

Dokumentation - Erschließungs- und Visualisierungswerkzeuge für den Imagekatalog der Musikabteilung der Staatsbibliothek zu Berlin

Inhaltsverzeichnis

1	Anforderungen der Applikation	2
1.1	Hardware	2
1.1.1	System 1: Deployment	2
1.1.2	System 2: Development	3
1.2	Software	3
1.3	Installation	3
1.4	Konfiguration	4
1.5	Bootstrap	5
1.6	Infrastruktur	5
2	Frontend	6
2.1	Aufbau	6
2.2	Gestaltung	7
2.3	Komponenten	7
2.4	Dashboard	7
2.5	Filterungslisten	7
2.6	Kartenübersicht	10
2.7	Korrektur	11
2.8	Jobs System	11
2.9	Vorraussetzungen	11
2.10	Redis	13
2.10.1	Resque	13
2.10.2	Datenbankadapter	14
2.10.3	Überblick	14
2.10.4	Korrekturalgorithmus	15

3	Konvertierung der Datenquellen	15
3.1	nbconv	17
3.2	labelconv	17
3.3	semconv	18
3.4	catconv	18
4	Verbesserungen	18
4.1	Accountmanagement	18
4.2	Bildscrolling und 16:9	18
4.3	Korrekturalgorithmus	19
4.4	Spracherkennung	19
4.5	Gewichtung von Treffern	19
4.6	Mehr Daten	19
4.7	Nutzen anderer Systeme zum Abgleich	20
4.8	Rückfluss von Daten aus dem Frontend	20
4.9	Modularität	20
4.10	Datenbasis und Datenverarbeitung	21
4.11	Signaturgenauigkeit	21
4.12	Schnittstellen	21
4.13	Nutzung der Daten	22
5	Meilensteine	22
5.1	OCR	22
5.2	Korrektur	22
6	Ergebnisse	23

1 Anforderungen der Applikation

Im folgenden soll kurz auf die hard- und softwareseitigen Anforderungen der Applikation eingegangen werden.

1.1 Hardware

Es wurde während der Entwicklung des Systems auf zwei Systemen folgender Spezifikationen gearbeitet:

1.1.1 System 1: Deployment

Bauteil	Beschreibung
CPU	Intel Xeon E3-1225 V2 @3.20GHz (Quad Core)

Bauteil	Beschreibung
RAM	16GB
Festspeicher	2TB HDD

1.1.2 System 2: Development

Bauteil	Beschreibung
CPU	Intel Core I7 @3.40GHz (Quad Core mit Hyper-Threading)
RAM	24GB
Festspeicher	250GB Samsung 850 EVO SSD

Beide Systeme wurden zum Import und zur Aufbereitung der Daten genutzt und liefen zufriedenstellend.

Für den Produktiveinsatz mit dem Komplettsatz der Daten wäre allerdings mehr CPU-Leistung (mehr Kerne) sinnvoll, da die Import- und Korrekturprozesse auf den beiden oben gezeigten Systemen eher lange liefen. Aufgrund der hohen IO-Last während der Import- und Korrekturschritte des Systems ist eine SSD einer normalen Festplatte vorzuziehen.

1.2 Software

Wie in einem der Meetings des zweiten Projektsemesters vereinbart, ist die Applikation sowie alle ihre Komponenten vollständig kompatibel zur Linuxdistribution Debian 8 (Jessie).

1.3 Installation

Schritte zur einfachen Installation der Applikation auf einem Debian-System als user root:

1. `wget https://raw.githubusercontent.com/albrechtsimon/htwmusic_webapp/master/puppet/bootstrap.sh`
2. `bash bootstrap.sh`
3. `git clone https://github.com/albrechtsimon/htwmusic_webapp.git`
4. `cd htwmusic_webapp/puppet`
5. `puppet apply --modulepath=./modules site.pp`

Nach diesen Schritten sollte die Applikation via `http://server_name` erreichbar sein.

1.4 Konfiguration

Die Applikation ist mittels einer Konfigurationsdatei auf das jeweilige System anpassbar. Die folgende Tabelle beschreibt die Konfigurationsvariablen:

Variable	Beschreibung
<code>acronyms_path</code>	Pfad zu einer CSV-Datei, welche Abkürzungen sowie deren Langform enthält
<code>raw_data_path</code>	Pfad zu einem Ordner, welcher Dateien in der Ordnerstruktur und im Format der OCR-Tools enthält
<code>words_path</code>	Pfad zu einem Ordner, der Text-Dateien mit einem Wort pro Zeile enthält
<code>musical_works_path</code>	Pfad zu einer CSV-Datei, die Werke enthält
<code>people_path</code>	Pfad zu einer CSV-Datei, die Personen enthält
<code>debug_output</code>	Boolean. Gibt an, ob die Applikation beim Importprozess Debuginformationen ausgeben soll oder nicht

Variable	Beschreibung
<code>data_limit</code>	Ganzzahl. Beschreibt, nach wie vielen Records der Datenimport beendet werden soll.

Die Stammdaten für die Pfade in der Konfiguration, die während der Entwicklung verwendet worden sind, sind hier auffindbar.

1.5 Bootstrap

Sobald die im vorangehenden Abschnitt vorgestellten Konfigurationsparameter gesetzt sind und die Applikation wie oben beschrieben installiert worden ist, kann ein (in diesem Falle synchroner) Import wie folgt angestoßen werden:

1. `su - htwmusic`
2. `cd htwmusic_webapp`
3. `export RAILS_ENV=production`
4. `bundle exec rake bootstrap`

Sollte ein reiner Import ohne Lauf des Korrekturalgorithmus gewünscht sein, sollte als vierter Punkt stattdessen `bundle exec rake bootstrap2` ausgeführt werden.

1.6 Infrastruktur

Die Hauptapplikation, welche auf Basis von Ruby on Rails 4 entwickelt worden ist, interagiert mit mehreren anderen Softwarekomponenten. Die Struktur während der Entwicklungsphase sowie die Deployment-Infrastruktur während der Entwicklung (kann in einem separaten Dokument betrachtet werden)[`applikationsstruktur_deployment.md`].

Nach dem zweiten Projektsemester sieht die Infrastruktur der Softwarekomponenten wie folgt aus:

Komponente	Aufgabe	Konfiguration
Nginx	Anlaufpunkt der Webappli- kation, Reverse Proxy zu Passenger	<code>/etc/nginx/</code>

Komponente	Aufgabe	Konfiguration
Passenger	Applikationsserver	Commandline-Argumente, änderbar hier
Ruby on Rails	Applikationsframework	Konfigurationsdateien. Zu finden hier
ElasticSearch	Volltextindizes, Wortkorrektur	Das System nutzt aktuell die default Konfiguration von ElasticSearch. Falls notwendig ist diese anpassbar in <code>/etc/elasticsearch/elasticsearch.yml</code>
PostgreSQL	Persistenzschicht der Rails-Applikation	PostgreSQL nutzt seine Standardkonfiguration. Anpassungen möglich in <code>/etc/postgresql/\$version/main/</code>
Redis	Speicherort für Queues	Redis nutzt ebenfalls die Standardkonfiguration. Anpassungen möglich in <code>/etc/redis</code>

2 Frontend

2.1 Aufbau

Die Basis für das Frontend bildet Rails. Rails dient einerseits als Backend und stellt mit der verwendeten Templateengine ERB alle Ansichten dar. ERB umfasst eine Template-syntax, welche durch viele Funktionen und Modulen die Entwicklung vereinfacht. Für den Nutzer werden die entsprechenden Daten aufbereitet und abschließend ausgeliefert. Hierbei sind weitere Technologien eingebunden wie u.a. Bootstrap.

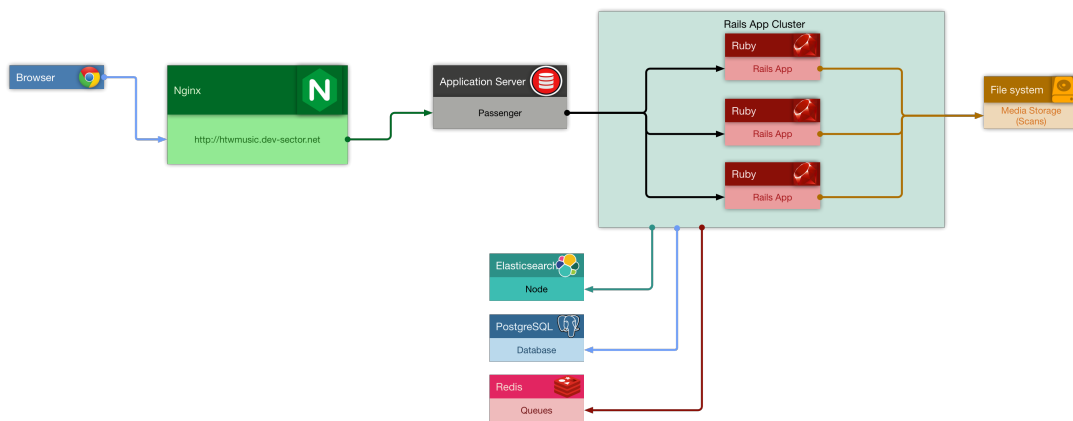


Abbildung 1: Infrastructure

2.2 Gestaltung

Die Gestaltung des Frontend orientiert sich einerseits an der Farbgebung der Webseite der Staatsbibliothek zu Berlin, andererseits an einer einfachen Tasterturgebundenen Bedienung.

2.3 Komponenten

Das Frontend setzt sich aus drei Komponenten zusammen, die im Nachfolgenden Abschnitt näher erläutert werden.

2.4 Dashboard

Das Dashboard bildet den Einstieg in die Applikation, in diesem werden alle Kartenzustände grafisch dargestellt.

1. Schnellauswahl für die Filterlisten
2. Grafische Darstellung der Karten Status
3. Übersicht der Karten

2.5 Filterungslisten

In der Listenübersicht kann nach Inhalten von Karten gesucht werden, um thematisch zusammenhängende Karten zu finden.

1. Suchen nach Kartennummern

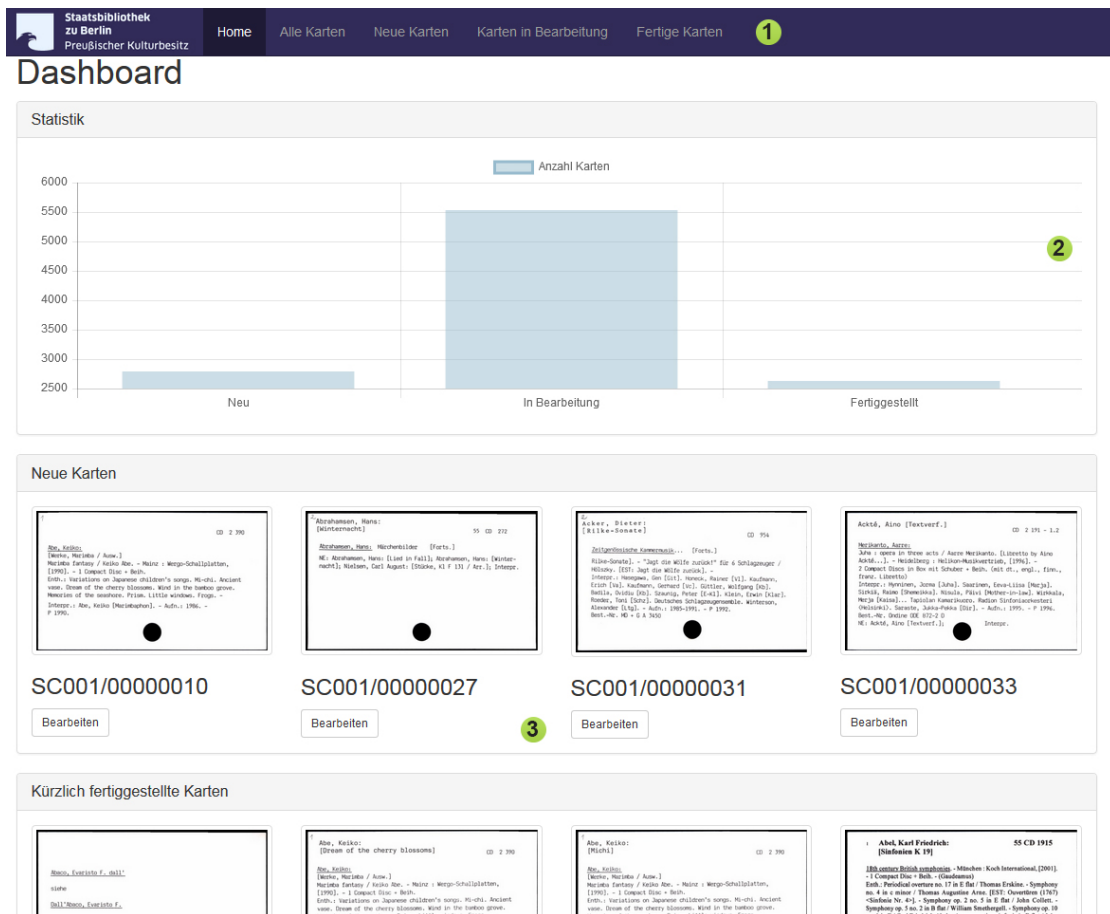


Abbildung 2: alt text

Karten (Alle)

Suche nach Karten-ID

Suchen

✕

1

3

Suche nach Worten

Suchen in OCR Text

✕

2









Thumbnail	Karten-ID	Aktueller Status	Actions
	SC001/00000001	In Bearbeitung	<div>Karte anzeigen</div> <div>Karte bearbeiten</div>
	SC001/00000002	In Bearbeitung	<div>Karte anzeigen</div> <div>Karte bearbeiten</div>
	SC001/00000003	In Bearbeitung	<div>Karte anzeigen</div> <div>Karte bearbeiten</div>
	SC001/00000004	In Bearbeitung	<div>Karte anzeigen</div> <div>Karte bearbeiten</div>
	SC001/00000005	Fertiggestellt	<div>Karte anzeigen</div> <div>Karte bearbeiten</div>
	SC001/00000006	Fertiggestellt	<div>Karte anzeigen</div> <div>Karte bearbeiten</div>
	SC001/00000007	In Bearbeitung	<div>Karte anzeigen</div> <div>Karte bearbeiten</div>
	SC001/00000008	Fertiggestellt	<div>Karte anzeigen</div> <div>Karte bearbeiten</div>

Abbildung 3: alt text

2. Suche für die OCR-Texte
3. Kartenvorschau
4. Kartenummer
5. Status der Karte

2.6 Kartenübersicht

Die Kartenübersicht bildet alle gefundenen und verarbeiteten Daten ab. Diese werden in zwei verschiedenen Ansichten präsentiert, um die Übersicht zu gewährleisten. ###Felderübersicht Hier werden alle wichtigen Kartendaten angezeigt, dazu gehören die Karte selbst, sowie der erkannte OCR-Text.

Staatsbibliothek zu Berlin
Preußischer Kulturbesitz

Home Alle Karten Neue Karten Karten in Bearbeitung Fertige Karten

Formular **Korrekturen** 1

Aktueller Status: In Bearbeitung 2

Aargauer Komponisten [Forts.]

P 1993.

Best.-Nr. Jecklin J 297.2

5s C0 348

NE s Mie, Peter: [Konzerte, Vc Kl Orch (1983)]; Wehrli, Werner: [Sinfonietten op. 20]; Blum, Robert: [Sinfonien Nr. 8]; Interpr.

55 CD 368

Aargauer Komponisten [Forts.]

P 1993.

Best.-Nr. Jecklin JS 297-2

NE: Mie, Peter: [Konzerte, Vc Kl Orch (1983)]; Wehrli, Werner: [Sinfonietten op. 20]; Blum, Robert: [Sinfonien Nr. 8]; Interpr.

Gelesener Text

Werner Wehrli

Peter Mie

Korrektur

Werner Wehrli

Peter Mie

Feld-Typ

Verfasserangabe

Verfasserangabe

Status

Warnung

Warnung

Best.-Nr. Jecklin J 297.2

5s C0 348

NE s , : [Konzerte, Vc Kl Orch (1983)]; . : [Sinfonietten op. 20]; . : [Sinfonien Nr. 8]; Interpr.

Neues Feld

Speichern

abschließen und weiter

Abbildung 4: alt text

1. Auswahl der derzeitigen Übersicht
2. Status der derzeitigen Karte

3. Aus der OCR erfasster Text.
4. Bild der Karte
5. Der aus der OCR gelesene Textabschnitt
6. Gefundene Korrektur
7. Typ aus Katalogübersicht
8. Status des Feldes
9. Speichern oder Löschen von Feldern
10. Erzeugen eines neuen Feldes
11. Speichern der derzeitigen Änderungen
12. Speichern der derzeitigen Änderungen und Versiegeln der Karte

2.7 Korrektur

Der Korrekturtab zeigt alle Änderungen die durch die Korrektur automatisch verarbeitet wurden. Hier ist ersichtlich aus welchen Verzeichnissen die gefundenen Daten erzeugt wurden und weshalb die Änderungen übernommen wurden.

1. Karte
2. unkorrigierter OCR-Text
3. Text nach der Korrektur
4. Teilstück das als Korrekturvorlage diente
5. Korrektur des Teilstückes
6. Basis, auf dessen Grundlage das Teilstück korrigiert wurde.

2.8 Jobs System

Aufgrund der Anzahl der zu bearbeiteten Karten war es sinnvoll ein Job System einzuführen, welches in Teilen eingesetzt wurde. Die Grundlage dafür bildet Resque, ein Queue System von GitHub <https://github.com/resque/resque>. Die nötigen Bestandteile von Resque können durch entsprechende Gems in Rails integriert werden und sind so in der Lage auf entsprechende Ressourcen zuzugreifen. Hierfür werden Jobs entsprechend der Definition erzeugt und können dann von Workern abgearbeitet werden. Die hierfür nötigen Informationen, werden in Redis abgelegt.

Link: Tabelle zur Beschreibung der Jobs

2.9 Voraussetzungen

Für die Nutzung der Jobs sind folgende Bestandteile notwendig

Staatsbibliothek zu Berlin

Preußischer Kulturbesitz

Home

Alle Karten

Neue Karten

Karten in Bearbeitung

Fertige Karten

Formular

Korrekturen

Korrekturen

2

Abrahamsen, Hans:
[Winternacht]

55 CD 272

Abrahamsen, Hans: Märchenbilder [Forts.]

NE: Abrahamsen, Hans: [Lied in Fall]; Abrahamsen, Hans: [Winter-
nacht]; Nielsen, Carl August: [Stücke, Kl F 131 / Arr.]; Interpr.

1

Rohtext:

2

TAbrahamsen, Hans [Winternacht] Abrahamsei Hans: Märcheetiilder [Farts.] 55CO 212 NE : Anrahamsen, Hans: [Lied in Fall]: Abrahamsei, Hans: [Winter- nacht]: Nielsen, Carl August: [Stücke, Kl F In1 7 Arr.]: Interpret

Text mit Korrekturen:

3

Abrahamsen Hans Winternacht Abrahamsen Hans märcheetiilder Farts 55,1 212 NE Abrahamsen Hans lied in Fall Abrahamsen Hans Winter nacht Nielsen Carl Aust Stücke kl F und 7 Arr Interpret

#	Von 4	Nach 5	Grund 6
282	TAbrahamsen	Abrahamsen	Abrahamsen found 16 times. matching exactly with: person index
283	Abrahamsei	Abrahamsen	Abrahamsen found 16 times. matching exactly with: person index
284	Märcheetiilder	märcheetiilder	CARD INDEX CORRECTION: Märcheetiilder => märcheetiilder [score: 0.9285714, freq: 3]
285	55CO	55,1	MUSICAL WORK CORRECTION: 55CO => 55,1 [score: 0.5, freq: 3]
286	Anrahamsen	Abrahamsen	Abrahamsen found 16 times. matching exactly with: person index
287	ILied	lied	lied found 7 times. matching exactly with: musical work index, word index klier found 1 times. matching exactly with: person index

Abbildung 5: alt text

2.10 Redis

Redis ist eine in-memory Datenstruktur die auf einem einfachen Key-Value Cache basiert. In diesem werden die Jobs mit ihren Parametern als JSON gespeichert. Als Beispiel kann folgender JSON String dienen:

```
{'class': ExternalInterpreterLookup, 'args': [ 1 , 'normal' ] }
```

Die Klasse beschreibt den Job der durch einen Workern durch den Invoke Process erzeugt wird.

Die Argumente bestimmen die Parameter der `perform()` Methode, die jeder Job anbieten muss.

Zu beachten ist, dass nur Parameter abgelegt werden können, die in einen JSON umgewandelt werden können. Es ist deshalb notwendig z.B. auf Objekt IDs zurückzugreifen, anstatt auf direkte Objektreferenzen bei der Übergabe an

```
Resque.enqueue(ExternalInterpreterLookup, 1, normal)
```

Näheres dazu in der Dokumentation von Resque und Redis <https://github.com/resque/resque>

2.10.1 Resque

Resque ist ein Job System, welches die Bestandteile von Redis nutzt, um Jobs zu erzeugen und auszuführen. Jobs können generell alle Klassen sein die eine `perform()` Methode anbieten, es empfiehlt sich diese jedoch separat zu strukturieren. Jobs im Projekt liegen in `lib/processing/jobs` Resque bietet mit Workern dann die Basis, Workern erzeugen die entsprechenden Klassen, die ihnen durch Redis übergeben werden und führen die Methode `perform()` auf diesen aus. Worker selbst sind Sub-Prozesse die über

```
COUNT=5 QUEUE=* rake resque:workers
```

gestartet werden können. Hierbei bestimmt Count die Anzahl der Worker. Im Projekt zeigte sich, dass eine Anzahl von Workern, die die maximale Kernzahl übersteigt, keinen Performancegewinn erzeugt. Teilweise stürzte die gesamte Workerstruktur dadurch ab, Erklärungen gab es dafür keine und es ist möglich das sich das nicht reproduzieren lässt, es ist daher lediglich als Anmerkung zu sehen.

Für die Verwaltung bietet resque gleich eine Schnittstelle im Frontend mit diese lässt sich mit

```
/resque_web
```

aufrufen und bietet eine Übersicht, die zumeist für das debuggen ausreichte. Näheres dazu in der Dokumentation von Resque.

2.10.2 Datenbankadapter

Es ist notwendig das eine persistente Datenbank zur Verfügung steht, diese muss in Rails definiert sein, da Resque im default diese für seine Jobs benutzt. In der Entwicklung wurden MySql und Postgress verwendet. Theoretisch sind auch andere denkbar.

2.10.3 Überblick

Es existieren derzeit folgende Jobs im Projekt:

2.10.3.1 CardFactory

Da die Notwendigkeit besteht, alle Karten in den übergebenen Jsons zu Objekte zu überführen, werden hier Karten stückweise angelegt. Hierfür werden die Json-Rohdaten in Teile zerlegt und dann auf mehrere Worker verteilt.

2.10.3.2 ExternalWorkLookup

Hier wird das Auflösen von Werken, aus dem Werksverzeichnis durchgeführt. Durch die Korrektur ist davon auszugehen das Werke bereits aufgelöst wurden. Deshalb reicht ein einfaches Matching.

2.10.3.3 ExternalInterpreterLookup

Hier wird das Auflösen von Interpreten, aus dem Interpretenverzeichnis durchgeführt. Für jede Person im Verzeichnis wird eine Suche ausgeführt, dafür werden zuerst volle Namen aus dem Verzeichnis im Text gesucht, dies wäre der Idealfall, sollte die Korrektur gut gearbeitet haben. Sollte kein Treffer gelandet werden, wird im Text nach Vor- und Nachnamen gesucht. Dies ist ungenau und kann zu Fehlern führen, umgeht aber Schreibweisen und Einrückungen.

2.10.3.4 SignatureLookup

Hier wird das Auflösen von Signaturen, aus dem Signaturenverzeichnis durchgeführt. Diese werden auf Basis von RegEx begriffen aufgelöst. Dies geschieht Zeilenweise von oben nach unten. Aufgrund der OCR ist die Zeilennummer nicht ausschlaggebend für die Position der Signaturen. Für die Suche wird der Text von Leerzeichen befreit und in Großbuchstaben umgewandelt, dies vereinfacht die Erkennung. Die RegEx-Begriffe werden aus der *signatures.txt* geladen und enthalten folgende angaben Bezeichnung nach Katalog Die Beispielzusammensetzung für Signaturen dieses Typs Beschreibung Zuweisung für eine Kategorie, bzw. Beschreibung des Katalogs RegEx Der RegEx-Begriff für die Suche Replacement Sollte es möglich sein eine komplexe Signatur durch Platzhalter aufzulösen, dient das Replacement für die Korrekturangabe nach dem Match.

2.10.3.5 FieldReplacer

Hier werden Felder nach gleichen Einträgen durchsucht, die dann im Anschluss auf einen neuen Typ korrigiert werden. JobBuilder Jobs werden durch eine Nummer klassifiziert. Jede Karte besitzt dafür den zuletzt durchgeführten Job als Referenz. Der JobBuilder nutzt diese Identifikation um den nächsten Job für die Karte zu erzeugen. Dies definiert sich durch die Reihenfolge die in JOBS::JOBS definiert ist.

2.10.3.6 JobCreator

Der JobCreator ist eine Hilfsklasse die den JobBuilder erzeugt, nachdem alle Parameter gesetzt wurden. Dies war nötig um möglichst Modular zu agieren und einen Job ohne Parameter zu besitzen.

2.10.3.7 AcronymReplacer

Um die Verarbeitung einfacher zu gestalten, sollen Synonyme ersetzt werden, so dass diese einheitlich vorliegen. Da die Integration mit der Elastic Search Korrektur nicht vollständig Modular aufgebaut wurde, wird dieser Schritt im Vorbereitungsprozess durchgeführt, sollte die Modularität gewährleistet sein, sollte die Funktionalität hierher ausgelagert werden.

2.10.3.8 DataCrawler

Als Erweiterung gedacht, jedoch bisher nicht weiter verfolgt. Sollte es notwendig sein Karten oder Daten nachträglich hinzuzufügen, sollte dies hier implementiert werden. Derzeitig wurde hier nur das Job verhalten der Worker getestet.

2.10.4 Korrekturalgorithmus

Da die Texterkennung fehlerbehaftet ist, sollte ein Algorithmus entworfen werden, welcher die Fehler der OCR ausbessern sollte. Die Arbeitsweise des Algorithmus ist in folgender Grafik skizziert:

Die Implementierung des Algorithmus kann im Card-Model der Railsapplikation betrachtet werden.

3 Konvertierung der Datenquellen

Bis jetzt nutzt das Projekt 4 Datenquellen:

- OPAC-Bilder
- OPAC-Indexierung

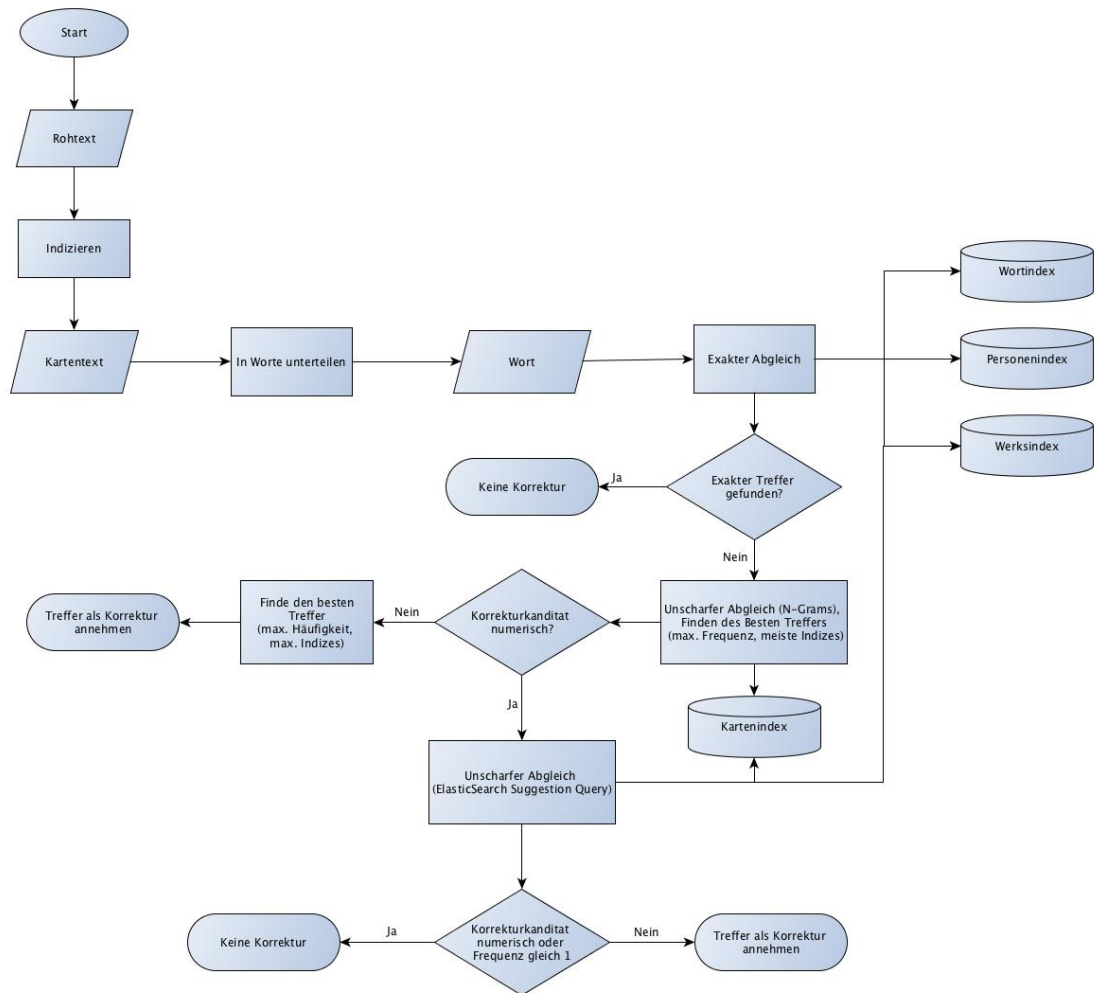


Abbildung 6: Korrekturalgorithmus

- GND
- Labeldaten

Im folgenden werden die Tools für die Verarbeitung der Daten dokumentiert.

3.1 nbconv

nbconv ist eine Sammlung von iPython-Notebooks. Die Notebooks enthalten Untersuchungen über die Daten und unterschiedliche Konvertierungsskript und API-Anleitungen. Die Datei `ocr_error_rates.ipynb` erzeugt die Graphen für das IMI-Showtime-Poster. Das Skript `index-database2json.ipynb` exportiert die Indexierungs-Daten als JSON-Dateien. Diese Schritte müssen im dauerhaften betrieb der Anwendung nicht mehr ausgeführt werden.

3.2 labelconv

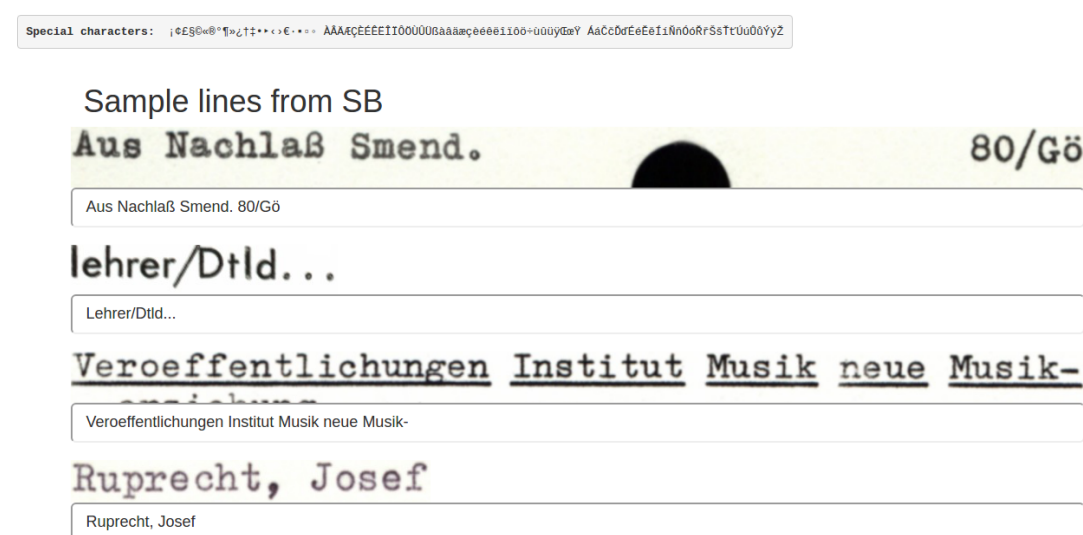


Abbildung 7: labelconv_screenshot

labelconv ist ein simples Tool um Trainingsdaten für die OCR zu erstellen. Im Gegensatz zu den hOCR-Tools von ocrpy bietet dieses Tool Features um besser mit den Katalogdatensatz zu arbeiten. Es kann ein Seed für die zufällige Auswahl von Textzeilen festgelegt werden. Danach wird im Browser ein Sample aus dem jeweiligen Katalog gezeigt. So können unabhängige Samples zum trainieren und zum testen erstellt werden.

3.3 semconv

Dieses Skript dient der Datensammlung zur Verbesserung des Erschliessungsprozesses. Die Extrahierung der Daten aus der GND ist ein größeres Thema als gedacht und dieses Skript liefert lediglich einen Ausgangspunkt.

3.4 catconv

Das Modul ist zuständig für die Konvertierung der Katalogkarten. Es werden lediglich Skripte bereitgestellt, dabei orientiert sich am Aufbau von Ocropy (kein Zustand, kleine Programme mit nur einer Aufgabe).

- convert.py: Umwandlung von TIF zu jpgs.
- process.py: Vorprozessierung//Binarisierung
- export.py: Umwandlung ins Exportformat

Die iPython-Notebooks dokumentieren das Trainings eines OCR-Modells, welches fuer das Skript process.py benoetigt wird.

4 Verbesserungen

4.1 Accountmanagement

Das Frontend umfasst kein Accountmanagement. Auch wenn dies nicht genutzt werden soll, um einzelne Nutzer zu überwachen, wäre jedoch die Umsetzung nicht sonderlich kompliziert und würde die Anwendung für weitere Szenarien öffnen, beispielsweise Fremdleistungen.

4.2 Bildscrolling und 16:9

Für die Ansicht wurde sich gewünscht das die Karte beim Scrollen durch die Felder immer sichtbar sein sollte. Dies wurde insofern abgeschwächt, das nun der restliche unbearbeitete Text unter den Feldern sichtbar ist. Hier müsste eruiert werden, ob und inwiefern die Umsetzung vom Bildmitlauf noch gewünscht wäre. Zudem wäre ein Breiteres Design möglich, da das derzeitige noch etwas Freiraum an den Seiten bietet. Es gab jedoch kein uns bekanntes Szenario das einen anderen Aufbau gerechtfertigt hätte. Hier könnten alternative entworfen werden.

4.3 Korrekturalgorithmus

Gerade der Korrekturalgorithmus ist einer der Punkte, an dem in der Fortführung des Projektes weitergearbeitet werden sollte. Mögliche Punkte, die hierbei bearbeitet werden könnten sind im Folgenden aufgeführt:

4.4 Spracherkennung

Die aktuelle Version des Algorithmus versucht unter anderem, wie im obigen Diagramm ersichtlich, Korrekturen für Worte anhand eines Wortindex zu finden. Dieser Index ist momentan nicht sprachabhängig implementiert. Das heißt, dass Worte aus unterschiedlichen Sprachen in einem Index liegen.

Man könnte nun versuchen, diesen Index sprachspezifisch aufzusplitten, sodass Worte einer Sprache jeweils in ihrem eigenen Index liegen. Gegeben den Fall, es würde eine Methode gefunden, mit welcher ermittelt werden kann, in welcher Sprache der Text einer Karte verfasst ist oder auch welcher Sprache einzelne Wörter im OCR-Text angehören, könnte die Fehlerrate des Algorithmus gesenkt werden.

4.5 Gewichtung von Treffern

Die Implementierung des Wählens von Treffern einer Query an ElasticSearch ist momentan relativ primitiv. Werden Worte, die zur Korrektur herangezogen werden sollen, gleich oft gefunden, entscheidet sich der Algorithmus immer für den ersten Treffer. Durch dieses Vorgehen entstehen falsche Ersetzungen.

Hier könnte versucht werden, weitere Metriken in den Algorithmus einfließen zu lassen, welche eine präzisere Auswahl einer korrekten Ersetzung ermöglichen. Der Punkt der Gewichtung könnte von der im vorangehenden Abschnitt angesprochenen Spracherkennung ebenfalls zugutekommen.

4.6 Mehr Daten

Während der Durchführung des Projektes sowie der Entwicklung des Algorithmus wurde auf einem relativ kleinen Datenset gearbeitet. Die Resultate, die das Verfahren zeigt, lassen darauf schließen, dass der Ansatz in die richtige Richtung geht. Dem Gesetz der großen Zahlen entsprechend wird es sicherlich so sein, dass die generelle Korrektheit der Ersetzungen mit dem Erhöhen der Grunddatenmenge steigen wird. Grunddatenmenge bezeichnet in diesem Kontext alles an textuellen Daten, die der Algorithmus zum Abgleich heranzieht:

- einzelne Worte
- Werke

- Personen
- alle OCR Rohtexte

4.7 Nutzen anderer Systeme zum Abgleich

Ein Punkt, der im Verlauf des Projektes nicht mehr adressiert werden konnte, war der Abgleich der einzelnen der OCR entstammenden Worte Daten anderer Systeme. Würde man annehmen, dass die Drittquellendaten der Wahrheit entsprechen, könnte man mit den Daten dieser Quellen

1. verhindern, dass bereits korrekte Worte durch Falsches ersetzt werden
2. weitere Abgleiche anstellen, die beispielsweise in die Gewichtung der Korrekturkandidaten hineinspielen könnten.

4.8 Rückfluss von Daten aus dem Frontend

Sollte das System in den Produktiveinsatz übergehen, wäre es sinnvoll, alle Korrekturen, die von Anwendern des Systems vorgenommen werden, zu speichern und als wahr und valide aufzufassen. Diese manuell eingegebenen Daten sollten wiederum in einen neuen Index in der Applikation einfließen, welcher bei der Bewertung von Treffern im Korrekturalgorithmus ein wesentlich höheres Gewicht als (vermutlich alle) anderen Indizes haben sollte. Dies würde zum einen die Fehlerrate des Systems verringern und zum anderen mittel- bis langfristig die Datenmenge auf die der Algorithmus mit Vertrauen zurückgreifen kann erhöhen und deren Qualität verbessern.

4.9 Modularität

Aufgrund von Problemen beim Import und der Verarbeitung der Karten, läuft der Korrekturprozess nicht Modular. Hier konnte keine kurzfristige und zufriedenstellende Lösung gefunden werden, Callbacks zu integrieren, so das Jobs anderen Jobs melden können, wenn diese durchlaufen wurden. Sollte dies umgesetzt sein, kann die Korrektur auch in einen Job ausgelagert werden. Dies gilt dann auch für das Replacement, erst dann wäre das Job System völlig asynchron. Derzeit sind nur nachgelagerte und vorgelagerte Prozesse in Jobs möglich.

Zudem muss für den JobCreator derzeit ein Subprozess generiert werden, der in einem Interval neue JobCreator-Jobs erzeugt, damit dieser wiederum neue nachfolgende Jobs erzeugt. Evtl wäre es möglich hier einen besseren Ansatz zu finden.

4.10 Datenbasis und Datenverarbeitung

Die Datenbasis bildet einerseits die OCR, die sicherlich weiter optimiert werden kann, um die Genauigkeit zu erhöhen, andererseits werden die Werke etc. aus einer Extraktion aus der GND gespeist. Diese stellte sich jedoch als teilweise unzureichend heraus, da viele Daten unberücksichtigt sind. Daraus resultiert, dass viele Karten ungenügend aufgelöst werden. Könnten hier mehr Daten angereichert werden, wäre eine Verbesserung, einerseits der Korrektur, andererseits der Auflösung möglich.

In Zukunft sollten zur Datenverarbeitung stärker auf leistungsfähigere Frameworks gesetzt werden. Aufgrund der Datenmenge würde eine Verteilung der Prozessierung auf mehrere Rechner mit einem Framework wie Spark Vorteile bringen. Eine Umstellung auf ein Machine Learning framework würde es möglich machen GPU's zur Erkennung der Textzeilen zu nutzen. Ocropy nutzt bis jetzt "nur" eine CPU-Implementierung für die OCR. Eine Eigenentwicklung wird aber nicht mehr nötig sein sobald die Erkennung vom transkriptorium-Projekt übernommen wird.

4.11 Signaturgenauigkeit

Derzeit sind Signaturen durch RegEx-Begriffe ausgezeichnet. Diese sind für Treffer sehr genau, zeigten jedoch, dass es durchaus Gemeinsamkeiten in der fehlerhaften Erkennung von Zeichen im OCR-Text gibt. So werden oft "S" oder "5" als jeweilige Partner vertauscht. Beispiel: "55 CD 131543" wird zu "5S CD 131S43" Durch diesen Umstand ist es schwierig korrekte Signaturen umzusetzen, dies liegt vor allem daran, dass Signaturen teilweise mehrdeutige Schreibweisen besitzen, so existieren 55 CD XX, sowie CD XX. Eine Fehlerkennung ist deshalb nicht ausgeschlossen, wenn die vorangestellten Zeichen nicht richtig durch die OCR erkannt wurden. eindeutige Signaturen sind derzeit sehr zuverlässig und können durch die Ersetzungsregeln auch dann gefunden werden, wenn diese Fehlerbuchstaben enthalten. So lassen sich Signaturen wie DMS XX oder NUS XX leicht erkennen und lieferten im Test sehr gute Ergebnisse. Probleme bereiten vor allem sehr uneindeutige Signaturen wie CD oder sehr lange Signaturen, da die Fehlerrate mit der Länge der Signaturen zunimmt. Aufgrund dieser Tatsache, wäre es Ratsam, die Signaturen aus den Kartenbildern direkt zu extrahieren und so zumindest die Datenbasis, die derzeit beim ganzen OCR-Text liegt zu begrenzen. Mit unseren Mitteln, war dies jedoch bisher nicht möglich. Dies würde zumindest die Fehlerrate senken, die durch verdrehte Buchstabenkombinationen im Quelltext aufkommen.

4.12 Schnittstellen

Derzeitig liegen alle Daten die erfasst wurden nur im System selbst vor, dies umfasst die Datenbank, den Elasticsearch und die Kartenbilder selbst. Der Hauptaugenmerk lag in der allg. Verfügbarkeit und Machbarkeitsstudie des Projektes. Trotz Fehlerraten und

Misserkennung wäre aber bereits eine Nutzung denkbar. Hierfür müssten Schnittstellen definiert werden über die diese Daten abgerufen werden könnten.

4.13 Nutzung der Daten

Evtl. wären auch weitere Nutzungsszenarien denkbar, die bisher nicht in Erwägung gezogen wurden, da schlicht keine Daten außer den Bildern zur Verfügung standen. So wäre eine experimentelle Suche möglich die die Ergebnisse auf Wunsch dem allg. Datenbestand hinzufügt, um diese für die Öffentlichkeit zugänglich zu machen.

5 Meilensteine

Als agiles Projekt konzipiert und durchgeführt, änderten sich Teilaspekte durch anhaltende Wöchentliche Meetings, jedoch blieben die Ziele im allg. gleich.

So definiert sich als Hauptziel, die Entwicklung eines Systems zur Verarbeitung von Kartenbildern und Überführung in ein System mit anschließender Fehlerkorrektur, Datenaufbereitung/Visualisierung, sowie evtl. Datenextraktion. Daraus definieren sich Meilensteine die im Wochenplan festgehalten wurden. Diese unterscheiden sich je nach Anwendungsgebiet. Der größte Meilenstein des Projektes, war klar die Präsentation auf der Messe der HTW, diese war ein voller Erfolg, so dass das Projekt als solches als Erfolg verbucht werden kann, auch wenn nicht alle Meilensteine gänzlich erfüllt wurden. Die Meilensteine definieren sich aus dem aktuellsten Wochenplan, der für das Projekt angelegt wurde.

5.1 OCR

Die grundlegende Aufgabe der Verbesserung und Automatisierung der OCR konnte konsequent bearbeitet. Im Verlauf der Arbeit wurde klar das die Fehlerraten, die in vielen Publikationen berichtet werden, nicht in realen Datensets erreicht werden können. Es wurden neben der OCR auch an weiteren Methoden zur Datenextrahierung gearbeitet wie zum Beispiel die Erkennung der Kartensprache und des Kartentyps. Jedoch kann man diese Daten nur weiterverarbeiten, wenn man auch die Genauigkeit validiert, ein Integration in die Pipeline vornimmt und ein Nutzungsformat spezifiziert.

5.2 Korrektur

Im Großen und Ganzen wurden alle Meilensteine abgearbeitet und umgesetzt. Dazu gehört das entwickeln eines Systems inkl. ElasticSearch Integration, der Prozessmodellierung und Abarbeitung, Entwicklung eines Persistenzmodels, Anlegen eines Jobsystems für asynchrone Tasks, Korrektur von Ergebnissen auf Basis von GND Daten. Aufgrund

der Veränderung des Prozesses, wurden GND Daten nicht mehr Live über Pazpar2 abgerufen, sondern extern extrahiert. Dies schließt Daten externer Anbieter derzeit aus, da dafür keine Daten erhoben wurden, in diesem Punkt kann nur eine Teilweise Erfüllung angesehen werden. Die Technik dahinter ist jedoch vorhanden, so dass einfach weiteren Daten importiert werden können, wenn diese erhoben wurden. Durch Zeitmangel vor allem am Ende des Projektes, wurde die Accountverwaltung nicht implementiert, diese war zwar immer optimal, war aber theoretisch im ersten Plan festgehalten. Der Export von Daten war theoretisch vorgesehen, wurde jedoch nicht umgesetzt, dies würde neu kalkuliert werden müssen.

6 Ergebnisse

Für das Masterprojekt 1 und 2 wurde ein System entwickelt, das aufzeigt, dass eine automatisierte Erfassung technisch möglich ist. So wurden Bilder in Texte umgewandelt und anschließend versucht, diese zu korrigieren, um abschließend Daten aus diesen gewinnen zu können. Die Ergebnisse sind sehr stark abhängig vom gewählten Katalog und der Qualität desselben. So spielte auch die Sprache eine große Rolle, da die OCR ursprünglich auf Englischem Text trainiert wurde. Dies führt zu einer verschobenen Spracherkennung. Dies kann mit mehr validen Trainingsdaten umgangen werden, wofür jedoch eine händische Erfassung notwendig wäre. Dies wäre deshalb voraussichtlich ein Problem an Ressourcen. Mit Verbesserung der OCR, ist davon auszugehen dass sich das Ergebnis aller anderen Schritte anhebt. Da dies jedoch nicht zu 100% erreicht werden kann, produziert die Korrektur bereits zufriedenstellende Ergebnisse. Diese ist jedoch abhängig von den bereits erhobenen Daten. So zeigte sich, dass vor allem die Inhalte der Karten selbst ein Problem darstellen, da diese zwar logisch strukturiert sind, jedoch nicht ausreichend klare Daten beinhalten. Da dies die letzte Instanz in der Kette vor der Nutzerinteraktion darstellt, sind hier die Korrekturen maßgeblich, dazu gehören natürlich auch die Fehlerraten die durch die Korrektur selbst erzeugt werden. Die abschließende Extraktion stützt sich dann auf die korrekte Korrektur, weshalb dort Misserkennung das größte Problem darstellt. Sollten Bestandteile weder durch die OCR noch durch die Korrektur genau ermittelt worden sein, ist es sehr schwierig Daten zu erheben. Dies zeigte sich maßgeblich an den Signaturen, die je nach Katalogart sehr durchwachsende oder sehr gute Ergebnisse lieferten. Maßgeblich ist hier auch die Datenbasis die zugrunde liegt, Daten die nicht existieren, können nicht gefunden werden, weshalb eine weitere Datenspeisung ratsam ist. Trotz der angesprochenen Probleme, zeigt das Projekt, dass es möglich ist, maschinell Daten zu erheben. Diese entsprechen jedoch voraussichtlich nicht den Anforderungen die ein Bibliothekar an seine Sammlung stellt. Hierbei ist jedoch zu betrachten, dass Daten die nicht existieren, nicht gefunden werden können und so das in Kauf nehmen von Fehlerraten besser wäre, als 100%tige Ergebnisse. Jedoch ist eine Fehlerrate gegen Null immer anzustreben, weshalb eine weitere Beschäftigung mit dem Projekt ratsam ist.