval**

   Um das System dem Endnutzer bereit zu stellen, muss ein Entwickler in der vierten Phase der Pipeline die Freigabe des Systems genehmigen. Dies geschieht in der GUI von CodePipeline.

5. **CloudFormation Production-Environment**

In der letzten Phase, der Bereitstellung und Verteilung der Software, erstellt CloudFormation einen zweiten Stack mit Hilfe der Konfigurationsdatei. Nun läuft das System auch in einer Production Umgebung, parallel zu der immer noch existenten Staging Umgebung.

Die folgende Grafik visualisiert den Aufbau der CodePipeline des zu fertigenden Systems:

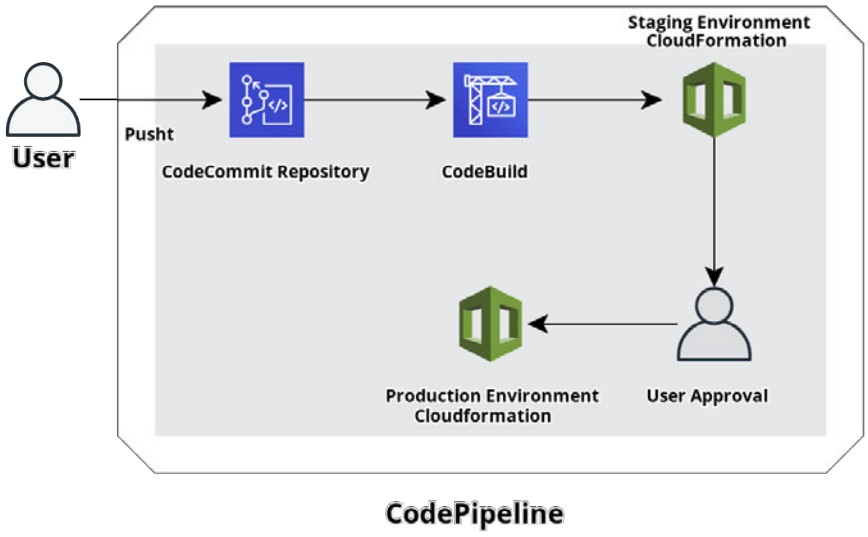


Abbildung 5.5: Visualisierung der CodePipeline

5.4 Tests

Um die Qualitätsanforderungen in Hinblick auf Funktionalität und Zuverlässigkeit zu erfüllen, wird die Software sowohl durch Unittests als auch durch einen umfangreichen Systemtest geprüft werden. Die Unittests werden mit dem Jest Framework implementiert und sollen eine größtmögliche Codecoverage haben.

Generelle Systemtests werden einerseits durch die Staging-Umgebung und andererseits durch eine parallele Verarbeitung der E-Mails vom System und von Menschenhand ermöglicht. Über einen längeren Zeitraum wird jede EML Datei von dem System und von

einem Menschen verarbeitet. Währenddessen werden die jeweiligen Ergebnisse miteinander verglichen und abgespeichert. Dies ermöglicht eine, zu einem späteren Zeitpunkt angesetzte, Auswertung.

Die Auswertung der Performance, bei der Verarbeitung von PDF-Dateien durch die OCR-Schnittstelle wird mit einem Vergleich der Ergebnisse, die das System und ein Mensch auf Testdaten erzielen, ermöglicht.

6 Realisierung

Dieses Kapitel stellt die Realisierung des Systems dar. Dabei näher darauf eingegangen, welche Schwierigkeiten sich bei der Realisierung ergaben, sowie die Frage, ob das System den Anforderungen standhält.

Das System wurde in TypeScript, einer auf dem ECMAScript-6-Standard basierenden, objektorientierten Programmiersprache implementiert. Als Paketmanager wurde npm benutzt. Erwähnenswerte Packages, die benutzt wurden, sind das offizielle AWS Typescript-SDK, ein Wrapper um das Commandlinetool imagemagick, mailparser zum Einlesen der EML-Dateien [16], und 'jest' für das Testen des Systems. Das AWS-Ökosystem wurde erstellt, wie es im Entwurf geplant wurde.

6.1 Funktionale Anforderungen

Alle funktionalen Anforderungen konnten bei der Realisierung erfüllt werden. Die vierte funktionale Anforderung, die Erkennung und Speicherung von falschen Werten, ist dabei so ausführlich, wie der Zeitrahmen es ermöglichte, implementiert worden. Bei der Realisierung der funktionalen Anforderung gab es keine nennenswerten Schwierigkeiten.

6.2 Qualitätsanforderungen

- **Funktionalität:**
  Das System hat alle Funktionalen Anforderung implementiert und somit die Qualitätsanforderung an die Funktionalität erfüllt.
- **Zuverlässigkeit:**
  Um die Zuverlässigkeit des Systems zu gewährleisten, wurden Unit-Tests implementiert. Für die Erstellung der Tests wurde das 'jest' Package genutzt. Dieses

erstellt auf Wunsch einen Bericht über die Testabdeckung. Für das realisierte System beträgt die Abdeckung 99,86 Prozent. Die Abbildung 6.1 zeigt den Bericht:

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	99.86	99.33	98.21	99.86	
domain/model	100	100	100	100	
Address.ts	100	100	100	100	
Claim.ts	100	100	100	100	
Clerk.ts	100	100	100	100	
ContactPerson.ts	100	100	100	100	
Damage.ts	100	100	100	100	
Evaluation.ts	100	100	100	100	
MoneyAmount.ts	100	100	100	100	
Name.ts	100	100	100	100	
Policy.ts	100	100	100	100	
Policyholder.ts	100	100	100	100	
Reserve.ts	100	100	100	100	
Result.ts	100	100	100	100	
ThirdParty.ts	100	100	100	100	
domain/service	100	99.47	100	100	
AdditionalInfoExtractorService.ts	100	100	100	100	
ClerkExtractorService.ts	100	100	100	100	
ContactPersonExtractorService.ts	100	100	100	100	
DamageExtractorService.ts	100	100	100	100	
EvaluationService.ts	100	75	100	100	18
FormatService.ts	100	100	100	100	
FormatServiceFactory.ts	100	100	100	100	
GuidewireFormatService.ts	100	100	100	100	
NessyFormatService.ts	100	100	100	100	
PolicyExtractorService.ts	100	100	100	100	
PolicyholderExtractorService.ts	100	100	100	100	
ServiceAddressExtractorService.ts	100	100	100	100	
ThirdPartyExtractorService.ts	100	100	100	100	
infrastructure/service	98.55	96.43	93.33	98.53	
CompareService.ts	100	100	100	100	
CounterService.ts	100	100	100	100	
DateFormatService.ts	100	100	100	100	
PhoneNumberParser.ts	100	100	100	100	
SimpleEmailParser.ts	87.5	50	50	87.5	16
TextService.ts	100	100	100	100	

Abbildung 6.1: Testabdeckung des Systems durch Unit Tests

Systemtests wurden durch die Bereitstellung einer Staging-Umgebung in der Cloud ermöglicht. Außerdem wurden zwei Tests implementiert, die lokales Testen der Use-Cases ermöglichen.

- **Benutzbarkeit:**
  Der Software liegt eine ausführliche Readme-Datei bei, in der erklärt wird, wie das

System aufzusetzen ist und wie es funktioniert. Benötigte Dependencies können durch den Befehl 'npm install' automatisch installiert werden. Fehlermeldungen und weitere Logs sind im AWS-Service CloudWatch einsehbar. Falls fehlerhafte Daten in das System eingepflegt werden, können diese über die graphische Oberfläche von DynamoDB anhand der Schadennummer gelöscht werden.

- **Effizienz:**
  Die Qualitätsanforderungen hinsichtlich der Effizienz konnten erfüllt werden. Die folgende Abbildung zeigt die Metriken der ParseEmail Lambda. Dies sind die Statistiken für die zwei Monate, in der die Lambda in Betrieb war. Die durchschnittliche Länge liegt bei 343 Millisekunden und die längste Ausführung betrug 1,83 Sekunden, was deutlich unter der angesetzten Obergrenze von 10 Sekunden liegt. Die Lambda-Funktionen kommen mit einer Memory-Size von 1024 Megabytes aus, wodurch die Kosten auf einem Minimum gehalten werden.

Abbildung 6.2: CloudWatch Metriken der ParseEmail Lambda

- **Änderbarkeit:**
  Das System wurde in Clean Archtitecture realisiert, wodurch neue Use-Cases oder Services ohne Probleme implementiert werden können. Hierfür muss jeweils nur eine Klasse bearbeitet werden.

- **Übertragbarkeit:**
  Durch die Clean Architecture ist es unproblematisch, Services, wie zum Beispiel DynamoDB gegen andere Services auszutauschen. Es müsste lediglich die Anbindung an den Service in einer Klasse implementiert werden, die dann in die Application injiziert wird. Der restliche Code bleibt unangetastet.

  Die Anforderung an die Übertragbarkeit hinsichtlich des Cloudanbieters konnte nicht in Gänze erfüllt werden. Zwar wurde Infrastructure as Code benutzt, was das Austauschen von Services innerhalb des Amazon Ökosystems problemlos ermöglicht; da jedoch Cloudformation als IaC-Service gewählt wurde, befindet sich das System in einem Vendor-Lock-in. Inwiefern dies verhindert werden könnte, wird in

Abschnitt 8.2 näher erläutert. Innerhalb von AWS kann das System ohne Probleme migriert werden. Durch die Nutzung von npm als Package-Manager, werden die benötigten Packages mit einem einzigen Command installiert.

6.3 Randbedingungen

- **Randbedingung 1:**
  Die erste Randbedingung wurde erfüllt. Das System wurde im AWS-Ökosystem entwickelt und konnte mit anderen, schon bestehenden, Services integriert werden.
- **Randbedingung 2:**
  Die ganze AWS-Infrastruktur wurde mit Infrastructure as Code errichtet. Kein Service wurde über die GUI eingerichtet und der Code wurde versioniert, wodurch diese Randbedingung erfüllt wurde.
- **Randbedingung 3:**
  Randbedingung 3 konnte ohne Probleme erfüllt werden. Das System wird automatisiert über CodePipeline gebuildet und deployet. Sobald Code ins Repository gepusht wird, läuft die Pipeline durch.
- **Randbedingung 4:**
  Die vierte Randbedingung konnte erfüllt werden. Für das System existiert eine Production- und eine Staging-Umgebung. Die Staging-Umgebung wird genutzt, um neue Änderungen am System zu testen. Sobald diese abgenommen werden, werden die Änderungen in die Production-Umgebung propagiert.

7 Auswertung

In diesem Kapitel wird die EML- und PDF-Verarbeitung des Systems evaluiert. Die Evaluation wird erläutert und die Ergebnisse vorgestellt. Im Fazit werden die Ergebnisse interpretiert und bewertet.

7.1 Evaluation der EML Verarbeitung

Dieser Abschnitt befasst sich mit der EML-Verarbeitung, sprich dem Auslesen von Werten aus den E-Mails. Hierbei wird die Korrektheit der Werte aus den ausgelesenen Felder getestet.

7.1.1 Testaufbau

Eine Auswertung ist tendenziell aussagekräftiger, umso größer die betrachte Sample-Größe ist, in diesem Fall die Anzahl der verarbeiteten E-Mails. Um eine relativ große Anzahl zu erlangen, lief die automatische Verarbeitung über zwei Monate parallel zur manuellen Verarbeitung. In diesem Zeitraum wurden insgesamt 1020 E-Mails verarbeitet. Für jede dieser E-Mails wurden die automatisch extrahierten Informationen mit den von Menschen abgetippten Informationen abgeglichen. Wurde beim Vergleich eine Differenz der Werte entdeckt, wurde dies als Fehler des automatischen Systems vermerkt. Wichtig dabei anzumerken ist, dass hierbei implizit davon ausgegangen wurde, dass bei der manuellen Verarbeitung keine Fehler auf Seiten der Arbeiter auftreten, was eine naive Annahme ist. Jedoch wäre der Aufwand um auf manueller Seite eine zweite Überprüfung der Werte durchzuführen, meiner Ansicht nach, unverhältnismäßig. Zudem sind die Mitarbeiter, die die manuelle Verarbeitung durchgeführt haben, ausgebildete Fachkräfte, weshalb die Fehlerquote als verhältnismäßig niedrig antizipiert werden kann. Damit ein Feld später automatisch im System angelegt werden darf, setzt claimsforce eine Fehlerrate, welche niedriger als drei Prozent ist, voraus.

7.1.2 Evaluationsergebnisse

Insgesamt enthält eine Mail 32 Felder. In der Abbildung 7.1 ist aufsteigend die jeweilige Anzahl der Fehler, für die im Anwendungsfall wichtigsten acht Felder zu sehen. Diese Felder befinden sich auf der X-Achse, die Y-Achse repräsentiert die Anzahl der Fehler. Die Y-Achse geht bis 1020, damit ein Gefühl der Relation von Fehlerzahl zur Gesamtanzahl der verarbeiteten E-Mails geschaffen wird.

Für die Felder mit geringer Fehlerzahl, claimId und dateOfDamage, wurden Binomialverteilungen mit unterschiedlichen Wahrscheinlichkeiten, die in diesem Fall für die Korrektheitsraten eines Feldes stehen, erstellt. In die Wahrscheinlichkeitsfunktionen der Verteilungen, $\binom{N}{k} \cdot p^k \cdot (1-p)^{(n-k)}$, wurden anschließend die Ergebnisse der Verarbeitung des einzelnen Feldes eingesetzt. Dadurch erhält man die Wahrscheinlichkeit, dass dieses Ergebnis zu jener Binomialverteilung gehört. Für N wird die Sample-Größe, 1020, eingesetzt, k ist die Anzahl richtig erkannter Ergebnisse und p ist in diesem Fall die Korrektheitsrate. Die Wahrscheinlichkeiten werden in Abhängigkeit zu den Korrektheitsraten visualisiert. Da wir mit diskreten Werten arbeiten, kann nicht einfach integriert werden. Deswegen wurde an dieser Stelle mit einem Verfahren der numerischen Integration, der zusammengesetzten Trapezformel, gearbeitet. Diese ermöglicht es, basierend auf den Sample-Daten, in dem die Quadratur über einem bestimmten Intervall durch die Quadratur des Gesamtbereichs geteilt wird, einzuschätzen, mit welcher Wahrscheinlichkeit die Korrektheitsrate in diesem Intervall liegt.

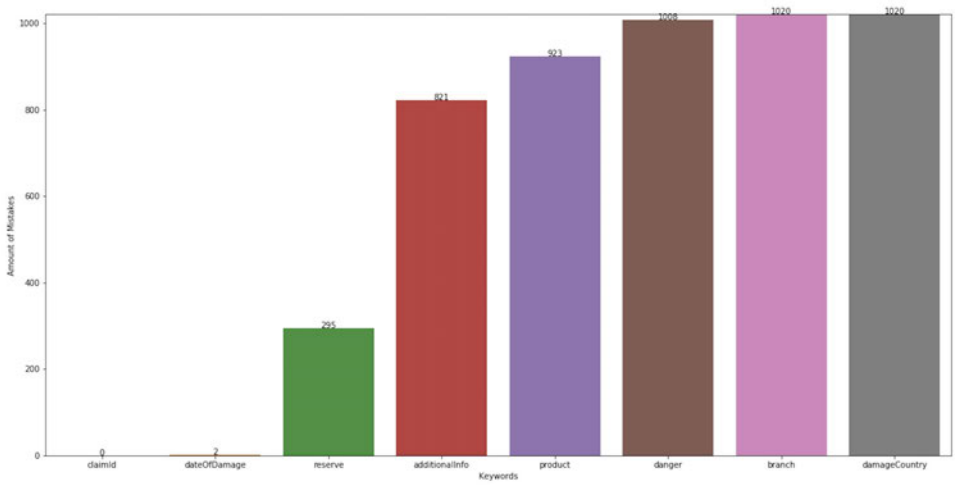


Abbildung 7.1: Fehlerzahl der Felder auf 1020 verarbeiteten E-Mails

- **claimId:** Die Schadennummer wurde keinmal falsch erkannt. Dies ist das einzige notwendige Feld, um im Alt-System einen Schaden anzulegen. Dies bedeutet, dass die automatische Schadenanlage umgesetzt werden kann. In der folgenden Abbildung ist die Wahrscheinlichkeit möglicher, dem Ergebnis unterliegenden Korrektheitsraten zu sehen.

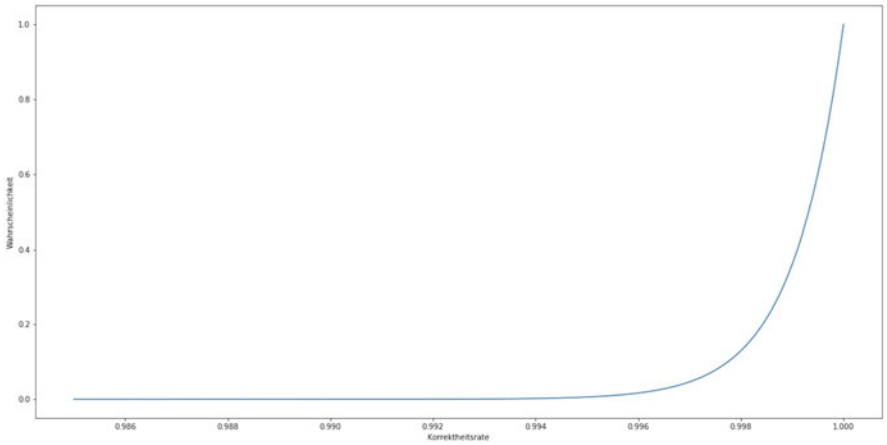


Abbildung 7.2: Wahrscheinlichkeiten der Korrektheitsrate für die Schadennummer

Die Approximation des Flächeninhalts des Intervall $[0.99, 1]$ geteilt durch die des Gesamtbereichs, $[0, 1]$, ergibt ≈ 0.99996. Dies repräsentiert die Wahrscheinlichkeit, mit der die Korrektheitsrate höher als 99 Prozent ist.

- **dateOfDamage:** Das Schadendatum ist zweimal falsch erkannt worden.

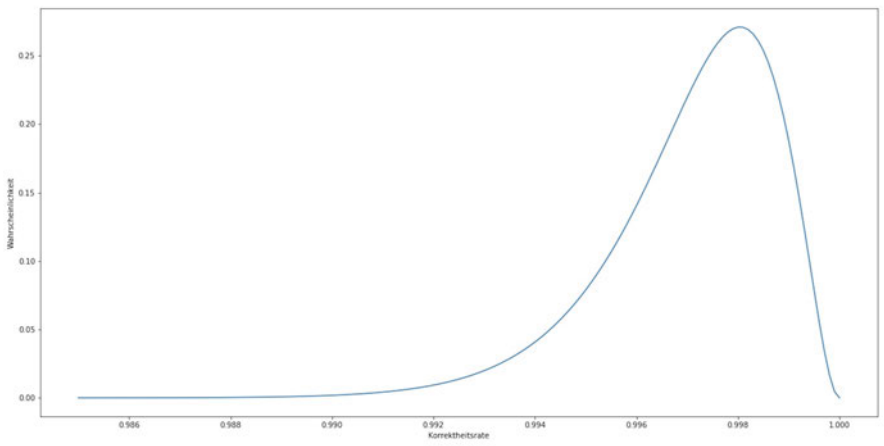


Abbildung 7.3: Wahrscheinlichkeiten der Korrektheitsrate für das Schadendatum

Die Approximation des Flächeninhalts des Intervall $[0.99, 1]$ geteilt durch die des Gesamtbereichs, $[0, 1]$, ergibt ≈ 0.99775. Dies repräsentiert die Wahrscheinlichkeit, mit der die Korrektheitsrate höher als 99 Prozent ist.

- **reserve:** Die Reserve wurde 295-mal falsch erkannt. Dies resultiert daraus, dass die ursprünglich eingegebene Reserve im Freitext der E-Mail noch einmal korrigiert wird. Dies kann ein Mensch bei der Bearbeitung berücksichtigen, das System jedoch nicht.
- **additionalInfo:** Bei der Erkennung der additionalInfo, des Freitexts, ist das Ergebnis des Systems in 821 der 1020 Fälle von der des manuell verarbeiteten Ergebnisses abgewichen. Dies ist dadurch bedingt, dass bei der manuellen Verarbeitung zum Teil Informationen hinzugefügt oder Rechtschreibfehler korrigiert werden.
- **product:** Die Auswahl des Produkts kontextabhängig. Deshalb wurde das Produkt 923-mal falsch erkannt. In der E-Mail steht häufig nicht der ganze Name des Produkts. Der Name ist abhängig von den anderen Feldern, was das System nicht berücksichtigt.
- **danger:** Die Schadenursache wurde 1008 mal falsch erkannt. Die Werte, die sich für die Schadenursache in der E-Mail befinden, werden im Alt-System manuell auf andere Werte gemappt. Aus 'LW' wird 'Leitungswasser/Rohrbruch. Dieses Mapping wird bei der automatischen Verarbeitung jedoch noch nicht vorgenommen.
- **branch:** Die Schadensparte wurde kein einziges mal richtig erkannt. Auch bei diesem Feld werden die Werte aus der E-Mail von den Mitarbeitern auf andere Werte im Alt-System gemappt, was das realisierte System aber nicht kann.
- **damageCountry:** Das Land, in dem der Schaden aufgetreten ist, wurde kein einziges mal richtig erkannt. Dies ist dadurch zu erklären, dass bei der manuellen Verarbeitung die ISOCodes der Länder auf die Ländernamen gemappt wurden. Die automatische Verarbeitung entnimmt der E-Mail 'DE' als damageCountry, wohingegen der Mensch bei der manuellen Verarbeitung 'Deutschland' eingibt.

Insgesamt konnten 4 der 32 Felder eine Fehlerrate von unter 3 Prozent erreichen. Pro E-Mail liegt die durchschnittliche Fehlerrate bei $\approx$ 29,41862 Prozent, sprich ≈ 9 der 32 Felder werden im Durchschnitt fehlerhaft erkannt.

7.2 Evaluation der PDF-Verarbeitung

Dieser Abschnitt befasst sich mit der PDF-Verarbeitung. Es wird die Performance zweier OCR Systeme, Tesseract und Textract, verglichen.

7.2.1 Testaufbau

Um die optische Zeichenerkennung zu evaluieren, wurden insgesamt 9 PDFs von jeweils Tesseract und Amazon Textract verarbeitet. Insgesamt wurden 88181 Zeichen verarbeitet. Die Menge der PDF-Dateien besteht zum größten Teil aus Rechnungen, aber auch aus anderen Dokumenten, wie zum Beispielen Ausschnitten aus Büchern oder Ähnlichem.

Bei Textract wurde die Dokumenten-Analyse benutzt, die, abgesehen von der Texterkennung, auch Tabellen und Formulare erkennt. Dies ist für die Entscheidung, welcher Service besser für den Anwendungsfall geeignet ist, von Vorteil, da viele der angehängten PDFs Rechnungen sind. Es wirkt sich jedoch nicht auf den Vergleich der Texterkennung aus.

Die PDFs wurden für Tesseract in TIFFs konvertiert. Ein entscheidender Faktor bei der Performance von Tesseract ist die Vorverarbeitung der Bilder. In diesem Fall wurden die TIFFs nur monochrom gemacht.

Die verarbeiteten PDFs wurden manuell abgetippt und überprüft. Etwaige, bei der manuellen Übertragung entstandene Flüchtigkeitsfehler, könnten sich auf das Ergebnis auswirken. Um eine qualitative Aussage über die Performance der beiden Texterkennungssysteme treffen zu können, wurden sowohl die manuell abgetippten, als auch die beiden automatisch erkannten Texte, von Absätzen befreit und zu einem String konkateniert. Nun wurde jeweils die Levenshtein-Distanz der manuellen Version zu der, der durch Textract und Tesseract erstellten Versionen berechnet.

7.2.2 Evaluationsergebnisse

Die Tabelle 7.1 enthält die Ergebnisse des Tests. Jede Reihe steht für ein verarbeitetes PDF. Für jedes PDF lässt sich die Zahl der enthaltenen Wörter und die beiden errechneten Levenshtein-Distanzen zwischen dem manuell abgetippten Text und den von Textract und Tesseract verarbeiteten Texten ablesen.

PDF	**Anzahl der Zeichen**	**Levenshtein Textract**	**Levenshtein Tesseract**
0	8568	987	2673
1	3336	312	293
2	4161	235	530
3	13555	323	9493
4	1401	216	112
5	741	5	0
6	19917	551	144
7	12116	410	1785
8	24386	524	199
Summe	88181	3563	15229

Tabelle 7.1: Ergebnisse des OCR Vergleichs

Insgesamt wurden 88181 Zeichen aus 9 verschiedenen PDFs verarbeitet. Die für alle PDFs aufaddierte Levenshtein-Distanz Textracts beträgt 3563. Daraus resultiert eine Fehlerquote von $\approx$ 4,04055 Prozent. Dies bedeutet, dass ungefähr jeder 25. Charakter falsch erkannt wurde. Die aufaddierte Levenshtein-Distanz der von Tesseract erzeugten Texte beträgt 15229. Das ergibt eine Fehlerquote von $\approx$ 17,27016 Prozent. Tesseract hat somit ungefähr jeden 6. Charakter falsch erkannt.

Bei der genaueren Betrachtung der Ergebnisse zeigt sich, dass Textract besonders gute Ergebnisse beim PDF mit der Nummer 3 erzielt hat. Dieses PDF ist eine fünfseitige Rechnung mit einer langen Tabelle an Bestellpositionen. Abbildung 7.4 zeigt einen Ausschnitt aus dieser Rechnung. Tesseract hat in diesem Fall eine Levenshtein-Distanz von 9493 zum manuell abgetippten Text, wohingegen Textracts Levenshtein-Distanz nur 323 beträgt. Dies könnte an der Tabellendetektion von Textract liegen, auf die im weiteren Verlauf des Abschnitts noch eingegangen wird.

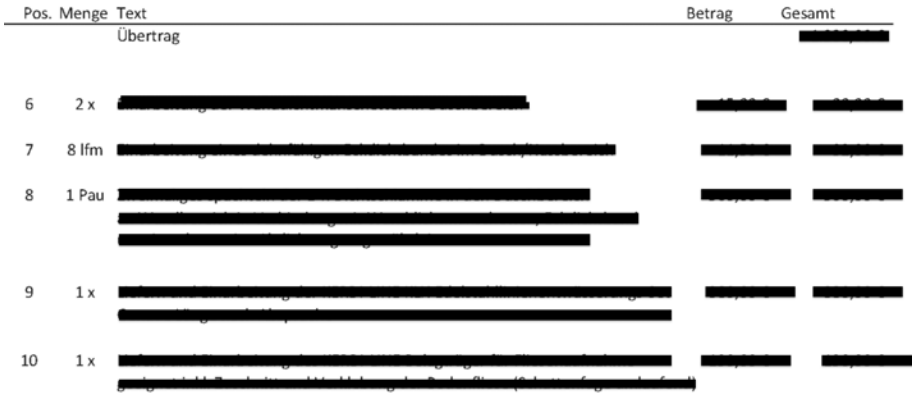
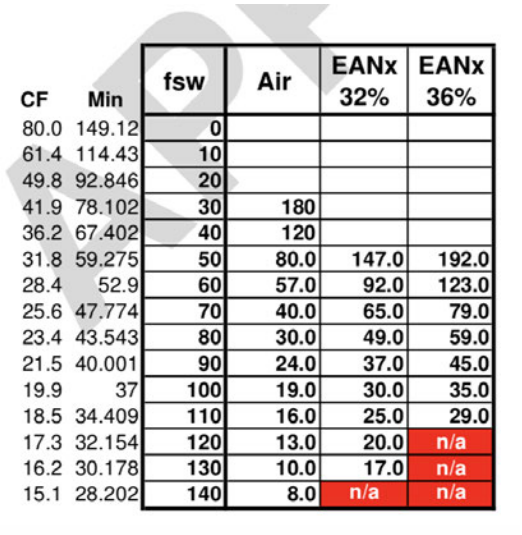


Abbildung 7.4: Geschwärzter Auszug aus PDF Nr. 3

Bei Betrachtung der Ergebnisse fällt auf, dass Textract weder Umlaute, noch das 'ß' richtig erkennt. In den Ergebnissen Textracts befinden sich anstelle von 'Ä,Ö,Ü' 'A,O,U'. 'ß' interpretiert Textract als ein 'B'. Es ist jedoch anzunehmen, dass Textract in zukünftigen Versionen die Erkennung deutscher Sonderzeichen implementieren wird.

Bei der Textanalyse eines Dokuments erhält man von Textract, abgesehen vom 'Raw Text', auch alle erkannten Tabellen und die erkannten Key-Value Paare als CSV-Dateien. Im Folgenden wird auf die Ergebnisse der Tabellen- und Formularerkennung von Textract näher eingegangen.

- **Formularerkennung:** Bei der Formularerkennung hat Textract die meisten Key-Value-Paare richtig erkannt. Darüber hinaus hat es aber an äußerst vielen Stellen ganz normalen Text als Key-Value-Paare missinterpretiert. Wenn bekannt ist, wonach in den Texten gesucht werden soll, kann diese Funktion hilfreich sein. Ansonsten müssten die Ergebnisse weiterverarbeitet und die falsch erkannten Paare herausgefiltert werden.
- **Tabellenerkennung:** Auch die Tabellenerkennung konnte nur gemischte Ergebnisse erzielen. Zwar wurden die Tabellen aus den Dokumenten zu einem großen Teil richtig erkannt, dennoch hat Textract an vielen Stellen Text als Tabellen missinterpretiert. Die Abbildung 7.5 zeigt eine Tabelle aus dem PDF mit der Nummer 0.



CF	Min	fsw	Air	EANx 32%	EANx 36%
80.0	149.12	0			
61.4	114.43	10			
49.8	92.846	20			
41.9	78.102	30	180		
36.2	67.402	40	120		
31.8	59.275	50	80.0	147.0	192.0
28.4	52.9	60	57.0	92.0	123.0
25.6	47.774	70	40.0	65.0	79.0
23.4	43.543	80	30.0	49.0	59.0
21.5	40.001	90	24.0	37.0	45.0
19.9	37	100	19.0	30.0	35.0
18.5	34.409	110	16.0	25.0	29.0
17.3	32.154	120	13.0	20.0	n/a
16.2	30.178	130	10.0	17.0	n/a
15.1	28.202	140	8.0	n/a	n/a

Abbildung 7.5: Beispielhafte Tabelle aus dem PDF Nr. 0

Diese Tabelle wurde von Textract komplett richtig erkannt. Die Tabelle 7.2 enthält die Ergebnisse, welche bei der Dokumentanalyse als CSV-Datei zurückgegeben wurden.

CF	**Min**	**fsw**	**Air**	**EANx 32**	**EANx 36**
80.0	149.12	0			
61.4	114.43	10			
49.8	92.846	20			
41.9	78.102	30	180		
36.2	67.402	40	120		
31.8	59.275	50	80.0	147.0	192.0
28.4	52.9	60	57.0	92.0	123.0
25.6	47.774	70	40.0	65.0	79.0
23.4	43.543	80	30.0	49.0	59.0
21.5	40.001	90	24.0	37.0	45.0
19.9	37	100	19.0	30.0	35.0
18.5	34.409	110	16.0	25.0	29.0
17.3	32.154	120	13.0	20.0	n/a
16.2	30.178	130	10.0	17.0	n/a
15.1	28.202	140	8.0	n/a	n/a

Tabelle 7.2: Ergebnisse der Tabellendetektion Textracts auf der Tabelle 7.5

Abschließend lässt sich sagen, dass Textract bei der reinen Texterkennung mit einer durchschnittlichen Fehlerrate von ≈ 4.04055 Prozent besser abgeschnitten hat als Tesseract. Darüber hinaus hat die Dokumenten-Analyse Textracts Resultate hervorgebracht, die zum Teil beeindruckend gute Ergebnisse, siehe Tabelle 7.2, aber auch schlechte Ergebnisse enthalten.

7.3 Methodische Abstraktion

Die Technik der optischen Zeichenerkennung liefert inzwischen eindrucksvolle Ergebnisse. Insbesondere Amazon Textract, welches auf Parametrisierung verzichtet, kommt bei der Verarbeitung von eingescannten Dokumenten auf eine sehr geringe Fehlerzahl. Trotz geringer Fehleranzahl ist die optische Zeichenerkennung für bestimmte Bereiche jedoch weniger geeignet als für andere. Insbesondere im Anwendungsfall ist die Korrektheit der erkannten Zeichen von höchster Wichtigkeit. Eine falsch erkannte Zahl hat innerhalb einer Rechnung große Auswirkungen. Aufgrund dessen ist die optische Zeichenerkennung in dieser Domäne nur unter Einhaltung großer Vorsichtsmaßnahmen einzusetzen. In anderen Bereichen kann die optische Zeichenerkennung jedoch uneingeschränkt eingesetzt werden, wie zum Beispiel bei der Unterstützung blinder Menschen. Menschen mit Sehbehinderung können sich eingescannte Texte mit Hilfe der optischen Zeichenerkennung vorlesen lassen. Hierbei ist es nicht von gravierender Bedeutung, ob ein Zeichen richtig oder falsch erkannt wird, da das verarbeitete Wort aus dem Kontext geschlossen werden kann.

Die Evaluationsergebnisse legen nahe, dass in Bereichen, in denen die erkannten Zeichen Informationen von großer Relevanz abbilden, die optische Zeichenerkennung mehr als eine Assistenz der menschlichen Arbeit fungiert, anstatt diese komplett zu ersetzen. Außerdem ist das Risiko von schwerwiegenden Konsequenzen aufgrund von falschen Zeichen in zahlenlastigen Texten größer als bei Texten, die wenig Zahlen enthalten. Falsch erkannte Zeichen in Wörtern weniger problematisch, da die ursprünglichen Zeichen sich meistens aus dem Kontext ergeben.

8 Fazit

In diesem Kapitel werden die Ergebnisse der vorangegangenen Tests vor dem Hintergrund des Anwendungsfalls betrachtet und bewertet. Darauf aufbauend wird ein Ausblick darüber gegeben, wie das System weiterentwickelt werden könnte. Zudem wird erläutert, was aus Zeit- oder Komplexitätsgründen wegfallen musste. Zum Schluss wird eine kurze Zusammenfassung gegeben.

8.1 Bewertung

Das realisierte System erfüllt alle Funktionalen Anforderungen. Außerdem erfüllt es, bis auf die Übertragbarkeit in Hinblick auf den Cloud Anbieter, alle Qualitätsanforderungen und Randbedingungen. Die Tests des System haben gezeigt, dass die automatische Schadenanlage ein komplexer Prozess ist. Viele Informationen erschließen sich nur aus dem Kontext, was das System vor eine große Herausforderung stellt. Dennoch kann claimsforce durch das System immens viel Zeit sparen. Das grundlegende Ziel des Systems, die Schadenanlage zu automatisieren, ist erfüllt worden. Die Schadensnummer, das einzige Feld welches zur Anlage eines Schadens notwendig ist, wurde bei über 1000 verarbeiteten E-Mails kein einziges Mal falsch erkannt. Darüber hinaus wurden drei weitere Felder mit einer Fehlerrate von unter drei Prozent erkannt, so dass sie, den Anforderungen claimsforces entsprechend, bei der Schadenanlage automatisch mit eingetragen werden können.

Für die Verarbeitung der PDFs wurden zwei verschiedene OCR-Systeme implementiert. Im Hinblick auf die Performance hat sich eindeutig die proprietäre Lösung von Amazon durchgesetzt. Die Anzahl der Fehler bei der Erkennung Textracts beliefen sich auf gerade mal ein Viertel der Fehler von Tesseract. Dennoch ist Textracts Fehlerquote, die sich auf vier Prozent beläuft, zu hoch, um den Anforderungen der Texterkennung in der Rechnungsdomäne zu entsprechen. Auch im Hinblick auf die Funktionalität schneidet Textract

deutlich besser ab. Tabellen- und Formulardetektion sind, auch wenn sie noch weit entfernt von Perfektion sind, Features, die im besonders im Anwendungsfall von erheblichem Nutzen sind. Inwiefern sich die Performance von Textract, als auch von Tesseract, möglicherweise optimieren lassen, wird im Ausblick näher erläutert. Hinsichtlich der Kosten schneidet selbstverständlich Tesseract als Free-and-Open-Source-Software besser ab als Textract. Allerdings belaufen sich die Kosten Textracts für die ersten 1 Millionen Seiten, bei einer geschätzten Anforderung von 1000 Seiten pro Monat, inklusive Tabellen- und Formatdetektion, auf gerade einmal 65 Dollar pro Monat.

Alles in allem hat Textract in Hinblick auf den Anwendungsfall deutlich besser als Tesseract performt. Ergänzend kommt hinzu: Dadurch, dass alle claimsforce-Services im AWS-Ökosystem laufen, bereitet die Einbindung Textracts keine großen Schwierigkeiten. Zudem ist weder Vorverarbeitung der PDFs, noch eine Parametrisierung von Nöten. Dies lässt Textract als eindeutigen Sieger aus diesem Vergleich hervorgehen.

8.2 Ausblick

Im Folgenden wird darauf eingegangen was aus Zeitgründen nicht aufgegriffen werden konnte, und was a Posteriori verbessert werden kann:

Um die letzte offene Qualitätsanforderung, die Übertragbarkeit, in Gänze zu erfüllen und um einen Vendor Lock-in zu verhindern, könnte anstelle von CloudFormation ein externes IaC Framework wie zum Beispiel Serverless oder Terraform benutzt werden, das eine weitere Abstraktionsebene schafft.

Um die Performance der EML-Verarbeitung zu steigern, können mehrere Schritte vorgenommen werden. Die Fehlerrate von drei verschiedenen Feldern kann durch die Implementation eines Dictionaries deutlich gesenkt werden. Die Schadenursache, die Schadensparte und das Land, in welchem der Schaden verursacht wurde, haben im claimsforce System zum Teil andere Bezeichnungen als in den E-Mails. Dadurch, dass zwischen den Werten der verschiedenen Kontexte ein Eins-zu-eins Mapping existiert, kann die Fehlerrate bei diesen Feldern durch eine einfache Map gesenkt werden.

Eine weitere Möglichkeit, die Fehlerrate bestimmter Felder zu reduzieren, ist, das sogenannte 'Additional Info' Feld auf relevante Informationen hin zu untersuchen. Dieses Feld ist ein Freitext und enthält in manchen Fällen zusätzliche Informationen zu anderen Feldern. Dies kann soweit gehen, dass der Wert, welcher ursprünglich für das Feld erkannt

wurde, im Freitext 'überschrieben' wird. Dies zu implementieren wäre, vor allen Dingen in Hinblick auf die Korrektheit der aus dem Freitext extrahierten Informationen, aufwändig, ein möglicher Grund für die geringere Priorität bei der Realisierung und etwaigen Weiterentwicklung des Systems.

Um die jetzige Schadenanlage zu beschleunigen, könnte eine GUI im Altsystem implementiert werden, welche die E-Mail den vom realisierten System extrahierten Daten gegenüberstellt, so dass der Mitarbeiter per Knopfdruck bestätigen kann, ob die extrahierten Daten korrekt sind oder nicht. Dies wäre ein Schritt von der vollständig automatisierten Schadenanlage weg, hin zu einer vom System assistierten manuellen Schadenanlage.

Bei der PDF-Verarbeitung kann die Performance, sowohl bei Textract als auch bei Tesseract, durch Vor- und Nachbereitung der Daten gesteigert werden. Textract erkennt zum jetzigen Zeitpunkt keine deutschen Sonderzeichen, wodurch die Fehlerrate bei der Texterkennung stark beeinflusst wird. Es ist jedoch davon auszugehen, dass Textract in Zukunft die Unterstützung deutscher Sonderzeichen implementieren wird, wodurch die Performance in dieser Hinsicht gesteigert werden könnte. Die Fehlerrate Tesseracts kann durch Vorverarbeitung der als Input verwendeten Dateien reduziert werden. Um hierbei optimale Ergebnisse zu erzielen, müsste jedoch die Art der Vorverarbeitung an die jeweilige Datei angepasst werden, was jedoch zusätzliche, manuelle Arbeit erfordert. Für den speziellen Anwendungsfall würde dies dementsprechend nicht in Frage kommen, da die Reduzierung der manuellen Arbeit gerade das Ziel des Systems ist.

Durch Zeitmangel bedingt konnte nichts implementiert werden, was Informationen aus den Texterkennungsergebnissen extrahiert. Dies wäre der nächste größere Schritt bei der Weiterentwicklung des realisierten Systems. Ich habe mich, basierend auf dem Vergleich der beiden Systeme, für Textract entschieden. So könnte das System auch Informationen aus den von Textract erkannten Tabellen und Formularen extrahieren. Da eine solche Informationsextraktion auf den Ergebnissen der Texterkennung operieren würde, müsste implizit davon ausgegangen werden, dass diese Ergebnisse frei von Fehlern sind. Dies ist jedoch im Hinblick auf die Testergebnisse aus Abschnitt 7.2.2 empirisch unhaltbar. Deswegen müsste die Implementation einer Informationsextraktion einen besonders vorsichtigen Ansatz verfolgen.

8.3 Zusammenfassung

Die manuelle Schadenanlage in der Versicherungswirtschaft ist mit hohem Zeitaufwand und dementsprechend hohen Personalkosten verbunden. Die Mitarbeiter müssen für die Anlage des Schadens Dateien aus verschiedenen Datenquellen berücksichtigen, welche meist unstrukturiert sind und keinem Standard unterliegen. Das realisierte System ist der erste Schritt in die Richtung einer automatisierten Schadenanlage. Eintreffende E-Mails mit Informationen über den Schaden werden, anstatt von Mitarbeitern, vom System entgegengenommen, verarbeitet und in einer Datenbank gespeichert. Die Auswertung der Ergebnisse dieses Prozesses zeigen, dass unter Berücksichtigung eines vernachlässigbaren Fehlerrisikos ein Teil der Informationen über den Schaden automatisch vom System angelegt werden kann. Dennoch müssen weiterhin gewisse Informationen per Hand aus den E-Mails extrahiert werden. Die Anhänge der E-Mails, welche aus Rechnungen im PDF Format bestehen, werden vom System Amazon Textract übermittelt, das eine Dokumenten-Analyse auf den Rechnungen vornimmt. Die Ergebnisse der Analyse werden noch nicht für die automatische Schadenanlage berücksichtigt, sondern vorerst nur in einer Datenbank gespeichert.

Zeitlich bedingt konnten einige Features, wie zum Beispiel die Verarbeitung der Ergebnisse der optischen Zeichenerkennung, nicht realisiert werden. Dies stellt jedoch kein Problem dar, da aufgrund der Clean Architecture, in der das System verfasst wurde, dem System weitere Funktionalität hinzugefügt wird, ohne den bestehenden Code verändern zu müssen.

Literaturverzeichnis

[1] *AWS Dokumentation - Cloudformation.* https://docs.aws.amazon.com/cloudformation/. – Aufgerufen am: 30.03.2020

[2] *AWS Dokumentation - CodePipeline.* https://docs.aws.amazon.com/codepipeline/. – Aufgerufen am: 30.03.2020

[3] *AWS Dokumentation - DynamoDB.* https://docs.aws.amazon.com/dynamodb/. – Aufgerufen am: 23.04.2020

[4] *AWS Dokumentation - Lambda.* https://docs.aws.amazon.com/lambda/. – Aufgerufen am: 26.02.2020

[5] *AWS Dokumentation - Serverless Application Model.* https://docs.aws.amazon.com/serverless-application-model/. – Aufgerufen am: 30.03.2020

[6] *AWS Dokumentation - Simple Email Service.* https://docs.aws.amazon.com/ses/. – Aufgerufen am: 15.04.2020

[7] *AWS Dokumentation - Simple Notification Service.* https://docs.aws.amazon.com/sns/. – Aufgerufen am: 10.02.2020

[8] *AWS Dokumentation - Simple Storage Service.* https://docs.aws.amazon.com/s3/. – Aufgerufen am: 13.04.2020

[9] *AWS Dokumentation - Textract.* https://docs.aws.amazon.com/textract/. – Aufgerufen am: 05.01.2020

[10] *Bundesamt für Sicherheit in der Informationstechnik - Definition Cloud Computing.* https://www.bsi.bund.de/DE/Themen/DigitaleGesellschaft/CloudComputing/Grundlagen/Grundlagen_node.html. – Aufgerufen am: 13.02.2020

[11] *Email (Electronic Mail Format).* https://www.loc.gov/preservation/digital/formats/fdd/fdd000388.shtml. – Aufgerufen am: 02.04.2020

[12] *Example Diagram for asynchronous Textract Use.* https://docs.aws.amazon.com/de_de/textract/latest/dg/api-async.html. – Aufgerufen am: 11.03.2020

[13] *GDV Statistik.* https://www.gdv.de/de/zahlen-und-fakten/versicherungsbereiche/ueberblick-24074. – Aufgerufen am: 12.02.2020

[14] *ISO 32000-1:2008.* https://www.iso.org/standard/51502.html. – Aufgerufen am: 23.04.2020

[15] *Kurzweil Reading Machine.* http://www.kurzweiltech.com/kcp.html. – Aufgerufen am: 01.04.2020

[16] *Nodemailer - Mailparser.* https://nodemailer.com/extras/mailparser/. – Aufgerufen am: 25.04.2020

[17] *OCRopus.* https://github.com/tmbarchive/ocropy. – Aufgerufen am: 23.04.2020

[18] *PDF Reference, Third Edition, version 1.4.* https://www.adobe.com/devnet/pdf/pdf_reference_archive.html. – Aufgerufen am: 10.04.2020

[19] *Projekt Gutenberg.* https://www.projekt-gutenberg.org/index.html. – Aufgerufen am: 23.04.2020

[20] *RFC 2045 - Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies.* https://tools.ietf.org/html/rfc2045. – Aufgerufen am: 02.04.2020

[21] *RFC 2046 - Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types.* https://tools.ietf.org/html/rfc2046. – Aufgerufen am: 02.04.2020

[22] *RFC 5322 - Internet Message Format.* https://tools.ietf.org/html/rfc5322. – Aufgerufen am: 02.04.2020

[23] *Synergy Research Group: Worldwide market share of leading cloud infrastructure service providers in Q4 2019.* https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/. – Aufgerufen am: 29.02.2020

[24] *AWS re:Invent 2018: Amazon DynamoDB Under the Hood: How We Built a Hyper-Scale Database (DAT321).* https://www.youtube.com/watch?v=yvBR71D0nAQ. 2018. – Aufgerufen am: 26.04.2020

[25] Baird, Henry ; Nagy, George: A Self-Correcting 100-Font Classifier, 03 1994, S. 106–115

[26] Buneman, Peter: Semistructured Data. In: *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. New York, NY, USA : Association for Computing Machinery, 1997 (PODS '97), S. 117–121. – URL https://doi.org/10.1145/263661.263675. – ISBN 0897919106

[27] Codd, E. F.: A Relational Model of Data for Large Shared Data Banks. In: *Commun. ACM* 26 (1983), Januar, Nr. 1, S. 64–69. – URL https://doi.org/10.1145/357980.358007. – ISSN 0001-0782

[28] Cukier, Kenneth: *Data, data everywhere: A special report on managing information*. Economist Newspaper, 2010

[29] DeCandia, Giuseppe ; Hastorun, Deniz ; Jampani, Madan ; Kakulapati, Gunavardhan ; Lakshman, Avinash ; Pilchin, Alex ; Sivasubramanian, Swaminathan ; Vosshall, Peter ; Vogels, Werner: Dynamo: amazon's highly available key-value store. In: Bressoud, Thomas C. (Hrsg.) ; Kaashoek, M. F. (Hrsg.): *SOSP*, ACM, 2007, S. 205–220. – URL http://dblp.uni-trier.de/db/conf/sosp/sosp2007.html#DeCandiaHJKLPSVV07. – ISBN 978-1-59593-591-5

[30] Fowler, M. ; Foemmel, M.: *Continuous integration*. 2005

[31] Haerder, Theo ; Reuter, Andreas: Principles of Transaction-Oriented Database Recovery. In: *ACM Comput. Surv.* 15 (1983), Dezember, Nr. 4, S. 287–317. – URL https://doi.org/10.1145/289.291. – ISSN 0360-0300

[32] Hamad, Karez ; Kaya, Mehmet: A Detailed Analysis of Optical Character Recognition Technology. In: *International Journal of Applied Mathematics, Electronics and Computers* 4 (2016), 12, S. 244–244

[33] International Standard Organization (ISO): *International Standard ISO/IEC 9126, Information technology - Product Quality - Part1: Quality Model*. 2001

[34] Levenshtein, V. I.: Binary Codes Capable of Correcting Deletions, Insertions and Reversals. In: *Soviet Physics Doklady* 10 (1966), Februar, S. 707

[35] Martin, Robert C.: *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Boston, MA : Prentice Hall, 2017 (Robert C. Martin Series). – URL https://www.safaribooksonline.com/library/view/clean-architecture-a/9780134494272/. – ISBN 978-0-13-449416-6

[36] Mell, Peter ; Grance, Tim: The NIST definition of cloud computing. (2011)

[37] Mori, S. ; Suen, C. Y. ; Yamamoto, K.: Historical review of OCR research and development. In: *Proceedings of the IEEE* 80 (1992), July, Nr. 7, S. 1029–1058. – ISSN 1558-2256

[38] Morris, Kief: *Infrastructure as Code: Managing Servers in the Cloud*. 1st. O'Reilly Media, Inc., 2016. – ISBN 1491924357

[39] Smith, Ray: *The Extraction and Recognition of Text from Multimedia Document Images*. GBR, Dissertation, 1987. – AAIDX81616

[40] Smith, Ray: A Simple and Efficient Skew Detection Algorithm via Text Row Accumulation. In: *Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 2) - Volume 2*. USA : IEEE Computer Society, 1995 (ICDAR '95), S. 1145. – ISBN 0818671289

[41] Smith, Ray: An Overview of the Tesseract OCR Engine. In: *Proc. Ninth Int. Conference on Document Analysis and Recognition (ICDAR)*, 2007, S. 629–633

[42] Smith, Ray: Hybrid Page Layout Analysis via Tab-Stop Detection. In: *Proceedings of the 2009 10th International Conference on Document Analysis and Recognition*. USA : IEEE Computer Society, 2009 (ICDAR '09), S. 241–245. – URL https://doi.org/10.1109/ICDAR.2009.257. – ISBN 9780769537252

[43] Tauschek, Gustav: *Reading machine, US Patent US2026330A*. https://patents.google.com/patent/US2026330A/. 1929

[44] Warnock, J.: *The Camelot Project*. www.eprg.org/G53DOC/pdfs/warnock_camelot.pdf. – Aufgerufen am: 11.04.2020

Glossar

Amazon Web Services Amazon Web Services ist der, zum jetzigen Stand, größte Cloud-Anbieter der Welt.

Electronic Mail Format Das Electronic Mail Format ist eine Dateierweiterung für die Speicherung von E-Mail-Nachrichten.

Levenshtein-Distanz Die Levenshtein-Distanz ist ein Maß um die Ähnlichkeit zweier Zeichenketten zu quantifizieren.

Optical Character Recognition Das Portable Document Format ist ein plattformübergreifendes Dateiformat, welches von Adobe 1993 veröffentlicht wurde.

Portable Document Format Das Portable Document Format ist ein plattformübergreifendes Dateiformat, welches von Adobe 1993 veröffentlicht wurde.

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Domänenspezifische Texterkennung am Beispiel der automatischen Rechnungsverarbeitung: Konzeption und Realisierung

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original