HTL - Perg
Höhere Abteilung für Informatik

Diplomarbeit

OCR-Post-Processing

Projektteam: Simon Haiden
Andreas Heindl
Lukas Hofer
Julian Primetshofer

Projektbetreuer: Prof. Mag. Michael Holzmann

In Zusammenarbeit mit Fabasoft AG

Betreuer Herr Franz Deimling

Bearbeitungszeitraum: 10.07.2023 – 04.04.2024

1. Eidesstattliche Erklärung

Hiermit versichern wir, die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der von uns angegebenen Quellen angefertigt zu haben. Alle Stellen, die wörtlich oder sinngemäß aus fremden Quellen direkt oder indirekt übernommen wurden, sind als solche gekennzeichnet.

Perg, 3.4.2024 Unterschrift Simon Haiden

(Simon Haiden)

Signiert von: Simon Haiden
Datum: 03.04.2024 19:09:45
A-TRUST
Dieses Dokument ist digital signiert!
Dieses mit einer qualifizierten elektronischen Signatur versehene Dokument hat gemäß Art. 25 Abs. 2 der Verordnung (EU) Nr 910/2014 vom 23. Juli 2014 ("eIDAS-VO") die gleiche Rechtswirkung wie ein handschriftlich unterschriebenes Dokument.
Prüfinformation: Informationen zur Prüfung der elektronischen Signatur finden Sie unter: www.a-trust.at/pdf

Perg, 3.4.2024 Unterschrift Andreas Heindl

(Andreas Heindl)

Signiert von: Andreas Heindl
Datum: 03.04.2024 19:12:16
A-TRUST
Dieses Dokument ist digital signiert!
Dieses mit einer qualifizierten elektronischen Signatur versehene Dokument hat gemäß Art. 25 Abs. 2 der Verordnung (EU) Nr 910/2014 vom 23. Juli 2014 ("eIDAS-VO") die gleiche Rechtswirkung wie ein handschriftlich unterschriebenes Dokument.
Prüfinformation: Informationen zur Prüfung der elektronischen Signatur finden Sie unter: www.a-trust.at/pdf

Perg, 3.4.2024 Unterschrift Lukas Hofer

(Lukas Hofer)

Signiert von: Lukas Hofer
Datum: 03.04.2024 19:07:46
A-TRUST
Dieses Dokument ist digital signiert!
Dieses mit einer qualifizierten elektronischen Signatur versehene Dokument hat gemäß Art. 25 Abs. 2 der Verordnung (EU) Nr 910/2014 vom 23. Juli 2014 ("eIDAS-VO") die gleiche Rechtswirkung wie ein handschriftlich unterschriebenes Dokument.
Prüfinformation: Informationen zur Prüfung der elektronischen Signatur finden Sie unter: www.a-trust.at/pdf

Perg, 3.4.2024 Unterschrift Julian Primetshofer

(Julian Primetshofer)

Signiert von: Julian Primetshofer
Datum: 03.04.2024 19:03:34
A-TRUST
Dieses Dokument ist digital signiert!
Dieses mit einer qualifizierten elektronischen Signatur versehene Dokument hat gemäß Art. 25 Abs. 2 der Verordnung (EU) Nr 910/2014 vom 23. Juli 2014 ("eIDAS-VO") die gleiche Rechtswirkung wie ein handschriftlich unterschriebenes Dokument.
Prüfinformation: Informationen zur Prüfung der elektronischen Signatur finden Sie unter: www.a-trust.at/pdf

2. Gendererklärung

Die in dieser Diplomarbeit verwendeten Personenbezeichnungen beziehen sich immer gleichermaßen auf weibliche und männliche Personen. Auf eine Doppelnennung und gegenderte Bezeichnungen wird zugunsten einer besseren Lesbarkeit verzichtet.

Perg, 3.4.2024 Unterschrift Simon Haiden

(Simon Haiden)

Perg, 3.4.2024 Unterschrift Andreas Heindl

(Andreas Heindl)

Perg, 3.4.2024 Unterschrift

(Lukas Hofer)

Perg, 3.4.2024 Unterschrift Julian Primetshofer

(Julian Primetshofer)

3. Danksagung

Wir möchten uns bei allen Personen und Organisationen, die uns diese Diplomarbeit ermöglicht haben, bedanken.

Besonders erwähnen möchten wir hier unseren Diplomarbeitsbetreuer Herrn Professor Mag. Michael Holzmann, der uns stets mit Rat und Tat zur Seite stand.

Weiters bedanken wir uns sehr herzlich bei unseren Diplomarbeitsbetreuern und Unterstützern von Fabasoft. Hier gilt unser Dank besonders Herrn Franz Deimling, welcher uns jederzeit bei Fragen und Problemen zur Seite stand und vor allem während unseres Praktikums umfassende Unterstützung leistete.

4. Kurzfassung

Bei dem Projekt OCR-Post-Processing handelt es sich grundsätzlich um die Nachverbesserung von Text, welcher von einer Texterkennung erstellt wurde. Ziel ist es, mit verschiedenen Ansätzen möglichst gute Verbesserungen zu erlangen.

Zur Realisierung werden dabei vier grundlegende Komponenten benötigt:

Das **Backend**, welches mittels drei verschiedener Ansätze versucht, die gegebenen Texte zu verbessern. Zu diesen zählen zwei Wörterbuchansätze und eine Verbesserung mittels KI. Weiters werden im Backend Statistiken berechnet, um beurteilen zu können, welches System wie gut funktioniert.

Getestet und visualisiert wird die Verbesserung durch das **Frontend**. Mit diesem können mehrere Dateien oder ganze Datasets gleichzeitig hochgeladen und mit allen gewünschten Korrektursystemen verbessert werden. Nach Abschluss der Verbesserung werden alle Veränderungen angezeigt und die im Backend berechneten Statistiken visualisiert.

Die Benutzeroberfläche und das dahinterliegende Backend werden von einer **API-Schnittstelle** verbunden. Dazu wurden in Python mittels Flask einige Endpunkte definiert, welche dazu benötigt werden, um entweder Informationen vom Frontend and das Backend oder umgekehrt zu senden.

Um auch die Verbesserung ohne eines Frontends verwenden und testen zu können, gibt es eine **Testpipline**, welche eine vordefinierte Ordnerstruktur durchlaufen kann und alle sich darin befindlichen Dateien verbessert und mit der Grundwahrheit vergleicht.

Das Ergebnis ist eine Gesamtanwendung, welche sich aus Python Modulen, JavaScript Code und PowerShell Skripten zusammensetzt.

5. Abstract

The OCR post-processing project involves the improvement of text generated by text recognition. The goal is to achieve the best possible improvements using various approaches. For implementation four components are required:

The **backend**, which tries to enhance the given texts using three different approaches, including two dictionary methods and one AI-based improvement. Additionally, the backend calculates statistics to assess the performance of each system.

Improvements are tested and visualized through the **frontend**, which allows for the uploading of multiple files or entire datasets at once to be improved with all desired correction systems. All changes are displayed after the improvement process is completed and the backend-calculated statistics are visualized.

An **API** connects the user interface and the backend. In Python, using Flask, several endpoints were defined to facilitate the exchange of information between the frontend and the backend.

Furthermore, a **test pipeline** allows for the improvement to be used and tested without a frontend. This pipeline can process a predefined folder structure, correcting all files contained and comparing them with the ground truth.

The result is a comprehensive application composed of Python modules, JavaScript code, and PowerShell scripts.

6. Inhaltsverzeichnis

7. Einleitung

7.1 Auftraggeber

Unser Auftraggeber ist die Fabasoft AG, diese wurde 1988 in Linz gegründet. Fabasoft ist ein international tätiges Softwareunternehmen, welches sich auf die Entwicklung von digitalen Businesslösungen für Behörden und sonstige Organisationen spezialisiert hat. Zu den Kernkomponenten von Fabasoft zählen Dokumentenmanagement, Workflowmanagement, Vertragsmanagement sowie Cloud-Lösungen. (vgl. fabasoft.com, 2024)

7.2 Ausgangslage

7.2.1 Problemstellungen

Fabasoft plant eine OCR-Engine einzusetzen, um beispielsweise Text aus PDFs oder Bildern zu extrahieren. Diese Engine ist oft fehleranfällig, vor allem wenn die Auflösung des Bildes (DPI) niedrig ist. Daraus folgen Wörter, die keinen Sinn ergeben, da Buchstaben von der Texterkennung falsch erkannt worden sind. Infolgedessen wurde das Projekt OCR Post Processing umgesetzt, um die Anzahl an falschen Wörtern zu minimieren.

7.3 Allgemeine Zielsetzung

Das Ziel ist, eine Software zu entwickeln, die in der Lage ist, .hOCR-Dateien einzulesen. Der darin enthaltene Text soll anschließend durch Korrektursysteme verbessert werden. Der verbesserte Text soll dann wieder in eine .hOCR-Datei exportiert werden. Weiters soll besonders darauf geachtet werden, die Software leicht erweiterbar zu programmieren, um das Hinzufügen von neuen Korrektursystemen so einfach wie möglich zu gestalten. Zusätzlich soll eine Webanwendung entwickelt werden, um die Leistung verschiedener Korrektursysteme zu vergleichen.

7.4 Ausgangssituation der Implementierung

7.4.1 Derzeitige Korrektursysteme

Die *ProcessingThread*-Klasse ist das Korrektursystem des Ausgangsproduktes. Bei jeder Verbesserung wurde eine Instanz dieser Klasse erstellt, welche als Thread lief, bis die Verbesserung abgeschlossen war. Bei diesem System wurde die Bibliothek PySpellChecker, welche in Kapitel 8.4.7 PySpellChecker genauer beschrieben wird, welche mittels Wörterbuchs eine Korrektur an den zu verbessernden Wörtern vornimmt, eingesetzt. Dabei wird jedoch nicht jedes Wort verbessert, sondern darauf geachtet, welche Konfidenz von bei der Datenübertragung ans Backend mitgeschickt wurde. Falls sich ein Wort unter dieser festgelegten Konfidenz befindet, wird dieses verbessert, ansonsten wird es nicht beachtet.

7.4.2 Frontend Aufbau

Das Frontend wurde im Zuge unserer Diplomarbeit erweitert. In diesem Unterkapitel wird der Ausgangszustand (zu erkennen in Abbildung 7.1), auf dem unsere Diplomarbeit aufbaut, erläutert. In diesem gab es lediglich die Möglichkeit, ein einziges Dokument zur Verbesserung hochzuladen.

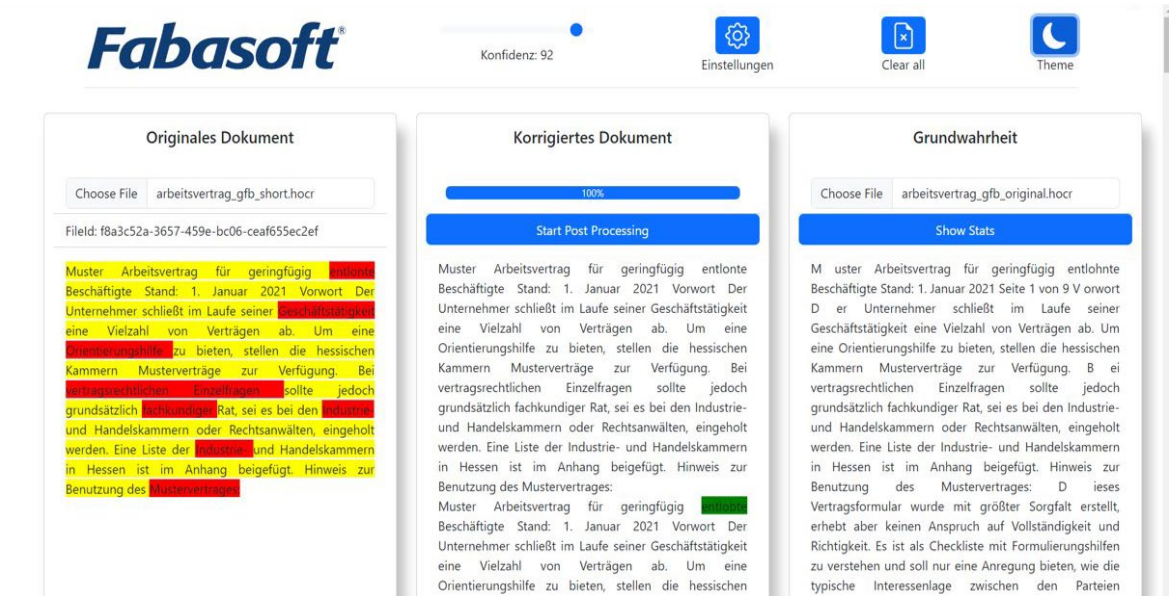


Abbildung 7.1 - Frontend Ausgangslage

In Abbildung 7.2 sind alle Funktionen der Frontend Applikation, auf die aufgebaut wurde zu erkennen. Der Benutzer hat einerseits die Möglichkeit ein Originales Dokument sowie eine dementsprechende Ground Truth hochzuladen. Er hat ebenfalls die Möglichkeit die API-URL in den Einstellungen zu verändern, die Statistiken mittels Stats Button abzufragen, das fertig korrigierte Dokument herunterzuladen oder eigene Verbesserungsvorschläge an die API zu senden.

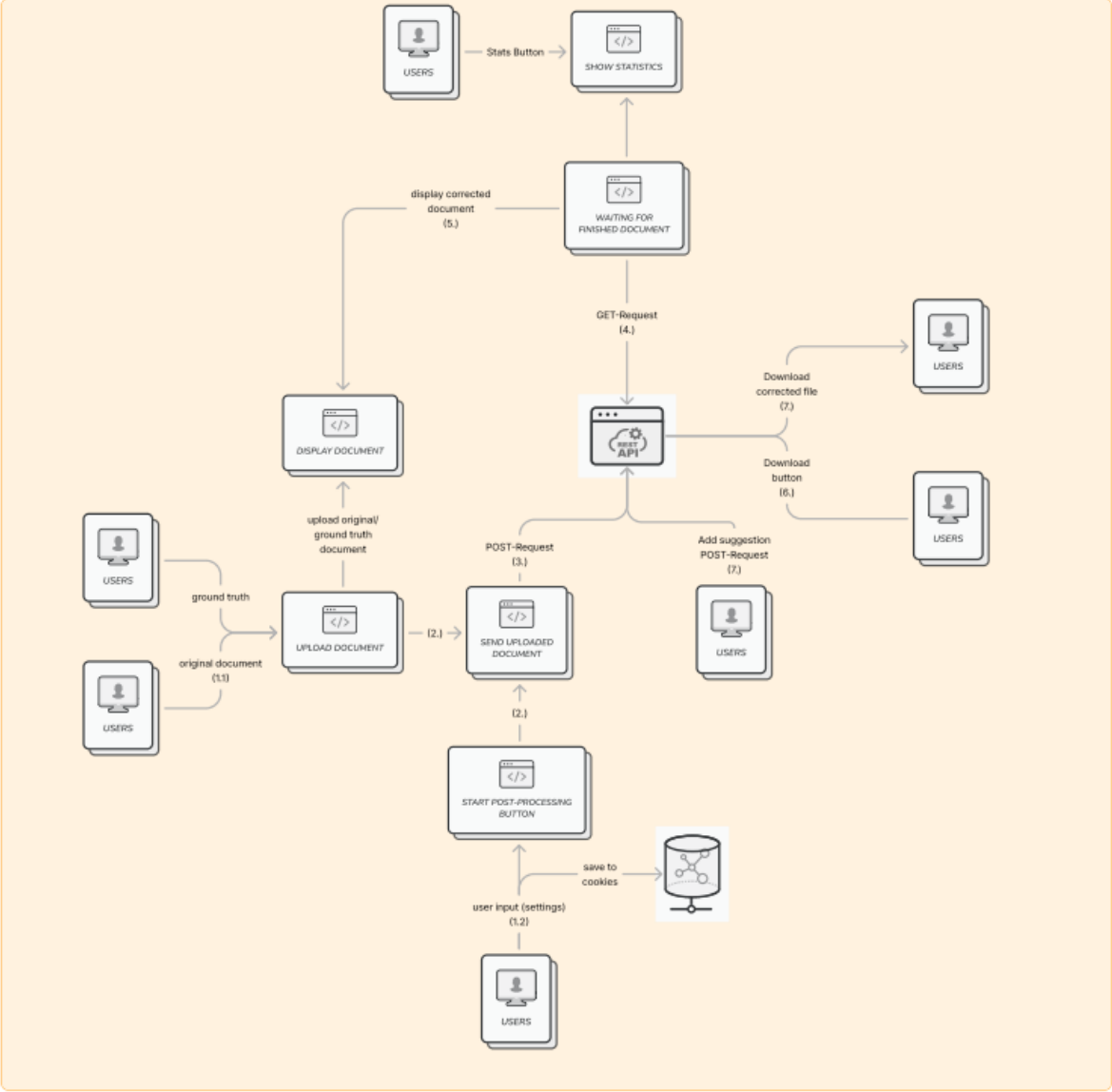


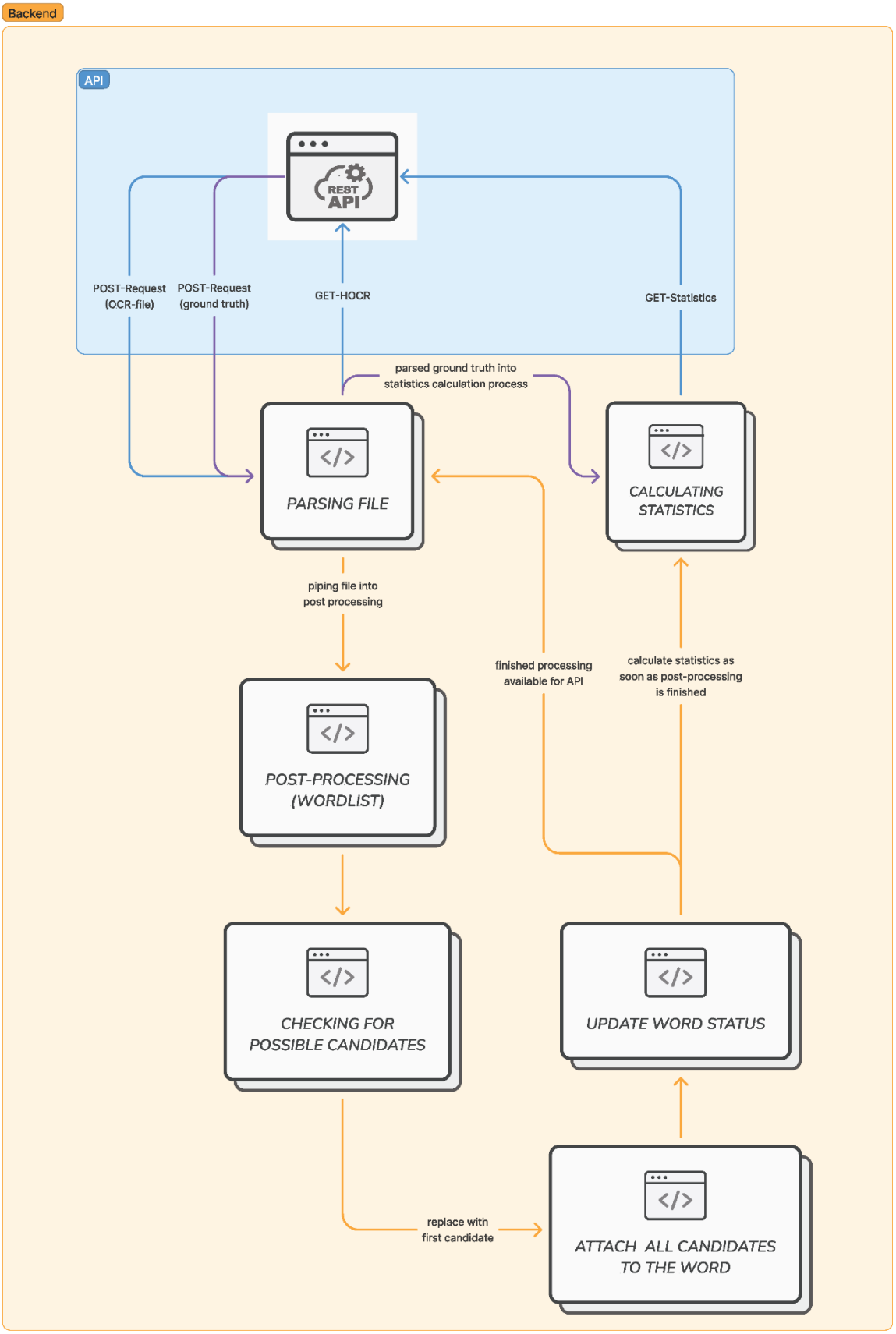
Abbildung 7.2 - Frontend Architektur

7.4.3 Backend Aufbau

Da das Backend von diesem Projekt auf einem bereits existierenden Backend basiert, wurden gewisse Grundfunktionalitäten übernommen. Bei dem ursprünglichen Aufbau ist es so, dass die .hOCR-Daten des zu verbessernden Dokuments per POST-Request an das Backend übermittelt werden. Da die Daten im hOCR-Format sind, müssen diese erst in eine richtige Form gebracht werden, damit diese weiterverarbeitet werden können. Dazu verwenden wir die in Kapitel 8.4.6 BeautifulSoup beschriebene Bibliothek, um die Daten in ein strukturiertes Datenmodell zu überführen.

Nachdem die Daten strukturiert wurden, werden sie mittels dem bereits vorhandenen Korrektursystem, welches in Kapitel 7.4.1 Derzeitige Korrektursysteme beschrieben wird, verbessert. Anschließend werden noch Statistiken bezüglich der Verbesserung berechnet, welche folgende Zahlen beinhaltet:

- Richtig korrigierte Wörter
- Falsch korrigierte Wörter


Backend
API
REST API
POST-Request (OCR-file)
POST-Request (ground truth)
GET-HOCR
GET-Statistics
parsed ground truth into statistics calculation process
PARSING FILE
CALCULATING STATISTICS
piping file into post processing
finished processing available for API
calculate statistics as soon as post-processing is finished
POST-PROCESSING (WORDLIST)
CHECKING FOR POSSIBLE CANDIDATES
UPDATE WORD STATUS
replace with first candidate
ATTACH ALL CANDIDATES TO THE WORD

8. Theoretische und fachpraktische Grundlagen und Methoden

8.1 Theoretische Grundlagen

8.1.1 Levenshtein Distanz

Die Levenshtein Distanz dient als Grundlage für wörterbuchbasierte Korrektursysteme. Diese Distanz wird auch als Editierdistanz bezeichnet, sie ermöglicht es zu berechnen, wie ähnlich zwei Zeichenketten sind. Die Levenshtein Distanz umfasst folgende drei Operationen: Einfügen, Löschen und Ersetzen. Die Levenshtein Distanz wird jedes Mal erhöht, wenn eine dieser Operationen verwendet wird. Diese Verfahren können individuell gewichtet werden. (vgl. wikipedia Levenshtein_distance, 2024)

Abbildung 8.1 - Levenshtein Distanz

Abbildung 8.1 - Levenshtein Distanz dient als Berechnungsbeispiel für die Levenshtein Distanz. In der Abbildung ergibt sich eine Editierdistanz von drei. Diese setzt sich aus zwei Einfügeoperationen und einer Ersetzungsoperation zusammen. Eingefügt wurden die Buchstaben „e“ und „a“. Ersetzt wurde der Buchstabe „i“ durch „f“.

8.1.2 Korrektursysteme

Es gibt zwei Arten von Korrektursystemen: wörterbuchbasierte und KI-basierte.

Wörterbuchbasierte Korrektursysteme nutzen ein Wörterbuch, um das ähnlichste Wort zu identifizieren. Die Ähnlichkeit eines Wortes wird in der Regel mithilfe der Levenshtein Distanz bestimmt.

KI-basierte Korrektursysteme sind meistens kontextbasiert, das bedeutet sie betrachten bei der Verbesserung den ganzen Satz. Daher ist es ihnen auch möglich die Wörter in die richtige Form zu bringen, wie zum Beispiel in den richtigen Fall oder die richtige Steigerungsform.

Um zu entscheiden, welches der vielen Systeme zum Einsatz kommen soll, sind die unten aufgelisteten Bibliotheken getestet worden.

- **oliverguhr/spelling-correction-german-base**
  Ein Modell von Hugging Face, welches auf NLP basiert.
- **bert-base-german-cased**
  Ein Modell von Hugging Face, welches auf NLP basiert.
- **SymSpell**
- Ein wörterbuchbasiertes Korrektursystem, auf Basis der Levenshtein Distanz, erweitert um die Fähigkeit, zwei vertauschte Zeichen zu identifizieren.
- **Pyspellchecker**
  Ein wörterbuchbasiertes Korrektursystem, das die Levenshtein Distanz nutzt.
- **Phunspell**
  Ein wörterbuchbasiertes Korrektursystem, welches mithilfe eines zusätzlichen Affix File funktioniert.

8.1.3 Testfälle

Die verschiedenen Korrektursysteme wurden vor der Implementierung getestet, um eine Entscheidungsgrundlage zu erhalten. Diese Tests teilen sich in 3 Testfälle auf. In jedem Testfall sind die Sätze „Input“ und „Correct Input“ zu finden.

Der Satz, der für die Korrektursysteme zur Verbesserung verwendet worden ist, ist der „Input“. „Correct Input“ beschreibt denselben Satz nur ohne Fehler. Daher sollten die beiden Sätze im Idealfall identisch sein, wenn der „Input“ verbessert worden ist.

8.1.3.1 Testfall 1

Dies ist der erste Test der Korrektursysteme. Der erste Testfall ist einfach gehalten, daher sind nur wenige Fehler im Satz enthalten.

Correct Input: Dieses Vertragsformular wurde mit größter Sorgfalt erstellt, erhebt aber keinen Anspruch auf Vollständigkeit und Richtigkeit.

Input: Dieses Vertragsiormular wurde mit größterSorgfalt erstelt, erhbbt aber keinen Ansprudh auf Vollstrndigkeit und Richtigkei

Tabelle 1 - oliverguhr/spelling-correction-german-base Testfall 1

System	Output	Corrected Words		accuracy
oli-verguhr/spelling-correction-german-base	Dieses Vertragsformular wurde mit größter Sorgfalt erstellt, erhebt aber keinen Anspruch auf Vollständigkeit und Richtigkeit.	before processing	after processing	100%
		Vertragsiormular	Vertragsformular	
		größterSorgfalt	größter Sorgfalt	
		erstelt	erstellt	
		erhbbt	Erhebt	
		Ansprudh	Anspruch	
		Vollstrndigkeit	Vollständigkeit	
		Richtigkei	Richtigkeit	

Tabelle 2 - bert-base-german-cased Testfall 1

System	Output	Corrected Words		accuracy
bert-base-german-cased	Dieses Buch wurde mit Recht angenommen, enthielt aber keinen Hinweis auf Recht und Unabhängigkeit.	before processing	after processing	0%
		Vertragsiormular	Buch	
		größterSorgfalt	Recht	
		erstelt	angenommen	
		erhbbt	enthielt	
		Ansprudh	Hinweis	
		Vollstrndigkeit	Recht	
		Richtigkei	Unabhängigkeit	

Tabelle 3 - SymSpell Testfall 1

System	Output	Corrected Words		accuracy
		before processing	after processing	
SymSpell	Dieses Vertragsformulare wurde mit größte Sorgfalt erstellt erhebt aber keinen Anspruch auf Voll-ständigkeit und Richtigkeit.	Vertragsiormular	Vertragsformulare	75%
		größterSorgfalt	größte Sorgfalt	
		erstelt	erstellt	
		erhbbt	erhebt	
		Ansprudh	Anspruch	
		Vollstrndigkeit	Vollständigkeit	
		Richtigkei	Richtigkeit	

Tabelle 4 - Pyspellchecker Testfall 1

System	Output	Corrected Words		accuracy
		before processing	after processing	
Pyspellchecker	Dieses vertragsformulare wurde mit größterSorgfalt erstellt, er-hebt aber keinen anspruch auf vollständigkeit und richtigkeit.	Vertragsiormular	vertragsformulare	62,5%
		größterSorgfalt	/	
		erstelt	Erstellt	
		erhbbt	erhebt	
		Ansprudh	anspruch	
		Vollstrndigkeit	vollständigkeit	
		Richtigkei	richtigkeit	

Tabelle 5 - Phunspell Testfall 1

System	Output	Corrected Words		accuracy
Phunspell	Dieses Vertragsformular wurde mit größter Sorgfalt Ersteller, erhebt aber keinen Anspruch auf Vollständigkeit und Richtigkeit.	before processing	after processing	87,5%
		Vertragsiormular	Vertragsformular	
		größterSorgfalt	größter Sorgfalt	
		erstelt	Ersteller	
		erhbbt	erhebt	
		Ansprudh	Anspruch	
		Vollstrndigkeit	Vollständigkeit	
		Richtigkei	Richtigkeit	

8.1.3.2 Testfall 2

Für den zweiten Testfall wurde wieder nur ein Satz gewählt, jedoch diesmal mit etwas schwierigeren Fehlern. Es wurde zweimal ein fehlerhaftes Wort mit einem korrekten Wort zusammengeschrieben. Zum Beispiel: Umoine (Um eine) oder KammernMustervelrträge (Kammern Musterverträge).

Correct Input: Um eine Orientierungshilfe zu bieten, stellen die hessischen Kammern Musterverträge zur Verfügung.

Input: Umoine Onentierungshilfe zu bietten, stallon die hessschen KammernMustervelrträge zur Verfüghg.

Tabelle 6 - oliverguhr/spelling-correction-german-base Testfall 2

System	Output	Corrected Words		accuracy
oli-verguhr/spellin g-correction-german-base	Um eine Orientierungshilfe zu bieten, fallen die hessischen Kam-mern Musterverträge zur Verfü-gung.	before processing	after processing	88,89%
		Umoine	Um eine	
		Onentierungshilfe	Orientierungshilfe	
		bietten	bieten	
		stallon	fallen	
		hessschen	hessischen	
		KammernMuster-velrträge	Kammern Musterverträge	
		Verfüghg	Verfügung	

Tabelle 7 - bert-base-german-cased Testfall 2

System	Output	Corrected Words		accuracy
bert-base-ger-man-cased	Zur kammern zu leisten, erhielten die beiden Bauern zur Verfügung.	before processing	after processi	14,29%
		Umoine	Zur	
		Onentierungshilfe	Kammern	
		bietten	leisten	
		stallon	erhielten	
		hessschen	beiden	
		KammernMustervelr-träge	Bauern	
		Verfüghg	Verfügung	

Tabelle 8 - SymSpell Testfall 2

System	Output	Corrected Words		accuracy
SymSpell	moine Orientierungshilfe zu bitten, stallone die hessischen Kammern unterverträge zur Verfüg hg	before processing	after processing	33,33%
		Umoine	moine	
		Onentierungshilfe	Orientierungshilfe	
		bietten	bitten	
		stallon	stallone	
		hessschen	hessischen	
		KammernMuster-velrträge	Kammern unterverträge	
		Verfüghg	Verfüg hg	

Tabelle 9 - Pyspellchecker Testfall 2

System	Output	Corrected Words		accuracy
Pyspellchecker	moine Orientierungshilfe zu betten, statlondie hessischen KammernMustervelrträge zur Verfüh	before processing	after processing	22,22%
		Umoine	moine	
		Onentierungshilfe	Orientierungshilfe	
		bietten	betten	
		stallon	statlondie	
		hessschen	hessischen	
		KammernMuster-velrträge	/	
		Verfüghg	Verfüh	

Tabelle 10 - Phunspell Testfall 2

System	Output	Corrected Words		accuracy
Phunspell	Umordne Klonierungshilfe zu bieten, Stall die hesseschen Gesellschafterverträge zur Verfüge.	before processing	after processing	33,33%
		Umoine	Umordne	
		Onentierungshilfe	Klonierungshilfe	
		bietten	bieten	
		stallon	Stall	
		hessschen	hesseschen	
		KammernMuster-velrträge	Kammern Musterverträge	
		Verfüghg	Verfüge	

8.1.3.3 Testfall 3

Um die Realitätsnähe zu erhöhen, wurde für den dritten Testfall, ein ganzer Absatz herangezogen. Die Korrektursysteme müssen in der Lage sein, ganze Dokumente zu verbessern, sobald sie implementiert sind.

Correct Input: Der Arbeitnehmer hat Anspruch auf einen gesetzlichen Mindesturlaub von derzeit 20 Arbeitstagen im Kalenderjahr – ausgehend von einer Fünf-Tage-Woche. Der Arbeitgeber gewährt zusätzlich einen vertraglichen Urlaub von weiteren …….. Arbeitstagen. Bei der Gewährung von Urlaub wird zuerst der gesetzliche Urlaub eingebracht.

Input: Der Arbeitnehmer hat Anspruch aut einn geselichen Mindesturlauh oon derzeit20 Arbeitstagen im Kalenderjahr - ausgebend von einer Fünf-Tage-Woche, DerArbeitgeber gewährt zusätzlich einen vertraglichen Urlaub wonweitaren ..Arbeitstagen. Bei der Gewährung won Urlaub wird zuerst der gesitzliche Urlaubeingebracht.

Tabelle 11 - oliverguhr/spelling-correction-german-base Testfall 3

<table>
<tr><th>System</th><th>Output</th><th colspan="2">Corrected Words</th><th>accuracy</th></tr>
<tr><td rowspan="12">oli-verguhr/spelling-correction-german-base</td><td rowspan="12">Der Arbeitnehmer hat Anspruch auf einen gesetzlichen Mindesturlaub von derzeit 20 Arbeitstagen im Kalenderjahr - ausgebend von einer Fünf-Tage-Woche, Der Arbeitgeber gewährt zusätzlich einen vertraglichen Urlaub von weiteren ..Arbeitstagen. Bei der Gewährung von Urlaub wird zuerst der gesetzliche Urlaub eingebracht.</td><td>before processing</td><td>after processing</td><td rowspan="12">93,75%</td></tr>
<tr><td>Aut</td><td>auf</td></tr>
<tr><td>Einn</td><td>einen</td></tr>
<tr><td>Geselichen</td><td>gesetzlichen</td></tr>
<tr><td>Mindesturlauh</td><td>mindesturlaub</td></tr>
<tr><td>Oon</td><td>von</td></tr>
<tr><td>Derzeit20</td><td>derzeit 20</td></tr>
<tr><td>Ausgebend</td><td>Ausgebend</td></tr>
<tr><td>DerArbeitgeber</td><td>Der Arbeitgeber</td></tr>
<tr><td>Wonweitaren</td><td>von weiteren</td></tr>
<tr><td>Won</td><td>von</td></tr>
<tr><td>Gesitzliche</td><td>gesetzliche</td></tr>
<tr><td>Urlaubeingebracht</td><td>Urlaub eingebracht</td></tr>
</table>

Tabelle 12 - bert-base-german-cased Testfall 3

System	Output	Corrected Words		accuracy
bert-base-german-cased	Der Arbeitnehmer hat Anspruch auf einn geselichen Mindesturlauh oon derzeit20 Arbeitstagen im Kalenderjahr - ausgebend von einer Fünf-Tage-Woche, DerArbeitgeber gewährt zusätzlich einen vertraglichen Urlaub wonweitaren ..Arbeitstagen. Bei der Gewährung won Urlaub wird zuerst der gesitzliche Urlaubeingebracht.	**before processing**	**after processing**	33,33%
		Aut	auf	
		Einn	den	
		Geselichen	Urlaub	
		Mindesturlauh	Urlaub	
		Oon	von	
		Derzeit20	an	
		Ausgebend	ausgehend	
		DerArbeitgeber	Er	
		Wonweitaren	von	
		Won	von	
		Gesitzliche	gesetzliche	
		Urlaubeingebracht	gewährt	

Tabelle 13 - SymSpell Testfall 3

<table>
<tr><th>System</th><th>Output</th><th colspan="2">Corrected Words</th><th>accuracy</th></tr>
<tr><td rowspan="11">SymSpell</td><td rowspan="11">Der Arbeitnehmer hat Anspruch aut einn gese lichen Mindes urlaub oon derzeit Arbeitstagen im Kalenderjahr ausgebend von einer Fünf Tage Woche dE Arbeitgeber gewährt zusätzlich einen vertraglichen Urlaub wo weiteren arBeitstagen Bei der Gewährung won Urlaub wird zuerst der gesetzliche Urlaub eingebracht.</td><td>before processing</td><td>after processing</td><td rowspan="11">37,5%</td></tr>
<tr><td>Aut</td><td>aut</td></tr>
<tr><td>Einn</td><td>einn</td></tr>
<tr><td>Geselichen</td><td>Gese lichen</td></tr>
<tr><td>Mindesturlauh</td><td>Mindes urlaub</td></tr>
<tr><td>Oon</td><td>Oon</td></tr>
<tr><td>Derzeit20</td><td>Derzeit 20</td></tr>
<tr><td>Ausgebend</td><td>Ausgebend</td></tr>
<tr><td>DerArbeitgeber</td><td>dE Arbeitgeber</td></tr>
<tr><td>Wonweitaren</td><td>wo weiteren</td></tr>
<tr><td>Won</td><td>won</td></tr>
<tr><td>Gesitzliche</td><td>gesetzliche</td></tr>
<tr><td>Urlaubeingebracht</td><td>Urlaub eingebracht</td></tr>
</table>

Tabelle 14 - Pyspellchecker Testfall 3

System	Output	Corrected Words		accuracy
Pyspellchecker	Der Arbeitnehmer hat Anspruch auto einn geselichen Mindesturlauh oon derzeit20 Arbeitstagen im Kalenderjahr - ausgebend von einer Fünf-Tage-Woche, DerArbeitgeber gewährt zusätzlich einen vertraglichen Urlaub wonweitaren ..Arbeitstagen. Bei der Gewährung wong Urlaub wird zuerst der gesitzliche Urlaubeingebracht.	before processing	after processing	0%
		Aut	auto	
		Einn	einn	
		Geselichen	/	
		Mindesturlauh	/	
		Oon	/	
		Derzeit20	/	
		Ausgebend	/	
		DerArbeitgeber		
		Wonweitaren	/	
		Won	wong	
		Gesitzliche	gesetzliche	
		Urlaubeingebracht	/	

Tabelle 15 - Phunspell Testfall 3

System	Output	Corrected Words		accuracy
Phunspell	Der Arbeitnehmer hat Anspruch au ein gesetzlichen Mindesturlaub von derzeit Arbeitstagen im Kalenderjahr - ausgebend von einer Fünftagewoche, Der Arbeitgeber gewährt zusätzlich einen vertraglichen Urlaub italienweiten ..Arbeitstagen. Bei der Gewährung wohn Urlaub wird zuerst der gesetzliche Auseinandergebracht.	before processing	after processing	43,75%
		Aut	au	
		Einn	ein	
		Geselichen	gesetzlichen	
		Mindesturlauh	Mindesturlaub	
		Oon	von	
		Derzeit20	derzeit 20	
		Ausgebend	/	
		DerArbeitgeber	Der Arbeitgeber	
		Wonweitaren	Italienweiten	
		Won	Wohn	
		Gesitzliche	gesetzliche	
		Urlaubeingebracht	Auseinandergebracht	

8.1.3.4 Auswertung

Tabelle 16 - Auswertung der Testfälle

Systeme	Testfall1	Testfall 2	Testfall 3	Durchschnitt
oliverguhr/spelling-correction-german-base	100%	88,89%	93,75%	94,2%
bert-base-german-cased	0%	14,29%	33,33%	23,8%
SymSpell	75%	33,33%	37,5%	48,6%
Pyspellchecker	62,5%	22,22%	0%	28,2%
Phunspell	87,5%	33,33%	43,75%	54,9%

In der Tabelle 16 - Auswertung der Testfälle, sind die Korrekturgenauigkeiten in Prozent pro Testfall sowie deren Durchschnitt angegeben. Dieses Ergebnis diente als Grundlage für die Entscheidung, welche Korrektursysteme implementiert werden sollen.

Bei den Testungen hat das System oliverguhr/spelling-correction-german-base eindeutig am besten abgeschnitten, aber bei genaueren Analysen stellte sich heraus, dass Wörter beliebig abgeschnitten oder hinzugefügt wurden. Deshalb hat sich dieses System nicht etabliert. Auf Wunsch des Auftraggebers sollte sowohl ein wörterbuchbasiertes als auch ein KI-basiertes Korrektursystem implementiert werden, weshalb die Entscheidung auf Phunspell und bert-base-german-cased fiel.

8.1.3.5 Batchwise script Ergebnis

Die Tabelle (Tabelle 17 - Testergebnisse pro DPI) enthält die Ergebnisse des Batchwise scripts (in Kapitel

8.1.3.6 Sidebar

Wurden die Dokumente nun hochgeladen, werden diese in der Sidebar angezeigt (in Abbildung 11.5 zu erkennen). Hier werden nun als oberste Ebene die Namen der hochgeladenen Dokumente angezeigt. Direkt darunter ist die hochgeladene Ground Truth zu erkennen. Unter dieser finden sich die jeweiligen Qualitätsstufen, welche wiederum alle Korrektursysteme untergeordnet haben. Klickt der Benutzer nun auf ein Korrektursystem wird das Dokument in entsprechender DPI-Stufe und die dazugehörige Ground Truth angezeigt. Dies kann in Abbildung 11.6 erkannt werden. In der Sidebar kann ebenfalls die Mindestwortlänge und Mindestkonfidenz für Wörter ausgewählt werden (genauer beschreiben in Kapitel 9.2.4 und 9.2.5).

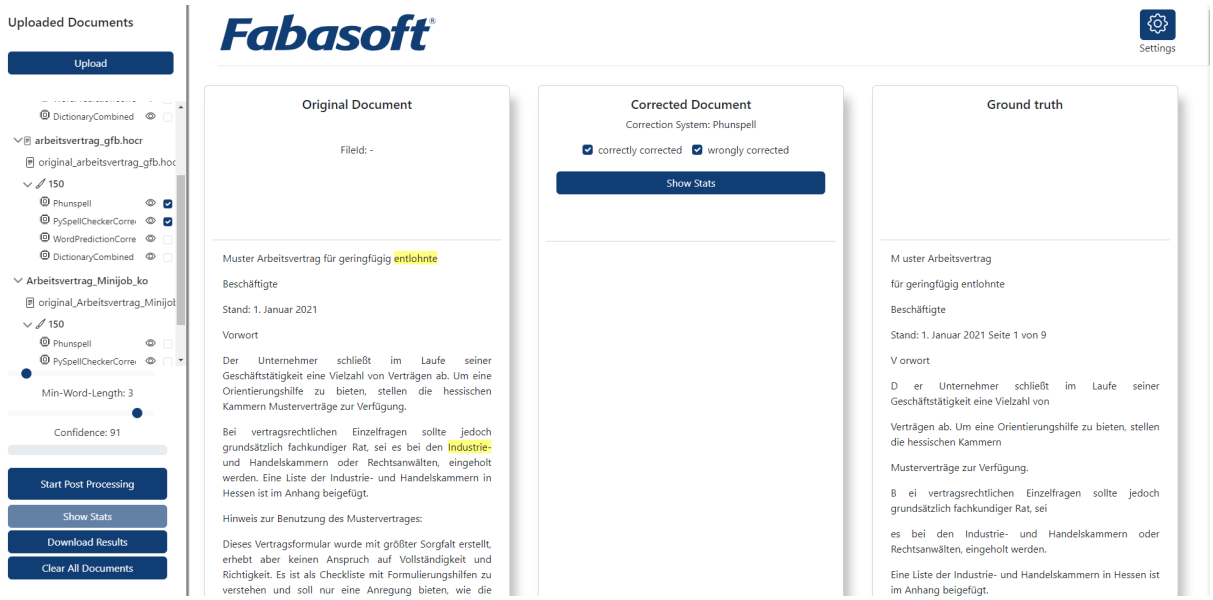


Abbildung 11.6 – Frontend Selected File

8.1.3.7 Verbesserungen Anzeigen

Wurden alle zur Korrektur gewünschten Files mittels Checkbox ausgewählt, kann nun mit drücken des Start Post Processing Buttons der Verbesserungsvorgang gestartet werden (genauer beschrieben im Kapitel zur Implementierung des Post Processing Vorgangs 10.1.9). Die richtig und falsch verbesserten Wörter werden im Anschluss farblich gekennzeichnet, falls eine Ground Truth hochgeladen wurde. In Abbildung 11.7 kann man dies erkennen, wobei in der linken Spalte das originale Dokument und in der rechten Spalte das korrigierte Dokument dargestellt ist. Die grün hinterlegten Wörter „monatlich“ und „entgeltlichen“ wurden vom Korrektursystem laut Ground Truth richtig verbessert. „Aderung“ wurde jedoch falsch verbessert und wird somit rot hinterlegt.

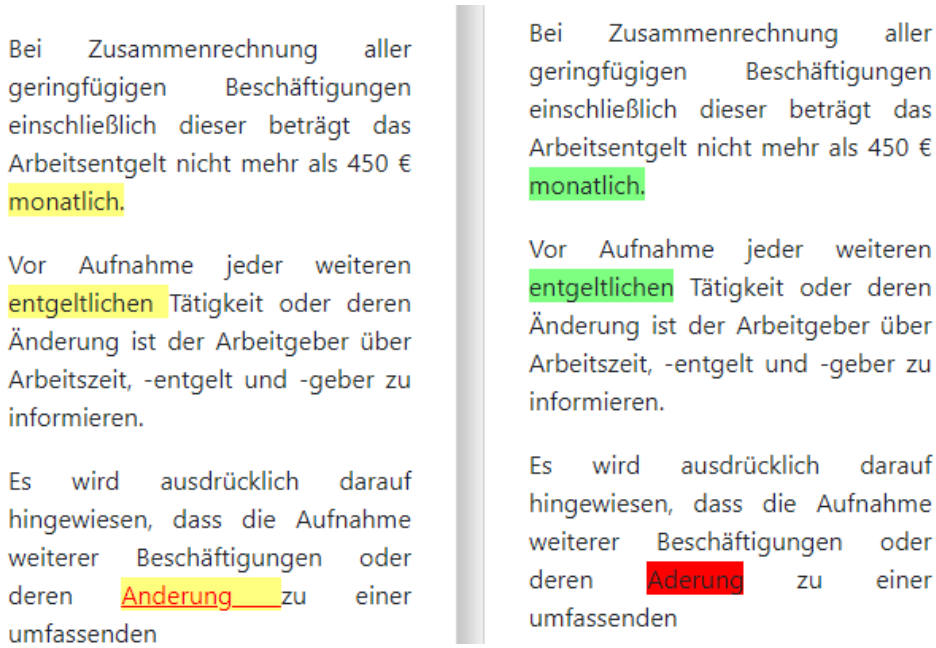


Abbildung 11.7 - Highlighting

Klickt der Benutzer mit einem Rechtsklick auf ein verbessertes Wort, so kann dieser sich auch alternative Vorschläge des Korrektursystems anzeigen lassen, wie in Abbildung 11.8 zu erkennen ist.

zu finden), welches im Rahmen dieser Analyse zur Anwendung kam. Die Tabelle stellt die Leistung der verschiedenen Korrektursysteme dar.

Die Spalte *Correctly* gibt die Anzahl der korrekt vorhergesagten Wörter an. Die Spalte *Wrongly* widerspiegelt die Anzahl der falsch vorhergesagten Wörter. *OCR_Correct* umfasst diejenigen Wörter, die bereits korrekt waren und vom jeweiligen Korrektursystem auch als korrekt markiert wurden. In der Spalte *OCR_False* ist die Anzahl an Wörter zu finden, die das Korrektursystem als falsch erkannt hat, jedoch richtig gewesen wären.

Die Summe der richtig gestellten Wörter ist in der Spalte *Correct* zu finden und setzt sich aus den richtig vorhergesagten (*Correctly*) und bereits korrekten Wörtern (*OCR_Correct*) zusammen. Auf der anderen Seite zeigt die Spalte *Wrong* die Summe aus falsch erkannten (*OCR_False*) und falsch korrigierten Wörtern (Wrongly).

Die Prozentsätze in den Spalten *Correct %* und *Wrong %* zeigen den Prozentsatz der insgesamt korrekten beziehungsweise falschen Korrekturen. Diese Werte dienen als Vergleichswert der verschiedenen Korrektursysteme.

Mithilfe der Tabelle 17 - Testergebnisse pro DPI werden die Auswirkungen der unterschiedlichen Auflösungen (**Fehler! Verweisquelle konnte nicht gefunden werden.**) auf die drei implementierten Korrektursysteme evaluiert. Auffällig ist, dass alle implementierten Korrektursysteme bei dem DPI-Wert von 100 am besten performen. Danach nimmt die Korrekturgenauigkeit stetig ab. Je niedriger der DPI-Wert, umso schlechter ist die OCR-Erkennung und die Korrektur fällt schwerer. Außerdem fällt auf, dass das Korrektursystem Pyspellchecker am besten abgeschnitten hat. Das ist dem Wert *Correct %* zu entnehmen.

Tabelle 17 - Testergebnisse pro DPI

systems	DPI	CORRECTLY	OCR_CORRECT	WRONGLY	OCR_FALSE	CORRECT	WRONG	CORRECT %	WRONG %
WordPrediction	150	22	49	248	22	71	270	20.821	79.179
WordPrediction	100	29	342	880	211	371	1091	25.376	74.624
WordPrediction	70	150	1117	3821	1395	1267	5216	19.543	80.457
WordPrediction	50	205	429	5762	1982	634	7744	7.567	92.433
PySpellChecker	150	3	55	56	26	58	82	41.429	58.571
PySpellChecker	100	69	398	245	269	467	514	47.604	52.396
PySpellChecker	70	348	1253	1835	1779	1601	3614	30.7	69.3
PySpellChecker	50	146	453	3136	2594	599	5730	9.464	90.536
Phunspell	150	5	51	91	32	56	123	31.285	68.715
Phunspell	100	57	349	443	316	406	759	34.85	65.15
Phunspell	70	244	1126	3121	1757	1370	4878	21.927	78.073
Phunspell	50	186	439	5355	2199	625	7554	7.642	92.358

8.1.4 DPI

Die Maßeinheit „Dots Per Inch“ (DPI) oder Punkte pro Zoll gibt die Anzahl an Bildpunkten in einem Zoll an. Ein Zoll beträgt 2,54 cm. Das bedeutet, dass bei hoher DPI-Anzahl auch die

Bildqualität steigt. Im Zusammenhang mit der optischen Zeichenerkennung (OCR) ist dies von großer Bedeutung, da die OCR-Engine genaue und klare Bilder benötigt, um Wörter zuverlässig zu erkennen. (vgl. pixx, 2024)

8.1.5 OCR

Die optische Zeichenerkennung (engl. optical character recognition) ist eine Technologie, welche es ermöglicht, aus einer Bild-Datei eine bearbeitbare, durchsuchbare und weiter verarbeitbare Datei zu machen. Eine OCR-Software kann zum Beispiel aus einer gescannten PDF-Datei eine Microsoft Word-Datei erstellen, um den eigentlichen Inhalt der PDF-Datei zur Verfügung zu stellen. (vgl. abbyy, 2024) (vgl. wikipedia Texterkennung, 2024)

8.1.5.1 Funktionsweise

Grundsätzlich erkennt eine OCR-Engine aus den einzelnen Bildpunkten eines Bildes Buchstaben und fügt diese dann zu Wörtern und Sätzen zusammen. Weiters wird der erkannte Text in Absätze, Abschnitte und Seiten gegliedert. Die kleinste Einheit beim Erkennen bleibt jedoch das Wort. Also auch wenn in der originalen Datei ein einzelner Buchstabe abgebildet wird, wird dieser als Wort mit einer Länge von eins im Ergebnis abgespeichert. Je nach Gruppierung der Wörter und Sätze werden zu diesen auch ihre Positionen mit abgespeichert, damit beim Ergebnis auch diese erhalten bleiben. Je nach Zeichenerkennung können auch andere Attribute bei den einzelnen Elementen abgespeichert werden oder sich die Gruppierungen unterscheiden (Beispiel Abbildung 8.2). (vgl. abbyy, 2024) (vgl. wikipedia Texterkennung, 2024)

8.1.5.2 Vorteile

Die optische Zeichenerkennung ermöglicht es, einen Text - anstatt ihn abzutippen - mit Technologieeinsatz in ein bearbeitbares Format zu übertragen, was Zeit erspart und Tippfehler vermeidet. Je nach Implementierung können auch verschiedene Formatierungen und Schriftarten erkannt werden. Google Tesseract kann zum Beispiel auch andere Schriften wie chinesische oder griechische Schriften erkennen. (vgl. github Tesseract Data, 2024) Außerdem können OCR-Engines auch größere Elemente wie Tabellen erkennen und direkt in das Ergebnis der Zeichenerkennung übernehmen. (vgl. abbyy, 2024) (vgl. wikipedia Texterkennung, 2024)

8.1.5.3 Nachteile

Sollte die Qualität des Bildes oder des Scans nicht ausreichend sein, kann die OCR-Engine nur eingeschränkt Wörter erkennen, was dazu führt, dass das Resultat wenig sinnvolle Wörter enthält. Statt einem Wort, oder einer sinnlosen Anreihung von Buchstaben kann es sogar dazu kommen, dass dann einfach ein Wort aus einer Vielzahl an Punkten oder anderen Sonderzeichen erkannt wird. (vgl. abbyy, 2024) (vgl. wikipedia Texterkennung, 2024)

8.1.5.4 Beispiele

Tesseract, EasyOCR, ocropus, ocropus 0.4, kraken, gocr, Ocrad, ocular, SwiftOCR, attention-ocr, RWTH-OCR, simple-ocr-opencv, Calamari, doctr (vgl. github Beispiele OCR-Engines, 2024)

8.1.6 hOCR-Standard

Der .hOCR-Standard beschreibt das Format des Ergebnisses der optischen Zeichenerkennung. Das Dateiformat basiert auf XHTML beziehungsweise HTML. XHTML ist eine striktere Version des HTML-Standards, welche sich mehr an XML anlehnt. Es werden nicht nur die Wörter, sondern auch weitere Daten wie das Layout, die Formatierung, aber auch die Genauigkeit des erkannten Wortes abgespeichert. Nach der Dublin-Core-Konvention werden Metadaten in dem HTML <meta> Tags abgespeichert. (vgl. wikipedia .hOCR-Standard, 2024) (vgl. w3schools, 2024)

Die Dublin-Core-Konvention ist eine Reihe von Metadatenstandards, die entwickelt wurden, um die Beschreibung von Ressourcen im Internet zu erleichtern. Sie wurde erstmals 1995 auf einer Konferenz in Dublin, Ohio, vorgeschlagen und seitdem weiterentwickelt. Die Konvention bietet eine Reihe von 15 grundlegenden Elementen, die verwendet werden können, um verschiedene Aspekte einer Ressource zu beschreiben, darunter Titel, Autor, Datum, Sprache, Themen und Beschreibung. (vgl. dublincore, 2024)

8.1.6.1 Begriffe des hOCR-Standards

- Element:
  - Im .hOCR-Standard ist ein Element ein Teil eines Bildes. Zum Beispiel der Tag, um eine Seite zu identifizieren, ist als Element zu kategorisieren
- Property:
  - Als Properties werden Eigenschaften von Elementen bezeichnet. Zum Beispiel die Korrekturgenauigkeit (confidence) bei einem Wort. Diese werden im „title“-Tag gespeichert und mittels Semikolons abgetrennt
- Metadaten:
  - In den Metadaten befinden sich Zusatzinformationen wie die Namen der Elemente und Properties und diverse andere Informationen.

(vgl. github .hOCR-Spezifikation, 2024)

8.1.6.2 Ergebnis nach Zeichenerkennung mit Google Tesseract

Tesseract ist eine optische Zeichenerkennung, welche von Google entwickelt wurde und frei zur Benutzung zur Verfügung steht. Tesseract speichert in den oben genannten <meta>-Tags,

da sich in dem Resultat mehrere befinden. In diesen befinden sich mehrere Informationen. Unter anderem beinhalten die <meta> Tags den Namen des OCR-Systems, und unter dem Namen „ocr-capabilities“ die Namen der Tags, welche zur Gliederung der .hOCR-Datei verwendet werden und die Namen der Genauigkeit. Außerdem wird noch der „Content-Type“ gespeichert, welcher besagt, dass sich in der Datei Text und HTML befinden, und in welcher Enkodierung diese sind. (vgl. github Tesseract Ergebnis, 2024) (vgl. wikipedia Tesseract, 2024)

Für die Testdaten im Testdatensatz wurde Tesseract verwendet, um diese zu erzeugen.

Unten ist ein kurzes Beispiel für eine .hOCR-Datei. Dies ist ein Ausschnitt aus einer, für Testzwecke verkürzten, Datei aus dem Testdatensatz. Die Tags wurden zwecks Vollständigkeit am unteren Ende wieder geschlossen.

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
 <head>
  <title></title>
  <meta http-equiv="Content-Type" content="text/html;charset=utf-8"/>
  <meta name='ocr-system' content='tesseract 5.1.0' />
  <meta name='ocr-capabilities' content='ocr_page ocr_carea ocr_par
ocr_line ocrx_word ocrp_wconf'/>
 </head>
 <body>
  <div class='ocr_page' id='page_1' title='image
"/home/user/Dokumente/ocr/output_png/dpi_100#rotated_0#blur_radius_0.5/arbe
itsvertrag_gfb/0.png"; bbox 0 0 2481 3507; ppageno 0; scan_res 100 100'>
   <div class='ocr_carea' id='block_1_1' title="bbox 381 1209 1989 1611">
    <p class='ocr_par' id='par_1_1' lang='deu' title="bbox 381 1209 1989
1611">
     <span class='ocr_line' id='line_1_1' title="bbox 390 1209 1800 1350;
baseline 0.002 -33; x_size 143; x_descenders 33; x_ascenders 26">
      <span class='ocrx_word' id='word_1_1' title='bbox 390 1209 834 1320;
x_wconf 96'>Muster</span>
      <span class='ocrx_word' id='word_1_2' title='bbox 882 1209 1800 1350;
x_wconf 95'>Arbeitsvertrag</span>
     </span>
     <span class='ocr_line' id='line_1_2' title="bbox 381 1467 1989 1611;
baseline 0.002 -33; x_size 144; x_descenders 32; x_ascenders 29">
      <span class='ocrx_word' id='word_1_3' title='bbox 381 1467 552 1581;
x_wconf 96'>für</span>
      <span class='ocrx_word' id='word_1_4' title='bbox 600 1467 1326 1611;
x_wconf 93'>geringfügig</span>
      <span class='ocrx_word' id='word_1_5' title='bbox 1383 1470 1989
1581; x_wconf 49'>entlonte</span>
     </span>
    </p>
   </div>
  </div>
 </body>
</html>
```

Abbildung 8.2 - .hOCR-Datei Beispiel

8.1.6.3 Aufbau – Relevante Elemente und deren Properties

Im Allgemeinen werden alle Elemente in einer hOCR-Datei mit einer ID versehen, welche fortlaufend ist, sich jedoch in ihrer Zusammensetzung bei manchen Elementen unterscheidet.

- ocr_page
  - Dieses Element stellt eine physische Seite dar.
  - Befindet sich in einem HTML Tag
  - id: page_<fortlaufende Nummer>
  - Properties:
    - image: Diese Property speichert den physischen Pfad zu der Bilddatei, welche von diesem Element dargestellt wird.
    - bbox: Diese Property ist die Bounding Box. Diese Box speichert die Koordinaten der linken oberen und rechten unteren Ecke.
    - ppageno: Diese Property speichert die physische Seitenzahl des Elements
    - scan_res: Diese Property speichert die dpi des Scans oder des Bildes, auf welchem die Zeichenerkennung angewandt wird.
- ocr_carea
  - Dieses Element stellt einen Textblock dar.
  - Befindet sich in einem HTML Tag
  - id: block_<Seiten-id>_<fortlaufende Nummer>
  - Properties:
    - bbox: Bounding Box, wie bereits bei den Properties der ocr_page beschrieben
- ocr_par
  - Dieses Element stellt einen Paragraphen in einem Textblock dar.
  - Befindet sich in einem HTML <p> Tag
  - id: par_<Seitennummer>_<fortlaufende Nummer>
  - lang: Die Sprache des Absatzes wir in diesem HTML-Attribut gespeichert
  - Properties:
    - bbox: Bounding Box, wie bereits bei den Properties der ocr_page beschrieben
- ocr_line
  - Dieses Element stellt eine Zeile dar
  - Befindet sich in einem HTML Tag
  - id: line_<Seiten-id>_<fortlaufenden Nummer>
  - Properties:
    - bbox: Bounding Box, wie bereits bei den Properties der ocr_page beschrieben
    - baseline: Die „Baseline“ stellt eine Art „Linie“ dar, auf welcher sich die Zeile befindet. Der erste Wert ist der Tangens des Winkels, in welchem die Zeile von links nach rechts abfällt. Der zweite Wert ist die Y-Koordinate relativ zur oberen linken Ecke der umgebenden Bounding Box

        - x_size: Diese Property ist Tesseract-spezifisch. Sie beschreibt die geschätzte Zeichengröße der Zeile in Pixel.
        - x_descenders: Diese Property ist Tesseract-spezifisch. Sie beschreibt die Anzahl an Pixel, um welche die Zeichen in dieser Zeile Unterlänge vorweisen. Eine Unterlänge hat zum Beispiel der Buchstabe „g“ (klein g), da dieser über den unteren Zeilenrand hinaus geht.
        - x_ascenders: : Diese Property ist Tesseract-spezifisch. Sie beschreibt die Anzahl an Pixel, um welche die Zeichen in dieser Zeile Überlänge vorweisen. Eine Überlänge hat zum Beispiel der Buchstabe „b“ (klein b), da dieser nicht eine halbe, sondern eine ganze Zeile hoch ist.
- ocrx_word
    - Dieses Element stellt ein Wort dar
    - Befindet sich in einem HTML Tag
    - id: word_<Seiten-id>_<fortlaufende Nummer>
    - Im Tag selbst befindet sich das eigentliche Wort, welches von der Zeichenerkennung erkannt wurde
    - Properties:
        - bbox: Bounding Box, wie bereits bei den Properties der ocr_page beschrieben
        - x_wconf: Die Konfidenz eines Wortes. Diese Konfidenz stellt den Prozentsatz dar, zu welchem sich die Zeichenerkennung bei der Erkennung dieses Wortes sicher ist.

(vgl. github .hOCR-Spezifikation, 2024)

8.2 Technologien

8.2.1 Python

Für das Backend unserer Diplomarbeit ist Python als Programmiersprache verwendet worden. Python ist eine vielseitige, interpretierte und objektorientierte Programmiersprache. Als Besonderheit ist die Formatierung anzumerken, da es keine geschwungenen Klammern gibt, sondern alles mit Einrückungen strukturiert wird. Ein Hauptgrund für die Wahl dieser Sprache liegt in der Vielzahl an bereits vorhandenen und gut funktionierenden Bibliotheken zur Verbesserung von Texten. (ryte wiki, 2015)

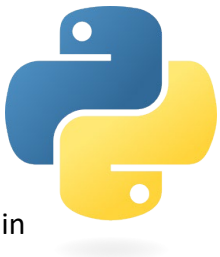


Abbildung 8.3 - Python (python, 2024)

8.2.2 Pip

Das Paketverwaltungsprogramm für Python Pakete wird Pip genannt. Dieses wurde in der Diplomarbeit verwendet, um Bibliotheken zu installieren, zu deinstallieren oder zu aktualisieren. (vgl. wikipedia Pip_(Python), 2024)

8.2.3 ReactJs

Für das Frontend unserer Diplomarbeit wurde das Framework ReactJs verwendet. Es handelt sich hierbei um eine Softwarebibliothek zur Erstellung von User-Interfaces, welches von Facebook entwickelt wurde. Es ermöglicht zudem die Unterteilung in einzelne Komponenten, welche einfach wiederverwendet werden können. (vgl. hubspot, 2024)

8.2.4 Bash

Die Bourne Again Shell (BASH) ist eine umfangreichere Version der Bourne-Shell (sh), der Standard-Shell in Linux- und GNU Betriebssystemen. Alle Befehle der Bourne-Shell sollen auch in der BASH lauffähig sein. BASH ist eine CLI, also ein Command Line Interface. Das bedeutet, der Benutzer interagiert über eine Befehlszeile mit dem Programm. BASH ist eine Betriebssystem-Shell. Das bedeutet, dass sie Zugriffe auf Ressourcen des Betriebssystems ermöglicht. Es können auch BASH-Befehle in Skripts zusammengefasst werden, damit sie nicht alle einzeln durch den Benutzer eingegeben werden müssen. (vgl. computerweekly, 2024)

Abbildung 8.4 - BASH-Logo (wikimedia bash logo, 2024)

8.2.5 Rest API

Representational State Transfer (REST) ist eine Architektur, welche Ressourcen, unter gewissen Anforderungen, zugängig macht. Diese Anforderungen umfassen:

- die Adressierbarkeit
    - Alle Ressourcen sind über eine URI (Unique Ressource Identifier) erreichbar.

Abbildung 8.5 - REST Symbol (opc-router rest symbol, 2024)

- einheitliche Schnittstellen
    - Auf Ressourcen wird einheitlich zugegriffen zum Beispiel über HTTP GET
- die Client-Server-Struktur
    - Die Daten sind auf einem Server abrufbar, und werden von einem Client abgerufen
- die Zustandslosigkeit
    - Jede Nachricht zwischen Client und Server enthält alle, für diese Anfrage nötigen Informationen.
- verschiedenen Repräsentationsmöglichkeiten der Daten
    - Verschiedene Darstellungsformen der Ressourcen, wie zum Beispiel XML oder JSON
- Hypermedia
    - Die Ressourcen sind alle über Hypermedia erreichbar. Ein REST-Client greift also ausschließlich über URLs (Uniform Ressource Locator) auf die API zu.

REST-APIs werden hauptsächlich mit HTTP beziehungsweise HTTPS umgesetzt und verwenden auch die HTTP-Methoden zum Zugriff auf die Ressourcen in Kombination mit den URLs. (vgl. ionos HTTP, 2024)

8.2.6 Docker Container

Ein Docker Container bietet eine isolierte und konsistente Umgebung für die Ausführung von Anwendungen, da er nur den notwendigen Code und die erforderlichen Bibliotheken enthält. Damit können Entwickler Anwendungen in einer standardisierten Umgebung bereitstellen, ohne sich um die Kompatibilität mit verschiedenen Betriebssystemen oder Infrastrukturen kümmern zu müssen. Diese Container sind portabel und können problemlos zwischen Entwicklung, Test- und Produktionsumgebungen verschoben werden, was die Bereitstellung und Wartung von Anwendungen vereinfacht.

Im Rahmen der Diplomarbeit wurden sowohl das Frontend als auch das Backend in separaten Docker Containern bereitgestellt. (vgl. aws Was ist der Unterschied zwischen Docker-Images und Containern?, 2024)

8.2.7 JavaScript

JavaScript ist eine Programmiersprache, welche zur Entwicklung von interaktiven Webseiten verwendet wird. In der Diplomarbeit wurde die Sprache zur Implementierung des User-Interfaces verwendet. Eine Besonderheit der Sprache liegt darin, dass sie nicht kompiliert wird, sondern Just-in-Time von einer JavaScript-Engine ausgeführt wird. (vgl. amazon, 2024)

8.2.8 Node Package Manager

Der Node Package Manager kurz NPM wird verwendet, um Softwarebibliotheken einfach über die Kommandozeile zu installieren. Installierte Pakete werden in der package.json gespeichert. (w3schools, 2024)

8.3 Entwicklungssysteme

8.3.1 Visual Studio Code

Visual Studio Code ist ein Open-Source Text-Editor, welcher bei dieser Diplomarbeit zur Implementierung des User-Interfaces verwendet wurde. (vgl. codeinstitute, 2024)

8.3.2 PyCharm Ultimate

PyCharm ist eine integrierte Entwicklungsumgebung (IDE), welche von Jet-Brains vor allem zur Programmierung mit Python entwickelt wurde. Mit PyCharm ist eine einfache Fehlerbehebung dank des integrierten Debuggers möglich, und es bietet auch einige andere Assistenzsysteme wie Code-Vervollständigung und intelligente Refaktorisierung. (biteno, 2024)

Abbildung 8.6 - PyCharm (jetbrains, 2024)

8.4 Bibliotheken und Plugins

8.4.1 OpenAPI-Generator

Der OpenAPI Generator Plugin für die Jet-Brains IDEs fügt ein GUI für den OpenAPI Generator hinzu, mit welchen sich Frontend-Clients, Backend-APIs und Dokumentation aus einer OpenAPI-Specification generieren lassen. Dieses Plugin ist kostenlos. (vgl. openapi-generator, 2024)

In dieser Diplomarbeit wurde das OpenAPI Generator Plugin für die Generierung des Frontend API-Clients und der Backend API aus einer OpenAPI-Specification verwendet.

8.4.2 Phunspell

Phunspell ist eine Python Bibliothek, welche entwickelt wurde, um die Funktionalitäten von Hunspell in Python Anwendungen zu nutzen. Die relevanteste Funktion für die Diplomarbeit, ist das Vorschlagen von Korrekturen für ein Wort.

8.4.3 Flask

Flask ist ein minimalistisch gehaltenes Web-Framework für die Programmiersprache Python. Durch diesen minimalistischen Ansatz wird in Flask keine bereits durch andere Bibliotheken umgesetzte Funktion neu implementiert, sondern diese kann in das Flask-Framework integriert werden. (vgl. ionos Flask Framework, 2024)

Abbildung 8.7 - Flask Logo (flask logo, 2024)

8.4.4 Hugging Face Transformers

Hugging Face ist eine Open-Source-Community, die sich auf die Entwicklung von künstlicher Intelligenz konzentriert. Das Ziel von Hugging Face Transformers ist es, die Entwicklung von künstlichen Intelligenzen zu erleichtern. Durch die API wird es ermöglicht, Modelle herunterzuladen und zu trainieren (vgl. huggingface, 2024)

Abbildung 8.8 - Hugging Face Logo (Hugging Face Logo, 2024)

8.4.5 Minineedle

Minineedle ist eine Python Bibliothek, welche wir dazu nutzen, zwei unterschiedlich lange Listen von Wörtern auf die gleiche Länge zu bringen. Dazu wird der Needleman-Wunsch-Algorithmus verwendet, welcher mit Hilfe von Backtracking versucht, einen optimalen globalen Similarity-Score zu erreichen. (vgl. hwlang, 2023)

8.4.6 BeautifulSoup

Beautiful Soup ist eine Python-Bibliothek, die speziell für das Parsing und die Manipulation von HTML- und XML-Dateien entwickelt wurde. Sie arbeitet mit verschiedenen Parsern zusammen, um auf intuitive Weise durch die Struktur dieser Dateien zu navigieren, Inhalte zu

suchen, zu modifizieren und zu extrahieren. Dies erleichtert die Aufgabe, Webdaten zu sammeln und zu verarbeiten, erheblich. Beautiful Soup bietet viele Funktionen und Werkzeuge, die es ermöglichen, komplexe Web-Scraping-Aufgaben mit relativ einfachem und verständlichem Code zu bewältigen. (vgl. crummy, 2024)

8.4.7 PySpellChecker

PySpellChecker ist eine Python-Bibliothek, die zur Identifizierung und Korrektur von Rechtschreibfehlern in Texten entwickelt wurde. Sie bietet eine einfache und effektive Möglichkeit, falsch geschriebene Wörter zu finden und Vorschläge für korrekte Schreibweisen zu generieren. Die Bibliothek basiert auf einem Wörterbuchansatz, unterstützt mehrere Sprachen und ermöglicht es Benutzern, eigene Wörterbücher hinzuzufügen oder die vorhandenen anzupassen. PySpellChecker kann sowohl einzelne Wörter als auch größere Textblöcke verarbeiten, wobei es neben der Korrekturfunktion auch die Möglichkeit bietet, eine Liste der falsch geschriebenen Wörter zu erhalten. (pypi, 2024)

8.4.8 Pandas

Pandas ist eine Python-Bibliothek, die für Datenanalyse und Datenmanipulation entwickelt wurde. Pandas kann auch mit häufigen Problemen in der Datenanalyse, wie fehlenden Werten, umgehen. In Pandas gibt es eindimensionale Arrays, welche „Series“ genannt werden und zweidimensionale Arrays, welche „Dataframes“ genannt werden. Pandas beinhaltet eine Vielzahl von verschiedenen Funktionen, um Daten aus verschiedenen Quellen wie zum Beispiel JSON-Dateien auszulesen. Dataframes und Series lassen sich auch auf verschiedene Weisen anzeigen, zum Beispiel durch die Ausgabe von bestimmten, relevanten Spalten, und nicht der ganzen Datenstruktur. (vgl. pandas, 2024)

8.4.9 Robot-Framework

Das Robot-Framework ist ein open-source Framework zur Test-Automatisierung. Dieses Framework verwendet eine einfache Syntax in den Skript-Dateien, und ist auch mit Python- oder Java-Code erweiterbar. Für das Robot-Framework existieren auch viele Tools und Library-Erweiterungen. (vgl. robotframework, 2024)

Abbildung 8.9 - Robot Framework Logo (robotframework, 2024)

8.4.10 Selenium

Die Selenium Library ist eine Python-Library, welche für die Nutzung mit dem Robot-framework auch die Erweiterung „robotframework-selenium-library“ bietet. Selenium erlaubt es dem Entwickler, von seinem Code aus, einen Browser zu öffnen, und in diesem Befehle auszuführen, als ob eine echte Person den Browser bedienen würde. Durch Selenium können anschauliche Programme und Tests für Webanwendungen umgesetzt werden, welche der Entwickler anschließend ausführen und live jeden Klick im Browser mitverfolgen kann. (vgl. selenium, 2024)

Abbildung 8.10 - Selenium Logo (swissq selenium logo, 2024)

8.4.11 IntelliBot @SeleniumLibrary patched

Der IntelliBot @SeleniumLibrary patched ist die gepatchte Version des IntelliBot @Selenium-Library für Pycharm, welcher der IDE erlaubt, die Syntax von .robot-Dateien zu verstehen, und auch die Syntax der Erweiterungen der Selenium Library zu verstehen. (vgl. intellibot, 2024)

8.4.12 Robot Helper

Der Robot Helper ist eine Firefox-Erweiterung, welche dem Benutzer erlaubt, seine Aktionen in Form von Robot-Befehlen mit der Verwendung der Selenium-Library aufzuzeichnen. Dies ermöglicht es dem Entwickler, Tests schneller zu schreiben, da er sich nicht für jedes Element die ID heraussuchen muss, um dieses anzusteuern. (robothelper mozilla addon, 2024)

Abbildung 8.11 - Robot Helper Logo (robothelper mozilla addon, 2024)

8.5 Sonstige Software

8.5.1 MobaXTerm

MobaXTerm ist eine Applikation, welche es ermöglicht, auf verteilte Systeme via SSH, RDP, FTP, VNC, … zuzugreifen. Es werden nützliche Tools wie ein SFTP Browser um Dateien hochzuladen und zu bearbeiten sowie eine SSH-Shell, um Befehle auszuführen, bereitgestellt. Im Rahmen der Diplomarbeit wurde die Software verwendet, um auf unsere Development VMs zuzugreifen. (vgl. mobaxterm, 2024)

8.5.2 Git

Git ist ein Open-Source-Tool zur Versionsverwaltung von Softwareprojekten. Zur Umsetzung der Diplomarbeit wurde mit der git feature branch workflow Methode gearbeitet. Diese verfolgt die Idee, dass für jedes neue Feature, welches implementiert wird, ein neuer Zweig erstellt wird. Dieser wird nach fertiger Implementierung und Testung in den development Zweig gemerged. So können alle Entwickler unabhängig voneinander neue Features entwickeln. (vgl. atlassian, 2024)

8.5.3 Cisco Jabber

Cisco Jabber ist eine Kommunikationssoftware, welche verwendet werden kann, um Meetings abzuhalten oder Kurznachrichten zu versenden. Für die Diplomarbeit wurde es zur Kommunikation mit dem Auftraggeber und für tägliche Meetings verwendet.

8.5.4 Remote Desktop

Das Remote-Desktop Tool von Windows wurde im Zuge der Diplomarbeit verwendet, um eine Remote Desktop Verbindung über das Remote Desktop Protokoll (RDP) herzustellen. Diese wurde benötigt, um einen grafischen Zugriff auf Development VMs zu ermöglichen.

8.5.5 Figma

Figma ist ein Online-Tool für Design und Prototyping. Es stellt unzählige Funktionen zum Erstellen von Entwürfen von User-Interfaces zur Verfügung. In der Diplomarbeit wurde es verwendet, um die Frontend Mockups und Klassendiagramme zu erstellen.

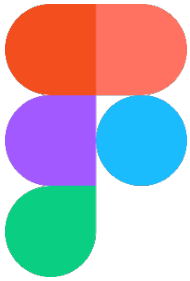


Abbildung 8.12 - Figma Logo (Figma, 2024)

8.5.6 Postman

Postman ist ein plattformunabhängiges Tool für die Verwaltung, Dokumentation und das Testen von APIs. Es bietet eine Vielzahl an Werkzeugen und Integrationen zum Entwickeln von APIs. (vgl. postman, 2024)

In dieser Diplomarbeit wird Postman zum Testen der API-Endpunkte verwendet.

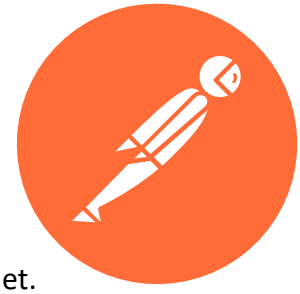


Abbildung 8.13 - Postman Logo (testautomatisierung postman logo, 2024)

9. Planung

9.1 Meilensteine und Stories

In der folgenden Auflistung steht das „M“ für das Wort „Meilenstein“, beziehungsweise wurde es bei der Benennung der Meilensteine als Abkürzung für dieses Wort verwendet. Die Stories und Abnahmekriterien, so wie die Namen der Meilensteine wurden auf Englisch verfasst. Die Storypunkte sind eine Einschätzung der Person, welche den Meilenstein zugeteilt bekommen hat, um selbst vor der Umsetzung einzuschätzen, wie lange sie dafür brauchen wird.

Die Meilensteine wurden für das Ferial/Pflichtpraktikum bei der Fabasoft AG definiert und jeweils für vier Wochen geplant. Jeder dieser Meilensteine wurde dann in zwei zwei-wöchige Meilensteine unterteilt. Die erste Zahl steht hierbei für Teil eins oder Teil zwei, und die zweite Zahl für den eigentlichen Meilenstein.

Beispiel: 2.1 Architecture

- 2: zweiter Teil des Meilensteins
- 1: Architecture

9.1.1 M1.1 Architecture

- Architecture of existing backend + frontend
    - Story: As developer I need to understand the project and how it is setup in detail. Therefore, I want a detailed diagrams of the architecture of the backend and the frontend as well as how they interact. Additionally, I need a class diagram covering the most important classes, so I now which parts are implemented, which parts are generated and how the project can/shall be extended.
    - Abnahmekriterien:
        - Diagram for backend architecture.
        - Diagram for frontend architecture.
        - Diagram for frontend and backend interaction.
        - Class diagram for backend
        - Class diagram for frontend
    - Storypunkte: 3

- Architecture of existing backend + frontend - Extension by planned features.
    - Story: As project lead, I need to know how the architecture of the project is set up and how it is going to be extended during the internship. Extend the existing diagrams by the planned classes and needed changes to achieve the milestones.
    - Abnahmekriterien:
        - Extended diagram for backend architecture
        - Extended diagram for frontend architecture
        - Extended diagram for frontend and backend interaction
        - Extended class diagram for backend
        - Extended class diagram for frontend
    - Storypunkte: 3
- Backend - Base class for correction
    - Story: As developer I want there to be a base class template that can be extended/adjusted to fit any complex correction system. An example shall be provided/applied on the existing code from the school project. The idea is to wrap any spell checker or other correction system by this class and make it useable by the framework. Set up the base class in a way that instances of a concrete correction system can be discovered and used via API.
    - Abnahmekriterien:
        - Base class implemented and documented.
        - Added API endpoints to list and use different implemented correctio systems.
        - Example class implemented for existing correction system.
        - Example tested and test documented/demonstrated to colleagues.
        - Definitions of Done:
            - Code needed for the feature is written (no TODOs are open)
            - There are tests for new features and the old tests still work.
            - Installation requirements are defined and documented in README or relevant document.
            - Demo + Tested by Story Owner
    - Storypunkte: 5

- Documentation of backend and how to extend it
    - Story: As developer I need to have some documentation on the backend of the ocr post-processing framework to be able to extend it and add arbitrary correction systems. I need to understand all methods of the base class and what needs to be implemented. Also, I want to be able to use the features of the IDE to know the datatypes and return values of the functions.
    - Abnahmekriterien:
        - Code documentation available in the source code.
        - Markdown or PDF or word file covering the steps needed to take to implement, add and test a new or existing correction system.
    - Storypunkte: 3

9.1.2 M1.2 Correction Systems

- Research spell correction systems
    - Story: As project lead, I want to know which spell correction systems are there and how they work to be able to evaluate the next steps. The documentation needs to include potential dependencies and licenses as well as a basic principle of how they work and are applied (e.g. word wise, sentence, context, dictionary, or AI).
    - Abnahmekriterien:
        - At least 5 spell correction systems found and documented.
    - Storypunkte: 3
- Basic testcase for spell correction systems
    - Story: As project lead, I want to know how well the researched correction systems work and need some first tests examples to compare them. Therefore 3 basic testcases shall be created and tested by all 5 spell correction systems in a similar manner. The results shall be documented and presented to the project lead.
    - Abnahmekriterien:
        - At least 3 basic testcases (spelling mistakes in sentences) created and documented.
        - Tested the basic testcases in the spell correction systems.

        - Demo for the project lead.
        - Definitions of Done:
            - Code needed for the feature is written (no TODOs are open)
            - There are tests for new features and the old tests still work.
            - Installation requirements are defined and documented in README or relevant document.
            - Demo + Tested by Story Owner
    - Storypunkte: 5
- Implement additional correction system
    - Story: As project lead, I want to be able to use and compare different correction systems available. Implement additional correction systems. Consider combining existing correction systems to improve the results.
    - Abnahmekriterien:
        - New correction system implemented.
        - Code and dependencies documented.
        - Definitions of Done:
            - Code needed for the feature is written (no TODOs are open)
            - There are tests for new features and the old tests still work.
            - Installation requirements are defined and documented in README or relevant document.
            - Demo + Tested by Story Owner
    - Storypunkte: 3
- Implement correction system using the base class defined in M1
    - Story: As project lead, I want to be able to use and compare different correction systems available. Set up 2-3 of the researched and tested spell correction systems and implement them using the base class defined in the M1.1 architecture.
    - Abnahmekriterien:
        - Base classes extended for every spell correction system + required methods.
        - Implemented code and dependencies documented.
        - Definitions of Done:

- Code needed for the feature is written (no TODOs are open)
- There are tests for new features and the old tests still work.
- Installation requirements are defined and documented in README or relevant document.
- Consulted UI/UX if applicable.
- Demo + Tested by Story Owner

  - Storypunkte: 8

9.1.3 M1.3 UI

- Mockups
  - Story: As user and project owner I need some mockups of the UI system to be able to view the UI design in advance and discuss the functionality. Draw some mockups for the upload of multiple files / or folders. Also draw some mockups for viewing and selecting different correction systems. Finaly some mockups shall be setup to reflect a test pipeline - it shall be possible to start the automatic processing of a set of documents and retrieve the aggregated results. Discuss this with the backend and plan the implementation for the next milestone.
  - Abnahmekriterien:
    - Mockups for:
      - Current system
      - Multi file/ folder system + docu how it shall be used.
      - View and selection of correction systems
  - Storypunkte: 3
- Discuss and revise mockups
  - Story: As user I want the best user experience possible, so I need the mockups of the processes in the UI to be discussed and possibly revised. Decide for implementation prioritization once mockups are ok.
  - Abnahmekriterien:
    - Meeting held.
    - Mockups revised.
    - Decision for next implementation steps
  - Storypunkte: 3

- Implementation of views
    - Story: As user I want to be able to test the actual interface not just the mockups. Start with the implementation of the frontend parts that do not depend on the backend. Prepare accessing the backend functionality as needed and discussed with the colleague.
    - Abnahmekriterien:
        - Mockups visuals implemented in the UI (HTML+CSS+ basic functionality)
        - Functionality prepared (does not have to work fully yet)
        - Definitions of Done:
            - Code needed for the feature is written (no TODOs are open)
            - There are tests for new features and the old tests still work.
            - Installation requirements are defined and documented in README or relevant document.
            - Consulted UI/UX if applicable.
            - Demo + Tested by Story Owner
    - Storypunkte: 5

9.1.4 M1.4 Testing

- Test API for basics
    - Story: As developer and user, I want the backend to be stable. Therefore, I need tests that upload different files and compare them to the expected output (hocr files). Calculate the expected difference on the files after they have been processed. Test at least an empty file and a very simple hocr file without an error, and one with a single error. Make sure that the tests (without expected correction) work for all the implemented correction systems from the colleagues.
    - Abnahmekriterien:
        - Script for basic test of hocr API
            - Empty file
            - Hocr file without error (compare to corrected version)
            - Hocr file with error (compare to corrected version)
        - Tests documented for basic correction system.
        - Tests documented for all available correction systems.

        - Definitions of Done:
            - Code needed for the feature is written (no TODOs are open)
            - There are tests for new features and the old tests still work.
            - Installation requirements are defined and documented in README or relevant document.
            - Demo + Tested by Story Owner
    - Storypunkte: 3
- Test backend batch wise
    - Story: As user of the project, I want to know how well it performs for a set of documents, I want to be able to start a script that iterates through folders containing sets with documents of different quality and obtain the results in a file. I also want a program to aggregate the statistics of the file and find out useful information about the quality of the correction systems.
    - Abnahmekriterien:
        - script to apply to dataset results for:
            - Files
            - Statistics per file
            - Errors in extra file if any occurred.
        - Definitions of Done:
            - Code needed for the feature is written (no TODOs are open)
            - There are tests for new features and the old tests still work.
            - Installation requirements are defined and documented in README or relevant document.
            - Demo + Tested by Story Owner
    - Storypunkte: 5
- Test API for basics
    - Story: As user I need the UI to be reliable even after some functionality has been added. Create automatic tests of the UI with a test framework such as app.test or appium or something similar (discuss before you start). Share and document the automatic test procedure with the colleagues for use before pushing the code. Revise and add tests as needed during the course of the project.

- Abnahmekriterien:
  - Tests for UI functionalities created.
  - Tests ran and documented.
  - Demo presented to colleagues.
  - Testpipeline shared with colleagues.
  - Definitions of Done:
    - Code needed for the feature is written (no TODOs are open)
    - There are tests for new features and the old tests still work.
    - Installation requirements are defined and documented in README or relevant document.
    - Demo + Tested by Story Owner
- Storypunkte: 5

9.1.5 M2.1 Architecture

- Tests for buildpipeline
  - Story: As deployer I want my containers to be stable and containing all the necessary files so that I am not surprised at runtime. Integrate the tests into the build pipeline.
  - Abnahmekriterien:
    - Tests integrated.
    - Build fails if tests fail.
  - Storypunkte: 2
- Unit Tests for API functions
  - Story: As developer I want to be able to verify that nothing is broken of the backend if new functionality is added. Add Unit Tests for API
  - Abnahmekriterien:
    - API unit tests available
  - Storypunkte: 3
- Unit tests for base class
  - Story: As developer I want to be able to verify that nothing is broken of the backend if new functionality is added. Add Unit Tests for base class functionality.

- Abnahmekriterien:
  - Unit tests for base class available
- Storypunkte: 3

- Update architecture diagrams
  - Story: As project owner I need to understand the project and dependencies. Update the architecture diagrams if changes require it.
  - Abnahmekriterien:
    - Up to date architecture diagrams
    - Changes documented.
  - Storypunkte: 2

9.1.6 M2.2 Correction Systems

- Application of test scripts
  - Story: As project owner, I want to know how well the researched correction systems work so I can plan future application. Use Test scripts from M1.4 to test the implemented correction systems.
  - Abnahmekriterien:
    - Systems tested.
  - Storypunkte: 1
- Results of test scripts evaluated
  - Story: As project owner I want to know which systems to select based on some concrete information. Aggregate information from tests and provide analysis and conclusions in documentation.
  - Abnahmekriterien:
    - Results documented.
    - Conclusions presented.
  - Storypunkte: 2

9.1.7 M2.3 UI

- Evaluate UI
  - Story: As project owner I need the UI to match the planned mockups. Compare the mockups to the implemented status and discuss deviations with project owner so that the different components can be planned accordingly.
  - Abnahmekriterien:
    - Mockups updated if required.
    - UI updated if required.
    - UI discussed with relevant parties.
  - Storypunkte: 2
- Implement UI and bugfixes
  - Story: As user of the UI, I want a nice user experience and full functionality. Implement the UI functions predefined. Fix bugs that were discovered by the test pipelines.
  - Abnahmekriterien:
    - No known bugs remaining.
    - Defined functions implemented.
  - Storypunkte: 5

9.1.8 M2.4 Testing

- Test API continued
  - Story: As developer and user, I want the backend to be stable, so that the frontend or other applications that build upon it run smoothly. Add tests for API endpoints not covered yet.
  - Abnahmekriterien:
    - All API endpoints necessary to upload, process and evaluate a file are covered by tests.
  - Storypunkte: 2
- Test backend batchwise – all systems

- Story: As a user of the system, I need all correction systems available to work continuously and provide the full functionality. Therefore, I want them thoroughly tested. Add tests for new correction systems / functionality that was not covered yet.
- Abnahmekriterien:
  - All available correction systems testable
- Storypunkte: 2

- UI Testing continued
  - Story: As a user of the UI I need everything to work smoothly, therefore testing of the UI functionality shall be covered by automatic tests.
  - Abnahmekriterien:
    - Automatic UI Tests extended to cover new functionality.
  - Storypunkte: 5

9.2 Frontend Mockups

9.2.1 Mockup

Das Mockup wurde zu Beginn der Diplomarbeit konzipiert, um ein Konzept für die neuen Features des Frontends zu entwerfen. Für die Erstellung wurde das Tool Figma verwendet.

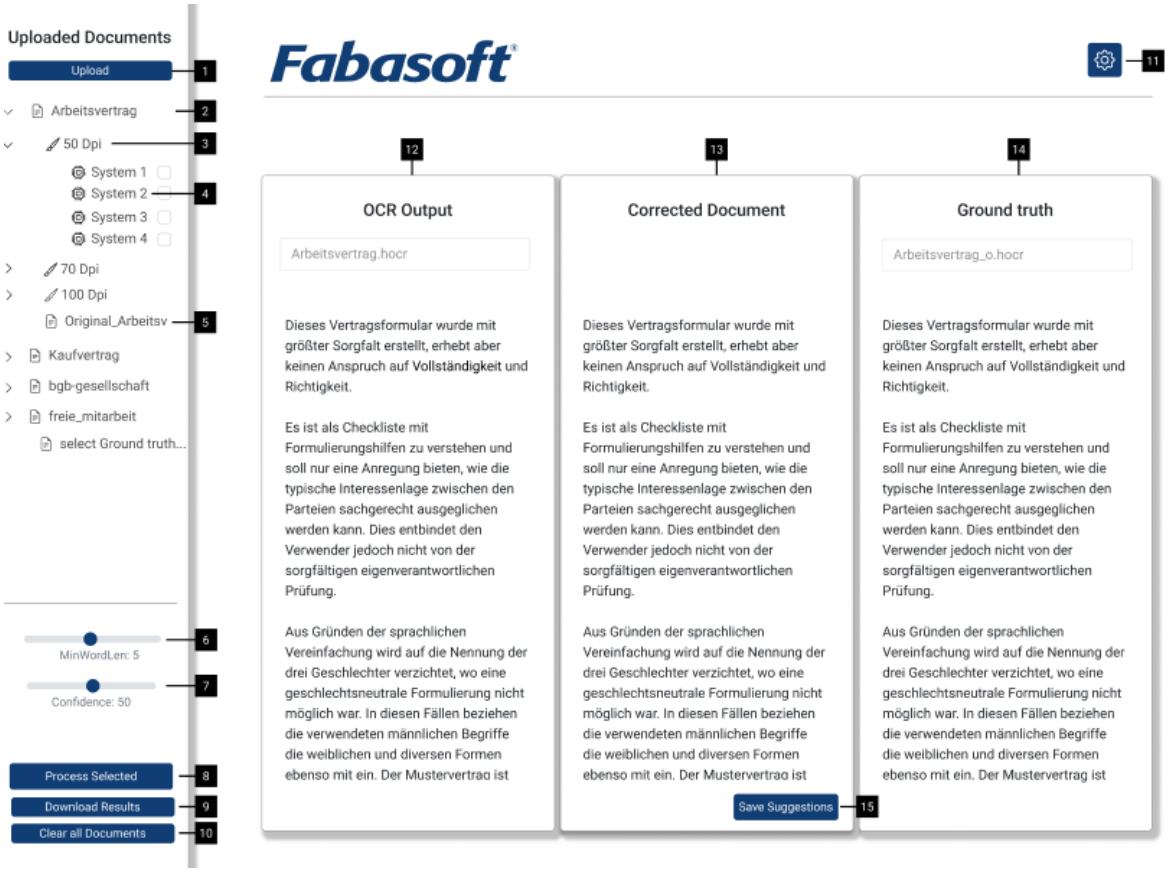


Abbildung 9.1 - Mockup Frontend

In Abbildung 9.1 kann das Mockup unserer Applikation erkannt werden, welches in diesem Kapitel genauer erläutert wird.

9.2.2 Upload Button

Der Button mit der Aufschrift „Upload“, in Abbildung 1 markiert mit der Nummer 1, wird dazu verwendet, um zu verbessernde .hOCR-Dateien oder eine entsprechende „Ground-Truth“ für diese hochzuladen. Beim Drücken dieses Buttons öffnet sich dafür ein Pop-Up, welches in Abbildung 2 dargestellt ist. In diesem kann dann mittels Radio Buttons ausgewählt werden, ob ein „Original Document“ (also die .hOCR welche die OCR-Engine, in unserem Fall Tesseract

produziert), eine „Ground Truth“ (die die korrigierte Form der .hOCR Datei darstellt) oder einen ganzen Ordner mit mehreren „Original Documents“ und „Ground Truths“ hochgeladen werden soll. Beim Hochladen eines „Ground Truth File“ muss zusätzlich noch das dazugehörige „Original Document“ mittels Dropdowns ausgewählt werden.

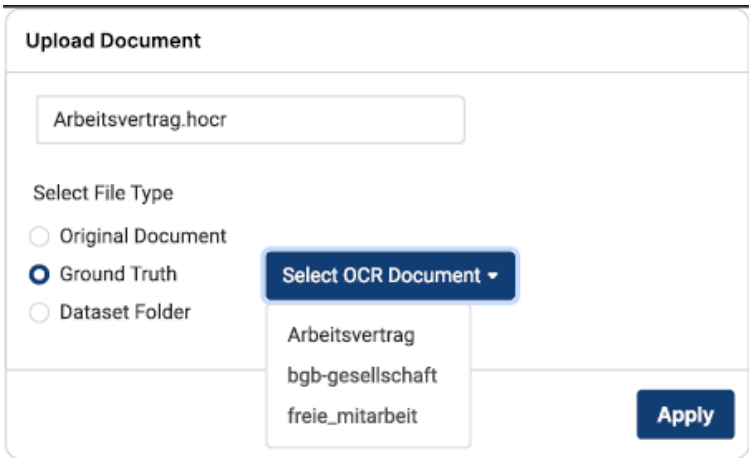


Abbildung 9.2 - Mockup Upload Pop-Up

9.2.3 Document Browser

Unterhalb des Upload Buttons befindet sich eine Sektion, in der sich die hochgeladenen .hOCR Dateien befinden (Abbildung 9.1, Nummer 2). Werden mehrere Qualitätsstufen eines Dokuments hochgeladen, wird dies ebenfalls in einer weiteren Ebene des Document Browsers kategorisiert (Abbildung 9.1, Nummer 3). Die unterste Ebene stellen die verschiedenen Korrektursysteme dar (Abbildung 9.1, Nummer 4). Wird eines davon angeklickt, wird das OCR-Dokument (Abbildung 9.1, Nummer 12) und gegebenenfalls das mittels entsprechendem Korrektursystem korrigierte Dokument (Abbildung 9.1, Nummer 13) und die „Ground Truth“ (Abbildung 9.1, Nummer 14) angezeigt. Wurde eine „Ground Truth“ hochgeladen, wird diese ebenfalls im Document Browser (Abbildung 9.1, Nummer 5) angezeigt.

9.2.4 Min Word Length Slider

Der Schieberegler, welcher in Abbildung 9.1 bei der Nummer 6 zu erkennen ist, wird dazu verwendet, um die Mindestlänge an Buchstaben festzulegen, die ein Wort haben muss, damit

es für das Post-Processing berücksichtigt wird. Im Anschluss werden nur Wörter berücksichtigt, welche eine Mindestlänge über dem eingestellten Wert haben.

9.2.5 Confidence Slider

Das Slider Element in Abbildung 9.1, markiert mit der Nummer 7, wird dazu verwendet, um den Konfidenzwert festzulegen. Der Konfidenzwert, genauer in Kapitel 8.1.5.3 beschreiben, legt fest, wie sicher sich die HOCR-Engine beim Erkennen eines Wortes war. Es werden nur Wörter für das Post-Processing berücksichtigt, welche einen Konfidenzwert unter dem ausgewählten Wert besitzen.

9.2.6 Process Selected Button

Der Button, welcher in Abbildung 9.1 mit der Nummer 8 markiert wurde, wird verwendet, um alle selektierten Dokumente an das Backend zu schicken, um das Post-Processing zu starten. Die Checkboxen befinden sich hierbei im Document Browser auf der Ebene der Korrektursysteme (Abbildung 9.1, Nummer 4). Damit wird ermöglicht, das Post-Processing eines Dokuments nur für ausgewählte Korrektursysteme zu starten.

9.2.7 Download Results Button

Der Button mit der Aufschrift „Download Results“ (zu erkennen in Abbildung 9.1, Nummer 9) kann zum Herunterladen der korrigierten Dokumente als .hOCR Dateien verwendet werden. Es werden jene Dateien heruntergeladen, welche davor mittels Checkbox ausgewählt wurden.

9.2.8 Clear all Documents

Der Clear all Documents Button, welcher in Abbildung 9.1 mit der Nummer 10 markiert wurde, kann verwendet werden, um alle hochgeladenen Dateien aus dem Document Browser zu löschen.

9.2.9 Einstellungen

Der Einstellungsbutton, welcher auf Abbildung 9.1 mit der Nummer 11 markiert ist, öffnet ein Pop-Up, welches in Abbildung 9.3 zu erkennen ist. Dieses kann verwendet werden, um die IP-Adresse der API festzulegen und um das Erscheinungsbild der Applikation zu ändern. Zur Auswahl stehen hierbei Dark-Mode und Light-Mode.

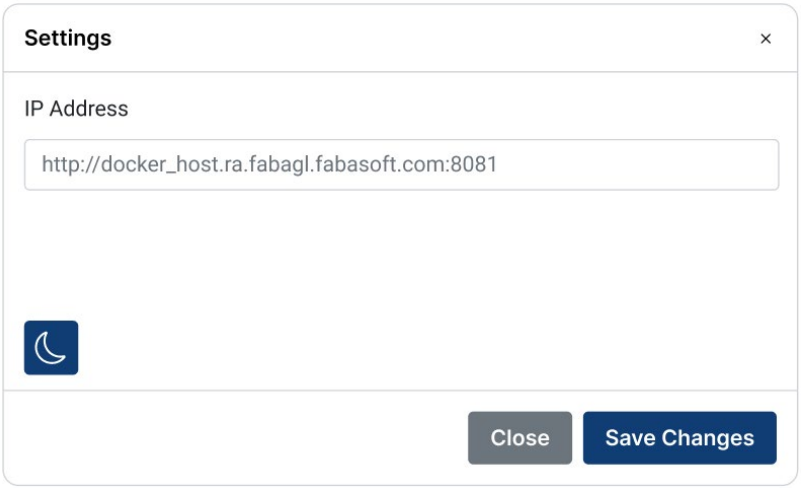


Abbildung 9.3 - Mockup Settings Pop-Up

9.2.10 Save Suggestions

Wird der Button mit der Aufschrift „Save Suggestions“ (zu erkennen in Abbildung 9.1, Nummer 15) gedrückt, so werden die vom Benutzer geänderten Wörter des korrigierten Dokuments an das Backend gesendet. Das Backend überschreibt nun die ausgebesserten Wörter durch die User-Suggestions, damit diese beim Download der Datei enthalten sind.

9.2.11 Progress Bar

Die Progress Bar, welche in Abbildung 9.4 zu erkennen ist, markiert mit der Nummer 1, zeigt an, wie viele Dokumente bereits fertig korrigiert worden sind.

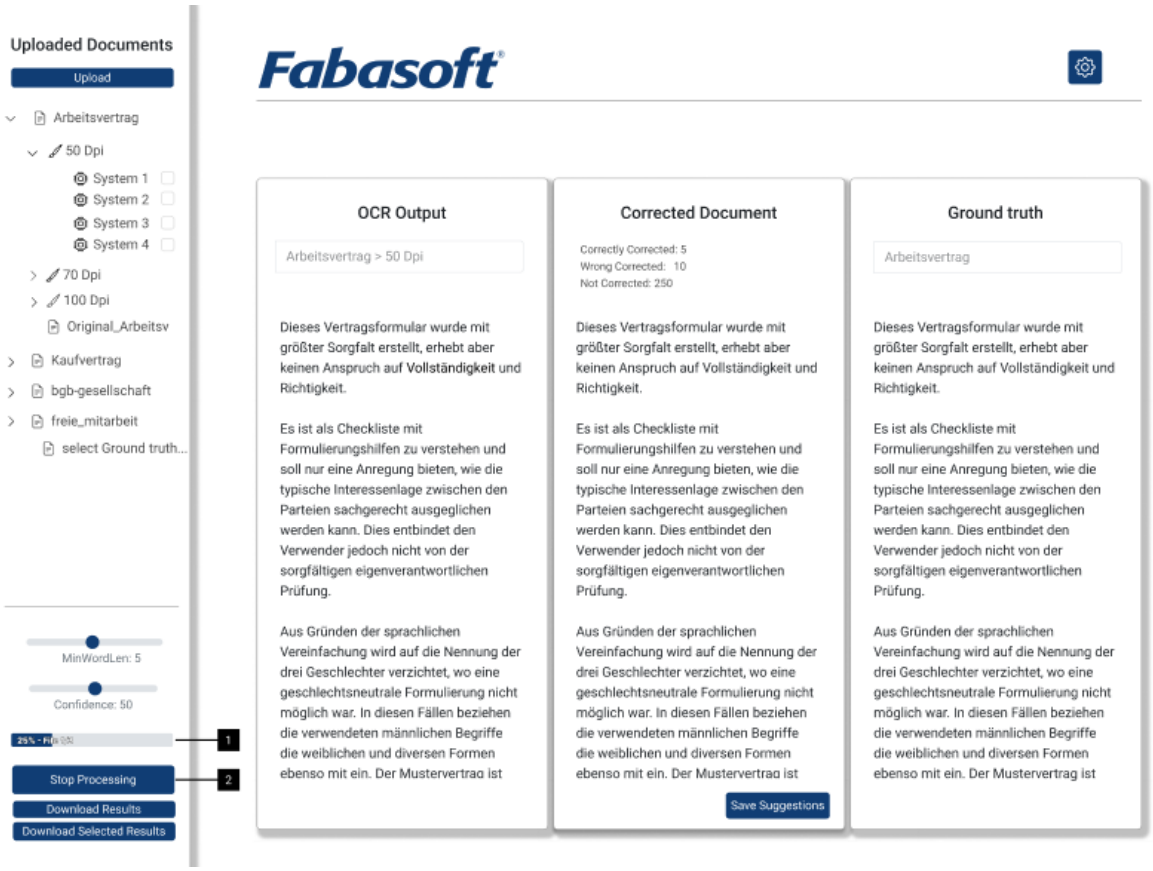


Abbildung 9.4 - Mockup Frontend (File Processing)

9.2.12 Stop Processing

Wird das Post-Processing gestartet, so wird der Start Processing Button zu einem Stop Processing Button (wie in Abbildung 9.4, Nummer 2 zu erkennen ist). Wird dieser vom Benutzer gedrückt, so wird der laufende Post-Processing-Prozess abgebrochen.

10. Implementierung

10.1.1 Backend Baseclass

Die Grundklasse (*SpellCorrection*) wurde konzipiert, um eine effiziente Möglichkeit zur Implementierung mehrerer Korrektursysteme zu bieten. Diese Klasse ist als abstrakte Klasse deklariert und dient als Vorlage für die abgeleiteten Klassen. Der Ansatz besteht darin, dass die abgeleiteten Klassen nur die notwendigen Methoden implementieren müssen, um spezifische Korrektursysteme zu realisieren. Ein weiteres Merkmal ist die Ableitung der Basisklasse von der Klasse "*threading.Thread*". Diese Ableitung bewirkt, dass alle abgeleiteten Klassen als Threads behandelt werden. Dies ermöglicht die parallele Verarbeitung mehrerer Dateien durch die gleichzeitige Ausführung der zugehörigen Korrektursysteme in separaten Threads.

10.1.1.1 Abstrakte Methoden

Die in Abbildung dargestellte Methode *correct*, welche dazu genutzt wird, um die hochgeladenen Daten zu verbessern, muss von jedem individuellen Korrektursystem überschrieben werden. Es ist zu beachten, dass diese Methode ein Objekt vom Typ OCRFile zurückgibt. OCRFile wird verwendet, um die korrigierten Daten effizient zu speichern und zu verwalten.

```
@abstractmethod
def correct(self, ocr_processing_information: OCRProcessingInformation) ->
OCRFile:
    pass


@abstractmethod
def get_current_progress(self) -> int:
    pass
```

Abbildung 10.1 - Base Class

Zusätzlich zur *correct*-Methode existiert eine weitere abstrakte Methode, die von den abgeleiteten Klassen implementiert werden kann bzw. soll. Diese Methode ist dafür verantwortlich, den aktuellen Fortschritt des Korrekturprozesses zurückzugeben. Der Fortschritt wird als Integer-Wert repräsentiert und ermöglicht eine dynamische Überwachung des Verbesserungsvorgangs. Bei den zurückgegebenen Werten handelt es sich um den aktuellen Index, bei welchem Wort sich der Verbesserungsfortschritt gerade befindet.

Die Architektur der Base Class bietet somit eine strukturierte Grundlage für die Implementierung verschiedener Korrektursysteme. Die Verwendung von Abstraktion und die Integration von Thread-Funktionalitäten erleichtern die Entwicklung und parallele Ausführung, während die Rückgabewerte der Methoden eine effiziente Verwaltung der korrigierten Daten und des Fortschritts ermöglichen. Diese Herangehensweise fördert eine modulare und flexible Gestaltung des Systems.

10.1.1.2 Methoden

Die Funktion *split_in_sentences* dient dazu, eine Sammlung von Wörtern, die als OCR-Erkennungsergebnisse vorliegen, in Sätze zu unterteilen. Die Wörter sind in einer internen Struktur gespeichert, die durch die Methode *get_base_data* zugänglich gemacht wird.

Die Kernfunktionalität dieser Methode besteht darin, die Liste von Wörtern durchzugehen und diese anhand von Satzzeichen in Sätze zu unterteilen. Ein Satz wird als eine Liste von OCR-Word-Objekten definiert. Die spezifischen Satzzeichen, die zur Identifizierung des Satzendes verwendet werden, umfassen Punkte, Doppelpunkte, Ausrufezeichen, Fragezeichen und Bindestriche. Wann immer eines dieser Satzzeichen am Ende eines Wortes erkannt wird, interpretiert die Methode dies als das Ende eines Satzes. Daraufhin wird der aktuelle Satz (eine Liste von OCRWord-Objekten) zu einer übergeordneten Liste von Sätzen hinzugefügt, und die Liste für den nächsten Satz wird zurückgesetzt.

Zum Abschluss des Durchlaufs durch alle Wörter überprüft die Funktion, ob der letzte Satz (der möglicherweise nicht mit einem der definierten Satzzeichen endet) noch unvollständige Wörter enthält. Ist dies der Fall, wird dieser ebenfalls der Liste der Sätze hinzugefügt. Das Ergebnis der Funktion ist eine verschachtelte Liste, wobei jede Liste einen Satz repräsentiert, der aus einer Sequenz von OCRWord-Objekten besteht. Diese Methode ermöglicht eine strukturierte Aufbereitung von OCR-Textdaten für weitere Verarbeitungsschritte, indem sie die Textdaten in logische Einheiten (Sätze) unterteilt.

```
def split_in_sentences(self) -> list[list[OCRWord]]:
    sentences = []
    sentence = []
    for ocrword in self.__ocr_processing_information.get_base_data():
        sentence.append(ocrword)
        if str(ocrword.get_word()).endswith(".") or
str(ocrword.get_word()).endswith(":") or str(
                ocrword.get_word()).endswith("!") or
str(ocrword.get_word()).endswith("?") or str(
                ocrword.get_word()).endswith("-"):
            sentences.append(sentence)
            sentence = []
    if len(sentence) != 0:
        sentences.append(sentence)

    return sentences
```

Abbildung 10.2 - split_in_sentences

10.1.2 Statistiken

10.1.2.1 Allgemein

Die Klasse OCRFileStatistics wurde entwickelt, um die Analyse und Bewertung von Texten, die durch optische Zeichenerkennung (OCR) erfasst wurden, zu unterstützen und zu optimieren. Der Hauptzweck dieser Klasse ist es, eine detaillierte Statistik über die Qualität und Genauigkeit der OCR-Ergebnisse im Vergleich zu einem Referenztext (Ground Truth) zu erstellen und bereitzustellen. Durch die Sammlung und Auswertung dieser Daten können Benutzer die Effektivität von OCR-Korrekturen beurteilen und potenzielle Verbesserungsmaßnahmen identifizieren.

10.1.2.2 Umsetzung

Der Konstruktor der Klasse fordert einige Parameter, welche übergeben werden müssen, um die OCR-Datei auswerten zu können.

```
def __init__(self, words_to_change=None, corrected_words=None,
             ground_truth_words=None, confidence=0,
                          min_word_length=0, wrongly_corrected=None,
             correctly_corrected=None, not_corrected_words=0,
                          statistics: dict[str, dict[str, int]] = None,
             evaluation_statistics=None, cs_statistics=None):
```

Abbildung 10.3 - Statistikberechnung Konstruktor

Die *evaluate*-Methode innerhalb der *OCRFileStatistics*-Klasse spielt eine zentrale Rolle bei der Bewertung und Statistikerstellung der OCR-Ergebnisse. Die Methode startet mit einer Überprüfung, ob bereits eine Auswertung der Korrekturen vorhanden ist, damit Leistung gespart werden kann, und die gleichen Statistiken bei erneutem Anfragen nicht noch einmal berechnet werden. Die Methode durchläuft parallel die Listen der Ground-Truth-Wörter (*gt_words*) und der korrigierten OCR-Wörter (*ocr_words*). Für jedes Wortpaar wird überprüft, ob der Status des OCR-Wortes CORRECTED ist und ob das korrigierte Wort mit dem entsprechenden Ground-Truth-Wort übereinstimmt.

Um die beiden Listen vergleichen zu können, müssen beide zuerst auf die gleiche Länge gebracht werden, wofür die *align_lists*-Methode existiert. Der Hauptzweck dieser Methode ist es, die Listen der korrigierten OCR-Wörter (*ocr_words*) und der Ground-Truth-Wörter (*gt_words*) so aneinander auszurichten, dass eine direkte Vergleichbarkeit der Wörter ermöglicht wird. Dazu wird der Needleman-Wunsch-Algorithmus angewendet, um die optimale Ausrichtung zwischen den Wortpaaren zu finden.

```
alignment: needle.NeedlemanWunsch[str] = needle.NeedlemanWunsch(gt_seq,
ocr_words_seq)
alignment.align()
gt_al, ocr_words_al =
alignment.get_aligned_sequences(core.AlignmentFormat.list)
```

Abbildung 10.4 - Anwendung des Needleman-Wunsch-Algorithmus

Fehlende Wörter oder zusätzlich eingefügte Wörter werden durch einen speziellen Platzhalten, welcher in der „gap_char“ Variable festgelegt wird, repräsentiert. Zum Abschluss werden die neu erstellten Listen *gt_words* und *ocr_words_copy*, die jetzt die ausgerichteten OCR-Word-Objekte enthalten, den Klassenvariablen *__gt_words_al* und *__ocr_words_al* zugewiesen.

Nachdem alle Wortpaare bewertet wurden, erstellt die Methode Statistiken über die verschiedenen Bewertungsstatus. Dies erfolgt durch Zählen, wie oft jeder Status in der Liste der korrigierten OCR-Wörter auftritt. Das Ergebnis wird in zwei Dictionaries gespeichert: *__evaluation_statistics* für die Bewertungsstatistik und *__cs_statistics* für die Statistik des Korrektursystems selbst. Diese beinhalten Informationen darüber, wie viele Wörter korrekt korrigiert,

fälschlicherweise korrigiert, nicht korrigiert wurden und wie viele zusätzliche oder fehlende Wörter es gab.

Am Ende der *evaluate*-Methode werden die erstellten Statistiken in einem Dictionary *__statistics* zusammengefasst, das sowohl die Evaluations- als auch die Korrektursystemstatistiken enthält. Dieses Dictionary wird dann zurückgegeben und kann für weitere Analysen oder zur Anzeige der Ergebnisse verwendet werden.

10.1.2.3 OCRWord Status

Die Klasse OCRStatus ist Teil eines Systems zur Verarbeitung und Bewertung von OCR-Ergebnissen. Sie definiert eine Reihe von Zuständen, die sowohl den aktuellen Bearbeitungsstatus eines Wortes im OCR-Prozess (OCRStatus) als auch das Ergebnis der Bewertung dieses Wortes (EvaluationStatus) nach einem Vergleich mit einem Referenztext (der Ground Truth) beschreiben.

1. **OCRStatus**

- **CS_NOT_FOUND**: Das Korrektursystem konnte kein passendes Ersatz- oder Korrekturwort finden.
- **ONLY_SPECIAL_CHAR**: Das Wort besteht ausschließlich aus Sonderzeichen. In vielen Fällen sind solche Wörter für die Textverbesserung nicht relevant und können spezielle Behandlung erfordern.
- **GAP_CHAR**: Ein Platzhalter für fehlende Wörter. Dieser Status wird verwendet, um die Abwesenheit eines Wortes an einer bestimmten Stelle im Text zu markieren, was auf ein Erkennungsproblem hinweist.
- **SPACIER**: Dieser Status wird verwendet, um einen Absatz zu markieren, was jedoch keine weitere Auswirkung im Backend hat. Dies ist wichtig für die richtige Visualisierung im Frontend.
- **TO_CORRECT**: Ein vorläufiger Status, der darauf hinweist, dass ein Wort korrigiert werden sollte. Dieser Status ist Teil eines Überarbeitungsprozesses und deutet darauf hin, dass weitere Analyse oder Korrektur notwendig ist.

- **CS_CORRECT**: Das Korrektursystem hat das Wort als richtig erkannt und hat keine Veränderung vorgenommen.
- **CORRECTED**: Das Wort wurde korrigiert. Dieser Status zeigt an, dass eine Korrektur durchgeführt wurde.
- **SKIP**: Das Wort wird im Korrekturvorgang übersprungen. Dieser Status wird bei jenen Wörtern gesetzt, welche sich über der gesetzten Konfidenz befinden und daher keine Korrektur benötigen.
- **INITIAL**: Der ursprüngliche Status eines Wortes, bevor es irgendeinen Korrektur- oder Bewertungsprozess durchlaufen hat.
- **ERROR**: Ein Fehlerstatus, der auf ein Problem während des OCR- oder Korrekturprozesses hinweist.

2. **EvaluationStatus**

- **OCR_CORRECT**: Das Wort wurde mit der Ground Truth verglichen und stimmt mit dieser auch überein.
- **OCR_FALSE**: Das Wort wurde mit der Ground Truth verglichen, stimmt jedoch nicht mit ihr überein.
- **NOT_EVALUATED**: Das Wort wurde noch nicht bewertet. Es hat noch keinen Vergleich mit der Ground Truth durchlaufen oder wurde aus dem Bewertungsprozess ausgeschlossen.
- **OCR_MISSING_WORD**: OCR hat ein Wort übersehen oder nicht erkannt. Dieser Status zeigt an, dass im OCR-Text ein Wort fehlt, das in der Ground Truth vorhanden ist.
- **OCR_ADDITIONAL_WORD**: OCR hat ein zusätzliches Wort eingefügt. Dies weist darauf hin, dass im OCR-Text ein Wort vorhanden ist, das in der Ground Truth nicht existiert.

Diese Zustände ermöglichen eine detaillierte Analyse und Nachverfolgung der Korrekturen und Bewertungen von OCR-Texten. Sie helfen dabei, die Qualität der OCR-Ergebnisse zu verbessern, indem sie Aufschluss darüber geben, welche Wörter korrigiert wurden, welche Korrekturen erfolgreich waren und wo potenzielle Fehler im OCR-Prozess liegen.

10.1.3 Backend Unit-tests

10.1.3.1 Allgemein

Unit-tests wurden eingesetzt, um die Zuverlässigkeit und Funktionalität sicher zu stellen. Unit-tests sind spezielle Tests, die kleine Teile des Codes isoliert überprüfen, um Fehler frühzeitig zu identifizieren und zu beheben. Durch die Anwendung von Unit-tests können wir systematisch verifizieren, dass jede Komponente wie vorgesehen funktioniert, bevor sie in das größere System integriert wird. Dieser Ansatz trägt dazu bei, die Qualität des Endprodukts zu erhöhen und die Wartung der Software zu vereinfachen. Unit-tests ermöglichen es auch, Änderungen am Code vorzunehmen oder neue Funktionen hinzuzufügen, während gleichzeitig sichergestellt wird, dass bestehende Funktionen nicht negativ beeinflusst werden. Somit bilden sie eine fundamentale Grundlage für die Entwicklung robuster, zuverlässiger und wartbarer Software.

10.1.3.2 Umsetzung

Um unsere Unit-tests verwenden zu können muss zuerst alles dafür vorbereitet werden. Dazu laden wir ein Testdokument, welches mittels Dateipfad in unserer config-file hinterlegt ist. Dieses Dokument wird dazu benötigt, um ein Korrektursystem durchlaufen lassen zu können. Des Weiteren wird dazu auch noch eine Konfidenz benötigt, damit das System richtig arbeiten kann.

```
@classmethod
def setUpClass(cls):
    file = open(config.HOCR_TESTFILE, "r")

    file_manager = FileParser(file.read())
    file.close()
    ocr_processing_information = file_manager.parse_hocr_file()
    cls.__ocr_processing_information = ocr_processing_information
    cls.__ocr_processing_information.set_confidence(80)
```

Abbildung 10.5 - Unit-test Setup

Wenn alle oben beschriebenen Vorgänge abgeschlossen sind, kann der eigentliche Test starten. Dazu werden alle Implementierungen der SpellCorrection Klasse durchlaufen, um jedes System zu testen. In jedem Durchlauf wird im ersten Schritt der Status des Korrektursystems überprüft. Danach folgt die eigentliche Verbesserung des Dokuments worauf hin dann überprüft wird, ob auch eine verbesserte Version des Dokuments zurückkommt. Anschließend

wird jedes Wort der Verbesserung durchlaufen und auf verschiedene Zustände überprüft. Dazu zählen:

- Ursprüngliches Wort ist „None“
- Ursprüngliches Wort ist ein leerer String
- Korrigiertes Wort ist „None“
- Korrigiertes Wort ist ein leerer String

Wenn dies fehlerfrei durchläuft, wird anschließend wieder der Status des Korrektursystems überprüft. In diesem Fall muss von dem System aus gesagt werden, dass die Verbesserung abgeschlossen ist. Abschließend wird noch überprüft, ob die *split_in_sentences*-Methode des Korrektursystems auch richtig funktioniert.

```
@ordered
def test_correction(self):
    for correction_system in SpellCorrection.__subclasses__():
        self.__correction_system =
correction_system(self.__ocr_processing_information)

        self.assertEqual(False,
self.__correction_system.finished_correction(),
                         "The status of the correction system should be False
| " + self.__correction_system.__class__.__name__)

        hocr_file =
self.__correction_system.correct(self.__ocr_processing_information)

        self.assertIsNot(hocr_file, None,
                         "The correction system did not return any result | "
+ self.__correction_system.__class__.__name__)

        words = self.__ocr_processing_information.ocr_file_to_words()

        for word in words:
            self.assertIsNot(word.get_word(), "",
                             "There is an OCRWord with no content in the base
word | " + self.__correction_system.__class__.__name__)
            self.assertIsNot(word.get_word(), None,
                             "There is an OCRWord with None as base word | " +
self.__correction_system.__class__.__name__)
            self.assertIsNot(word, "",
                             "The corrected word has no content. Make sure to
set it to the base word if there is no correction | " +
self.__correction_system.__class__.__name__)
            self.assertIsNot(word.get_corrected_word(), None,
                             "The corrected word is None. Make sure to set it
to the base word if there is no correction | " +
self.__correction_system.__class__.__name__)

        self.assertEqual(True, self.__correction_system.finished_correction(),
                         "The status of the correction system should be True |
" + self.__correction_system.__class__.__name__)

        sentences: list[list[OCRWord]] =
self.__correction_system.split_in_sentences()

        self.assertEqual(2, len(sentences),
                         "The split method does not work correctly | " +
self.__correction_system.__class__.__name__)
```

Abbildung 10.6 - test_correction Unit-test

10.1.4 API-Anpassungen

Die API existiert bereits vom Projekt des Vorjahres aus dem Projektentwicklungsunterricht und wird im Rahmen der Diplomarbeit um einige Endpoints erweitert und modifiziert. Es werden Endpoints auf mehrere Endpoints aufgeteilt, Endpoints um zusätzliche Parameter erweitert und zusätzliche Endpoints für die neuen Funktionalitäten der Diplomarbeit definiert.

Während der Planung und Diskussion der Funktionalitäten der Diplomarbeit wurden auch Funktionen und Endpoints definiert, welche nicht Teil der Diplomarbeit sind. Da diese aber bereits spezifiziert waren, wurden sie ebenfalls mittels des OpenAPI-Generator Plugins mit den anderen API-Endpoints mitgeneriert, jedoch weder im Frontend verwendet noch im Backend implementiert. Da das Schulprojekt der fünften Klasse auf dieser Diplomarbeit aufsetzt, werden diese Endpoints in diesem Projekt ausimplementiert und verwendet. Des Weiteren wird ein Wrapper verwendet, um die Code-Qualität zu steigern, und den Code wiederverwertbarer zu machen.

Die Bedeutungen von korrigiert und evaluiert sind für das Verständnis der Endpoints wichtig. Eine hocr-Datei wird als korrigiert angesehen, wenn die Korrektur durch ein Korrektursystem durchgeführt wurde. Eine hocr-Datei wird als evaluiert angesehen, wenn die Evaluierung mit der Ground-Truth fertiggestellt wurde.

Allgemein wurden in den meisten Endpunkten mehr Überprüfungen hinzugefügt, damit der Benutzer besser über Fehler informiert wird und weniger Fehler auftreten können.

10.1.4.1 Neue Endpoints

Um die neuen Funktionalitäten des Backends für das Frontend nutzbar zu machen, werden neue Endpoints definiert. Es wurden außerdem neue Endpoints definiert, welche Teile der Funktionalitäten von anderen Endpoints übernehmen.

- *get_all_correction_systems*
    - Dieser Endpoint stellt die Namen aller Korrektursysteme bereit, damit ein Client diese abfragen kann. Somit kann ein Client abfragen, welche Korrektursysteme im Backend implementiert sind, damit er diese später auswählen und benutzen kann.
- *get_evaluted_file_as_hocr*
    - Dieser Endpoint ist ein Teil des ursprünglichen *get_hocr*-Requests. Da dieser alte Request sich um die hocr-Dateien nach dem Korrigieren und nach dem Evaluieren gekümmert hat, wurde das Bereitstellen der evaluierten hocr-Datei in einen eigenen Endpoint ausgelagert und ist nun über diesen Request verfügbar. Dieser Request übernimmt im Wesentlichen die gleiche Funktionalität, wie der *get_corrected_file_as_hocr*-Request, jedoch wird im Code auf den Korrekturfortschritt geachtet.

- *get_progress_of_correction*
    - Dieser Endpoint ist ein Teil der ursprünglichen *get_hocr*-Requests und des *get_json*-Requests. Diese haben zuvor, wenn das Korrigieren nicht fertig war, den Fortschritt der Korrektur zurückgegeben. Dies wurde in den Request *get_progress_of_correction* ausgelagert. Dieser gibt dem Client nun den Korrekturfortschritt zurück, ohne die fertig-korrigierte Datei zurückzugeben, falls diese fertig ist.
- *get_progrss_of_evaluation*
    - Dieser Endpoint ist ein Teil des ursprünglichen *get_json_stats*-Requests. Dieser hat zuvor, wenn das Evaluieren nicht fertig war, den Fortschritt der Evaluierung zurückgegeben. Dies wurde in den Request *get_progress_of_evaluation* ausgelagert. Dieser gibt dem Client nun den Evaluierungsfortschritt zurück, ohne die fertig-evaluierte Datei zurückzugeben, falls diese fertig ist.

10.1.4.2 Decorator

Die Überprüfung, ob die UUID valide ist, wird in einen Wrapper ausgelagert, um die Wiederverwendbarkeit zu erhöhen, die Code-Qualität zu erhöhen und die Anzahl der Code-Zeilen zu reduzieren. Ein Wrapper in Python wird auch Decorator genannt und ist eine Funktion, welche eine andere Funktion als Parameter annimmt, in einer internen *wrapper*-Methode Code ausführt und anschließend diese *wrapper*-Methode zurückzugeben. Durch Annotations kann dieser Decorator dann vor einem Methodenaufruf ausgeführt werden, bevor der eigentliche Methodencode beginnt. (geeksforgeeks wrapper, 2024)

In diesem Decorator wird die UUID, welche bei den meisten Endpoints benötigt wird, auf ihre Validität überprüft. Wenn diese nicht korrekt ist, wird ein Error an den Client zurückgegeben, und sonst normal der Code der Methode, welche mit dem Decorator annotiert ist, ausgeführt.

```
def validate_uuid(func):
    def wrapper(*args, **kwargs):
        file_id = kwargs.get('file_id')
        if __processing.validate_uuid(file_id) is False:
            dict_to_return = {'code': 'Error', 'message': 'Invalid file
id'}
            return jsonify(dict_to_return)
        return func(*args, **kwargs)
    return wrapper
```

Abbildung 10.7 - validate_uuid-Decorator

```
@validate_uuid
def get_corrected_file_as_hocr(file_id):  # noqa: E501
    # code...
```

Abbildung 10.8 - Methode, welche mit dem Decorator annotiert wird

In dem obigen Beispiel wird abgebildet, wie solch eine Annotation in Python aussieht.

10.1.4.3 Modifizierte Endpoints

- *get_hocr* und *get_json*
  - Beide Endpunkte wurden auf *get_corrected_file_as_<type>* umbenannt.
  - Beide mit *valide_uuid*-Decorator versehen
- *get_json_stats*
  - Endpunkt auf *get_evaluated_file_as_json* umbenannt.
  - Mit *valide_uuid*-Decorator versehen
- *post_hocr*
  - Zusätzlicher Parameter *system.* Dieser Parameter bestimmt, welches Korrektursystem für die Verbesserung der hochgeladenen hocr-Datei ausgewählt und verwendet werden soll. Wenn bei *system* ein Korrektursystem ausgewählt wird, welches nicht existiert, dann wird die Korrektur nicht gestartet. Die Systeme können über den *get_all_correction_systems*-Endpoint abgefragt werden.

10.1.4.4 Endpoints mit Code-Veränderungen

- *post_ground_truth*
  - Mit *valide_uuid*-Decorator versehen
- *get_stats*
  - Mit *valide_uuid*-Decorator versehen
  - Mehr Werte in den Statistiken

10.1.4.5 Gleiche Endpoints

- *suggestions*
  - Mit *valide_uuid*-Decorator versehen
  - Keine Änderungen in der Methode selbst

10.1.4.6 Neue unbenutzte Endpoints

- *stop_post_processing*
  - Soll die Korrektur einer Datei stoppen
  - Nicht verwendet in der Diplomarbeit

10.1.5 Phunspell

Als Grundlage für die Implementierung des Korrektursystems Phunspell dient die Grundklasse. Diese ist im Kapitel 11.1.1 Backend Basisklasse beschrieben. Diese Klasse gibt die Methode *correct* vor, welche implementiert werden muss.

In der Methode wurde Phunspell mit der Phunspell Bibliothek eingebunden. Diese Bibliothek wurde mit Pip installiert. Weitere Infos zu Pip sind im Kapitel 8.2.2 Pip zu finden. Um ein Objekt von dieser Bibliothek zu initialisieren, muss *phunspell.Phunspell(„Sprache des Wörterbuches“)* aufgerufen werden. Als Parameter wird hierbei die Sprache des Wörterbuches übergeben. Mit dem erzeugten Objekt *.suggest(„Wort“)* werden nun Vorschläge für das Wort, als Liste zurückgegeben. Es folgt ein Beispiel für den beschriebenen Code.

```
import phunspell

pspell = phunspell.Phunspell('de_AT')
```

Abbildung 10.9 - Phunspell initialisieren

Wenn ein Wort korrigiert werden muss, das heißt, wenn es unter der vorgegebenen Konfidenz liegt, wird zunächst mit *Objekt.lookup(„Wort“)* überprüft, ob das Wort im Wörterbuch vorhanden und somit bereits korrekt ist. Wenn das Wort bereits korrekt ist, wird der Status des Wortes auf *CS_CORRECT* gesetzt. *CS_CORRECT* bedeutet, dass das Korrektursystem davon ausgeht, dass dieses Wort bereits korrekt ist. Zusätzlich wird das gleiche Wort als korrigiertes Wort gesetzt, mit *Objekt.set_corrected_word(„Wort“)*, da angenommen wird, dass es bereits korrekt ist. Der beschriebene Code ist unten dargestellt.

```
spelled_correctly = pspell.lookup(word.get_word())

if spelled_correctly:
    word.set_status(OCRStatus.CS_CORRECT)
    word.set_corrected_word(word.get_word())
```

Abbildung 10.10 - Phunspell lookup und status setzen

Wenn das Wort nicht im Wörterbuch ist, werden Korrektur Vorschläge, mit *Objekt.suggest(„Wort“)*, generiert. Der wahrscheinlichste Vorschlag wird als Korrektur gesetzt. Das passiert mit der Zeile *corrected = suggestion_list.pop(0)*. Die anderen Vorschläge werden als weitere Wortvorschläge hinzugefügt, was mit dem Code *word.set_other_candidates(suggestion_list)* geschieht. Die weiteren Vorschläge werden dann im Frontend, wie in der Abbildung 11.8 - Suggestions zu sehen, angezeigt.

10.1.6 WordPrediction

Die Klasse WordPredictionCorrection basiert auf der Grundklasse. Diese ist im Kapitel 11.1.1 Backend Basisklasse beschrieben. Die Basisklasse gibt die Methode *correct* vor, welche implementiert werden muss.

Um in dieser Methode das Hugging Face Modell „bert-base-german-cased“ benutzen zu können, wird eine Schnittstelle verwendet, welche von Hugging Face zur Verfügung gestellt wird. Diese Schnittstelle bietet eine Möglichkeit diese Modelle einfach zu benutzen, ohne tief in technische Details eintauchen zu müssen. In der *pipeline()* Methode, von Hugging Face Transformers, muss der Name des Modells festgelegt werden. Zurück gegeben wird ein initialisiertes Objekt. Es folgt ein Beispielcode, womit ein Objekt initialisiert wurde.

```
model_name = "dbmdz/bert-base-german-cased"
fix_spelling = pipeline(model=model_name)
```

Abbildung 10.11 - WordPrediction initialisieren

Der Code durchläuft jedes *OCRWord*, welches in der Liste *ocr_sentence* enthalten ist. Diese Listen werden von der Methode *split_in_sentences* zurückgegeben, welche im Kapitel 10.1.1.2 näher beschrieben ist.

Um das Verbessern von bereits korrekten Wörtern zu vermeiden, wird zunächst mit mit *Objekt.lookup(„Wort“)* überprüft, ob das Wort im Wörterbuch von Phunspell vorhanden und somit bereits korrekt ist. Wenn dies nicht der Fall ist, wird dem Wort der Status *SHOULD_CORRECT* zugewiesen, um zu kennzeichnen, dass dieses Wort verbessert werden muss. Falls das Wort jedoch korrekt ist, wird das Wort mit dem Code *ocrWord.set_status(OCRStatus.CS_CORRECT)* als bereits richtig markiert und anschließend wird mit dem nächsten Wort fortgefahren. Dies wiederholt sich so lange, bis alle Wörter in diesem Satz überprüft wurden.

Anschließend durchläuft das Programm den Satz erneut, sofern mindestens ein Wort in diesem korrigiert werden muss. Wenn der Satz erneut durchlaufen wird und er nun den Status SHOULD_CORRECT hat, wird anstelle des Worts ein [MASK]-Token in den Satz eingefügt. Wenn der erste [MASK]-Token eingefügt wurde, wird *replaced_first*, dem wert T*rue* zugewiesen, damit nur ein [MASK]-Token in den Satz eingefügt wird, da das Modell nur einen [MASK]-Token pro Aufruf verarbeiten kann. Das Beschriebene ist in der Abbildung 10.10 widergespiegelt.

```
for ocrWord in ocr_sentence:
    if replaced_first:
        sentence += ocrWord.get_word() + " "
    else:
        if ocrWord.get_status().__eq__(OCRStatus.SHOULD_CORRECT):
            sentence += "[MASK] "
            replaced_first = True
            word_to_correct = ocrWord
        else:
            sentence += ocrWord.get_word() + " "
```

Abbildung 10.12 - WordPrediction Input Satz zusammenbauen

Anschließend wird mit *fix_spelling(sentence)* ein Satz korrigiert. der Satz kann zum Beispiel wie folgt aufgebaut sein: „Ich [MASK] Simon“. Unbedingt zu beachten ist, dass nur ein [MASK]-Token enthalten sein darf.

Zurückgegeben wird eine JSON, die nach der Wahrscheinlichkeit absteigend sortiert ist. Das erste Element in der JSON repräsentiert das ähnlichste korrigierte Wort und enthält Informationen wie „score“ (Wahrscheinlichkeit), „token“ (Token-ID), „token_str“ (vorhergesagte Wort) und „sequence“ (korrigierter Satz). Die übrigen Elemente in der JSON sind Vorschläge (Suggestions) und enthalten ebenfalls die oben genannten Informationen.


```
[
  {
    "score": 0.4549489915370941,
    "token": 1089,
    "token_str": "bin",
    "sequence": "Ich bin Simon."
  },
  {
    "score": 0.3452669382095337,
    "token": 15996,
    "token_str": "heiße",
    "sequence": "Ich heiße Simon."
  },
  {
    "score": 0.028360676020383835,
    "token": 285,
    "token_str": "war",
    "sequence": "Ich war Simon."
  },
  {
    "score": 0.02146674320101738,
    "token": 18479,
    "token_str": "suche",
    "sequence": "Ich suche Simon."
  },
  {
    "score": 0.015545304864645004,
    "token": 9484,
    "token_str": "kenne",
    "sequence": "Ich kenne Simon."
  }
]
```


Abbildung 10.13 - bert base german cased response

10.1.7 Frontend Sidebar

Um das Anzeigen von mehreren Dateien zu ermöglichen, wurde im Frontend eine Sidebar implementiert. Die Sidebar Komponente registriert beim Initialisieren zuerst einen Change Listener in der „*FileStorage*“ Singleton Klasse (genauer beschrieben im Kapitel zur Umsetzung des Multifileuploads 10.1.8.2), um neu hochgeladene Dokumente anzeigen zu können. Die

Komponente speichert sich ebenfalls beim Initialisieren die verfügbaren Korrektursysteme, welche mittels API Abfrage vom Backend abgefragt werden.

10.1.7.1 Sidebar Komponentenstruktur

Die hochgeladenen Dokumente sollen in der Sidebar in einer Tree View angezeigt werden, sodass der Benutzer unserer Applikation die untergeordneten Qualitätsstufen sowie Korrektursysteme gegebenenfalls einklappen kann. Um dies zu bewerkstelligen, wurden eigene Komponenten für jede Stufe dieser Hierarchie erstellt, welche in Abbildung 10.14 zu erkennen sind. Hierbei handelt es sich bei der obersten Komponente „*SidebarItem.js*“ um die Überkategorie aller hochgeladenen Qualitätsstufen eines Dokuments. Wird diese Komponente ausgeklappt, kann der Benutzer alle hochgeladenen Qualitätsstufen erkennen. Wird diese Komponente erneut ausgeklappt, werden alle zur Verfügung stehenden Korrektursysteme angezeigt.

```
|- SidebarItem.js
|----- SidebarDpiEntry.js
|----------SidebarCorrectionSystemEntry.js
|----------SidebarCorrectionSystemEntry.js
|----------SidebarCorrectionSystemEntry.js
|----- SidebarDpiEntry.js
|----------SidebarCorrectionSystemEntry.js
|----------SidebarCorrectionSystemEntry.js
|----------SidebarCorrectionSystemEntry.js
|- SidebarItem.js
|----- SidebarDpiEntry.js
|----------SidebarCorrectionSystemEntry.js
…
```

Abbildung 10.14 - Komponentenstruktur Sidebar

Sobald eine „*SidebarCorrectionSystemEntry*“ Komponente einen Klick registriert, so wird das übergeordnete HOCRFile, das Korrektursystem sowie die dazugehörige Ground Truth Datei in die Singleton-Klasse welche in Abbildung 10.15 zu erkennen ist, gespeichert. Anschließend wird ein Change Listener, welcher in der Komponente, die den Inhalt der HOCR-Datei anzeigt, registriert ist, aufgerufen. Dies führt dazu, dass nun das ausgewählte Dokument für den Benutzer ersichtlich ist.

```
class CurrentFileToShow {

    constructor() {
        this._hocrFile = null;
        this._groundTruth = null;
        this._correctionSystem = "-";
        this._correctedFile = null;
        this.hocrChangeListener = [];
        this._downloadFunction = () => { };
    }
}
```

Abbildung 10.15 - CurrentFileToShow

10.1.8 Frontend Multi-file-upload

Um das Testen von mehreren Dateien für den Benutzer so einfach wie möglich zu gestalten, wurde ebenfalls die Möglichkeit implementiert, einen gesamten Ordner, den sogenannten „Dataset Folder“ hochzuladen. Dazu muss im Upload Dialogfenster, welches in Abbildung 10-3 zu erkennen ist, die Option „Dataset Folder“ ausgewählt werden. Nun kann der Benutzer mittels Document Browser einen Ordner hochladen.

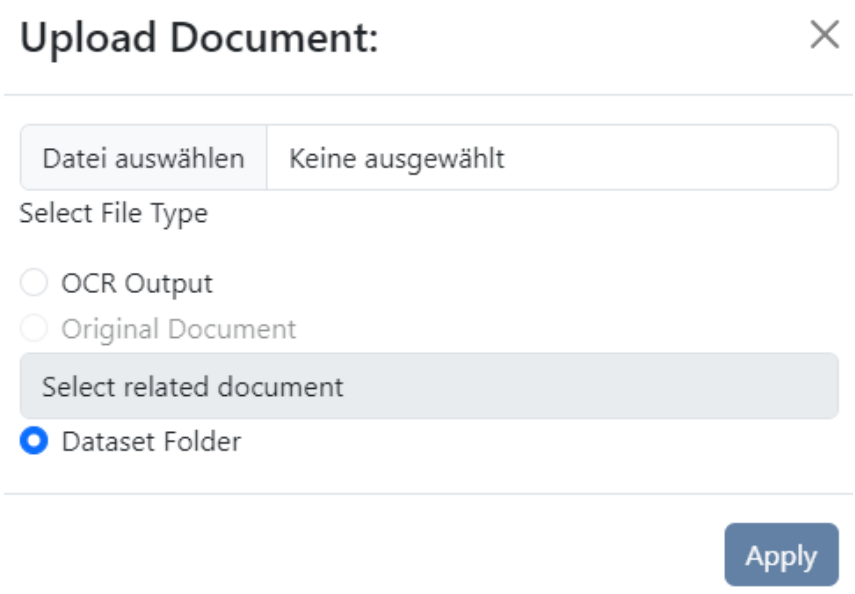


Abbildung 10.16 - Upload Dialogfenster

10.1.8.1 Struktur Dataset Folder

Damit erkannt werden kann, welche Ground-Truth zu welchen *.hOCR Dokumenten passt, sowie zur Feststellung deren dpi-Werte muss der hochgeladene Ordner einer festgelegten Struktur entsprechen.

[dpi]_Dateiname

Ein Beispiel für die Struktur eines gültigen Dataset Folder kann in Abbildung 10.17 erkannt werden. Hierzu ist anzumerken, dass die Ordnerstruktur für die Frontend Applikation keine Rolle spielt.

```
datasetFolder
├──dpi_100#rotated_0#blur_radius_0.5
│  └──hocr
│     dpi100_arbeitsvertrag_gfb.hocr
│     dpi100_Arbeitsvertrag_Minijob_kostenlos.hocr
│     dpi100_arbeitsvertrag_ohne_tarifbindung.hocr
│     dpi100_Architektenvertrag_kostenlos.hocr
│     dpi100_Aufhebungsvertrag_Arbeitsverhaeltnis_kostenlos.hocr
│
├──dpi_150#rotated_0#blur_radius_0
│  └──hocr
│     dpi150_arbeitsvertrag_gfb.hocr
│     dpi150_Arbeitsvertrag_Minijob_kostenlos.hocr
│     dpi150_arbeitsvertrag_ohne_tarifbindung.hocr
│     dpi150_Architektenvertrag_kostenlos.hocr
│     dpi150_Aufhebungsvertrag_Arbeitsverhaeltnis_kostenlos.hocr
│
├──dpi_50#rotated_0#blur_radius_0.5
│  └──hocr
│     dpi50_arbeitsvertrag_gfb.hocr
│     dpi50_Arbeitsvertrag_Minijob_kostenlos.hocr
│     dpi50_arbeitsvertrag_ohne_tarifbindung.hocr
│     dpi50_Architektenvertrag_kostenlos.hocr
│     dpi50_Aufhebungsvertrag_Arbeitsverhaeltnis_kostenlos.hocr
│
├──dpi_70#rotated_0#blur_radius_0.5
│  └──hocr
│     dpi70_arbeitsvertrag_gfb.hocr
│     dpi70_Arbeitsvertrag_Minijob_kostenlos.hocr
│     dpi70_arbeitsvertrag_ohne_tarifbindung.hocr
│     dpi70_Architektenvertrag_kostenlos.hocr
│     dpi70_Aufhebungsvertrag_Arbeitsverhaeltnis_kostenlos.hocr
│
└──original
   └──hocr
      original_arbeitsvertrag_gfb.hocr
      original_Arbeitsvertrag_Minijob_kostenlos.hocr
      original_arbeitsvertrag_ohne_tarifbindung.hocr
      original_arbeitsvertrag_original.hocr
      original_Architektenvertrag_kostenlos.hocr
```

Abbildung 10.17 - Beispiel Dataset Folder Struktur

10.1.8.2 Umsetzung des Multifileuploads

Wird ein Dataset Folder hochgeladen, wird durch die Dateinamen iteriert, wie in Abbildung 10.18 zu erkennen ist. Danach wird der Dateiname jeweils bei einem „_“ getrennt. Es wird nun geprüft, ob sich mindestens ein _ in dem Dateinamen befindet, da er ansonsten nicht der festgelegten Struktur entspricht. Es wird im Anschluss mittels Regular Expression überprüft, ob der DPI-Wert aus Zahlen besteht. Ist dies der Fall, so wird der dpi Wert dementsprechend gesetzt. Ansonsten bekommt der DPI-Wert den Wert null zugewiesen.

```
uploadedFolder.forEach(file => {
      const parts = file.name.split('_');
      if (parts.length >= 2) {
            const dpiString = parts[0];
            const newFilename = parts.slice(1).join('_');
            const numbers = dpiString.match(/\d+/);
            const dpiValue = numbers ? numbers[0] : null;
```

Abbildung 10.18 - Dataset Upload

Wurden die Dokumente hochgeladen, werden diese in der Singleton Klasse mit dem Namen FileStorage gespeichert. Diese enthält zwei Arrays. Eines davon speichert eine List von „*Filesets*“ (genauer beschrieben im Kapitel zum Frontend Datenmodell 10.1.8.3). Im anderen werden registrierte Changelistener auf dieses Array gespeichert. Ein solcher Changelistener auf das Fileset Array wird beispielsweise in der Sidebar registriert, da diese beim Hochladen eines Dokuments aktualisiert werden muss. Um das von uns hochgeladene Dokument nun zu speichern, wird die Methode „*addHocrFile*“ aufgerufen. Diese ist in Abbildung 10.19 dargestellt und dient dazu, ein neues Dokument zum Fileset hinzuzufügen. Da eine Datei auch in mehreren unterschiedlichen Qualitätsstufen hochgeladen werden kann. Diese Methode überprüft zuallererst, ob für die hochgeladene Datei bereits ein „*HOCRFileSet*“ Objekt existiert (genauer beschrieben im Kapitel zum Frontend Datenmodell 10.1.8.3). Ist dies nicht der Fall, so wird ein neues HOCRFileSet erstellt. Im Anschluss wird noch geprüft, ob für die hochgeladene DPI-Stufe bereits ein Dokument exisitert und im Anschluss werden noch die registrierten „*fileChangeListener*“ aufgerufen.

```
 addHocrFile(file, content, dpi = "manually uploaded", filename = file.name) {

        let currentFileset = null;

        this._filesets.forEach((fileset) => {
            if (fileset.filename === filename) {
                currentFileset = fileset;
            }
        });

        if (currentFileset == null) {
            let newArrayLength = this._filesets.push(new
HOCRFileSet(filename));
            currentFileset = this._filesets[newArrayLength - 1];
        }

        let dpiExists = false;

        currentFileset.hocrFiles.forEach((hocrFile) => {
            if (hocrFile.dpi == dpi) {
                dpiExists = true;
            }
        });

        if (!dpiExists) {
            currentFileset.hocrFiles.push(new HOCRFile(file, content, dpi,
CorrectionSystems.correctionSystems, filename));
        }

        this.fileChangeListener.forEach((func) => {
            func();
        });

    }
```

Abbildung 10.19 - addHocrFile Methode

10.1.8.3 Frontend Datenmodell

Um das Speichern der HOCR-Dateien sowie der Ground Truth Dateien zu bewerkstelligen, kommen zwei Klassen zum Einsatz. Die erste Klasse ist „*HOCRFileSet*“, welche in Abbildung 10.20 - HOCRFileSet KlasseAbbildung 10.20 zu erkennen ist. Diese wird verwendet, um eine oder mehrere HOCR-Dateien mit verschiedenen Qualitätsstufen und deren dazugehörige

Ground Truth zu speichern. Es werden hierbei der allgemeine Dateiname, ein Array mit „*HOCRFile*“ Objekten für die hochgeladenen Dokumente und ein „*HOCRFile*“ Object für die Ground Truth gespeichert.

```
class HOCRFileSet {

    constructor(filename) {
        this._filename = filename;
        this._hocrFiles = [];
        this._groundTruth = null;
    }
}
```

Abbildung 10.20 - HOCRFileSet Klasse

Zum Speichern der HOCR-Dateien wird die Klasse „*HOCRFile*“ verwendet. In dieser Klasse werden Informationen wie der DPI-Wert, der Dateiinhalt an sich sowie die korrigierte Version gespeichert. Ein weiteres Attribut ist ein Array namens „checked“, in welchem die zur Bearbeitung ausgewählten Korrektursysteme dieses Dokumentes gespeichert sind. Die Klasse enthält ebenfalls Methoden, welche beim Anhaken einer Checkbox in der Sidebar (genauer beschrieben im Kapitel zur Sidebar 11.1.4.3) ausgeführt werden, um das jeweilige Korrektursystem in das Array hinzuzufügen oder zu entfernen (in Abbildung 10.21 zu erkennen).

```
    addCheckedSystem(system) {
        this._checked.push(system);

    }

    removeCheckedSystem(system) {
        const index = this._checked.indexOf(system);
        if (index > -1) {
            this._checked.splice(index, 1);
        }
    }
```

Abbildung 10.21 - checked Methoden

Wird das Post Processing gestartet, so bekommt jedes File für jedes Korrektursystem vom Backend eine FileID, welche ebenfalls in der Klasse „*HOCRFile*“ als „*FileID*“ Objekt gespeichert

wird. Die Klasse „*FileID*“, welche in Abbildung 10.22 zu erkennen ist, speichert diese FileID zusammen mit dem verwendeten Korrektursystem.

```
class FileID {

    constructor(fileid, correctionSystem) {
        this._fileid = fileid;
        this._correctionSystem = correctionSystem;
    }

}
```

Abbildung 10.22 - FileID Klasse

Wurde nach dem Verbessern der Datei auch eine Ground Truth hochgeladen, so werden vom Backend ebenfalls Statistiken zur Verbesserung berechnet. Diese werden ebenfalls in der „*HOCRFile*“ Klasse als Stats Objekt gespeichert.

10.1.9 Post Processing Vorgang

Wird der „Start Post Processing“ Button (zu erkennen im Kapitel zur fertigen Frontend-Applikation 11.1.4.1) gedrückt, so muss das Post Processing gestartet werden. Um dies zu bewerkstelligen, gibt es eine Singleton Klasse namens „*ProcessingHandler*“ welche einige Methoden hierfür enthält. Eine davon ist die „*startPostProcessing*“ Methode (zu erkennen in Abbildung 10.23), welche zum Starten des Post Processing Vorgangs verwendet wird. Wird diese Methode aufgerufen, so wird zuallererst die „started“ Variable in der „*ProcessingHandler*“ Klasse auf true gesetzt. Somit wird der Start Post Processing Button zu einem Stop Post Processing Button und kann dazu verwendet werden, um den laufenden Vorgang abzubrechen. Danach werden alle vom Benutzer zur Verbesserung vorgemerkten Dateien im „*filesToProcess*“ Array gespeichert. Für jedes dieser Files wird dann ein Objekt der Klasse PostProcessing erstellt, welches dann zur Verbesserung verwendet wird.

```
    async startPostProcessing() {

        processingHandler.started = true;
        processingHandler.checkPostProcessingButtonEnabled();
        ProcessingHandler.filesToProcess = [];

        FileStorage.filesets.forEach((fileset) => {
            fileset.hocrFiles.forEach((hocrFile) => {

                hocrFile.checked.forEach((system) => {
                    ProcessingHandler.filesToProcess.push(new
FileToProcess(hocrFile, system, fileset.groundTruth));
                });

            });
        });

        processingHandler.numberOfProcessingFiles =
ProcessingHandler.filesToProcess.length;
        processingHandler.processedFiles = 0;

        for (const fileToProcess of ProcessingHandler.filesToProcess) {
            let postProcessing = new PostProcessing();
            await postProcessing.startPostProcessing(fileToProcess.hocrfile,
processingHandler.confidence, processingHandler.minWordLen,
fileToProcess.correctionSystem, fileToProcess.groundTruth,
processingHandler.abortController);
        }

        processingHandler.started = false;
        processingHandler.checkPostProcessingButtonEnabled();

    }
```

Abbildung 10.23 - startPostProcessing Methode

In der „*PostProcessing*“ Klasse gibt es für jeden benötigten API-Request eine Methode, welche einen Promise als Rückgabewert besitzt. Zuerst wird die „*postHocr*“ Methode aufgerufen, um das zu verbessernde Dokument hochzuladen. Die vom Backend vergebene FileID wird nun im „*HOCRFile*“ Objekt gespeichert. Als nächstes wird in einer Schleife so lange der Progress des Files abgefragt, bis dieses fertig verbessert wurde. Danach kann mit der „*getFileContent*“ Methode die verbesserte Datei angefragt werden. Wurde bereits eine entsprechende Grount Truth Datei hochgeladen, so wird diese nun auch ans Backend gesendet, um Statistiken zur

Verbesserung zu berechnen. Diese Statistiken werden dann in einem „Stats“ Objekt gespeichert. (mehr dazu im Kapitel zum Frontend Datenmodell 10.1.8.3)

```
function postHocr(confidence, filecontent, minLength, correctionSystem,
language, opts) {
    return new Promise((resolve, reject) => {
        ApiClientHandler.client.postHocr(confidence, filecontent, minLength,
correctionSystem, (error, data, response) => {
            if (error) {
                reject(error);
            } else {
                resolve(response);
            }
        });
    });
}

function getProgressOfCorrected(hocrFile, correctionSystem) {

    return new Promise((resolve, reject) => {
        ApiClientHandler.client.getProgressOfCorrected(hocrFile.getFileId(corr
ectionSystem).fileid, (error, data, response) => {
            if (error) {
                reject(error);
            } else {
                resolve(response);
            }
        });
    });
}

function getFileContent(file_id) {

    return new Promise((resolve, reject) => {

        ApiClientHandler.client.getCorrectedFileAsJson(file_id, (error, data,
response) => {
            resolve(response);
        });

    });
}
```

Abbildung 10.24 - API Methoden im PostProcessing

10.1.10 API-Client

Für die API-Calls wurde mittels Open API Generator (siehe Kapitel 8.4.1) ein Open API JavaScript Client generiert. Dieser wird in der Singleton Klasse „*ApiClientHandler*“ initialisiert, welche man in Abbildung 10.25 erkennen kann. Die IP-Adresse wird hierbei gegebenenfalls aus den Cookies geladen (in Abbildung 10.26 zu erkennen), falls ein solcher bereits gesetzt wurde. Ansonsten wird die Standardadresse von „*localhost:8081*“ angenommen.

```
class ApiClientHandler {

    constructor() {

        let defaultClient = new ApiClient(SettingsData._url);
        defaultClient.defaultHeaders = {}
        this._client = new DefaultApi(defaultClient);
    }
}

export default new ApiClientHandler();
```

Abbildung 10.25 - ApiClientHandler Klasse

```
class SettingsData {
    constructor() {
        this._url = "http://localhost:8081";
        this.urlChangeListeners = [];

        //get url saved in cookie
        if (Cookies.get('ipaddr') != undefined) {
            this._url = Cookies.get("ipaddr");
        }

    }
}
```

Abbildung 10.26 - SettingsData Klasse

10.1.11 Test-Skripte

10.1.11.1 Allgemein

Für die Testung des Backends sind zwei Skripte, welche aus insgesamt drei Skript-Dateien bestehen, erstellt. Diese Skripte wurden in Shell geschrieben, damit sie plattformunabhängig sind, und von jedem Gerät ausgeführt werden können, ohne etwas zusätzlich zu installieren.

- *basics.sh*
    - Testet die grundsätzlichen Funktionen des Backends.
- *batchwise.sh*
    - Lädt alle Dateien aus dem Dataset hoch. Dadurch werden die Korrektursysteme mit verschiedenen Dateien und Konfidenzen getestet. Die ausgewerteten Statistiken werden in einer Datei gesammelt, für die spätere Datenanalyse.
- *calls.sh*
    - Beinhaltet die Funktionen, welche von den anderen beiden Skripten aufgerufen werden.

10.1.11.2 Tests für Funktionalität

Das Skript *basics.sh* testet die grundsätzliche Funktionalität und Stabilität des Backends, damit programmiertechnische Fehler, welche zu einem Programmabsturz beziehungsweise zu dem Wurf einer Exception führen, gleich aufgedeckt werden können und anschließend nach dem Fehler gesucht werden kann.

Das Skript überprüft als erstes, ob das Bash-tool *curl* in dem aufrufenden Terminal installiert ist, da dieses dafür verwendet wird, um die API-Requests abzusetzen. Wenn *curl* nicht installiert ist, bricht das Skript ab, da es dann keine sinnvollen Befehle ausführen kann. (vgl. curl , 2024)

Daraufhin wird der Benutzer dazu aufgefordert, den Host, also eine IP-Adresse beziehungsweise eine Domain einzugeben, damit das Skript nicht auf eine Adresse hartcodiert ist. Das gleiche gilt auch für den Port, da zu diesem auch extra eine Eingabe gefordert wird. Für beide gibt es Standardwerte, damit einem das Testen von der Firma aus erleichtert wird. Der Standardwert für die IP-Adresse ist die, des Docker-Deployments in der Fabasoft-Cloud und der

Standard-Port ist 8081. Als Standardtest-Werte für die Konfidenz und der minimalen Wortlänge werden Konfidenz 88 und minimale Wortlänge 1 verwendet.

Im Skript selbst sind anschließend die Dateinamen für die Testfälle und jeweils Ausgaben zu den Tests enthalten. Alle API-Requests werden in Funktionen, welche in einer eigenen Skript-Datei ausgelagert sind, abgesetzt. In der *calls.sh* befinden sich drei Funktionen, welche für die Ausführung des *basics.sh* verwendet werden.

post_file_and_compare_to_correct_file:

Diese Funktion liest eine Datei von einem gegebenen Pfad ein und lädt diese mittels des Posts-Requests *uploadFile* ins Backend hoch. Anschließend wartet das Skript, bis das Verarbeiten der Datei im Backend fertig ist. Dafür werden Get-Requests verwendet, um den Status der Korrektur abzuprüfen und, wenn diese fertig ist, diese dann in das Skript zu laden. Sollte hier ein Fehler zurückkommen, wird das Skript abgebrochen, da es zum Prüfen der Stabilität gedacht ist. Wenn jedoch erfolgreich die Daten vom Backend abgerufen werden können, werden diese mit der Grundwahrheit, welche ebenfalls über einen Dateipfad übergeben wird, verglichen. Wenn sich die Anzahl der Zeichen um mehr als 5% unterscheiden, gilt der Test ebenfalls als fehlgeschlagen, weil von sehr groben Korrekturfehlern ausgegangen wird. Hierbei ist zu beachten, dass diese Funktion die Grundwahrheit nie in das Backend hochlädt, sondern lokal mit der heruntergeladenen Korrektur vergleicht.

get_stats_for_id:

Diese Funktion geht davon aus, dass die Funktion *post_file_and_compare_to_correct_file* bereits zuvor ausgeführt wurde. Dieser Funktion wird die file_id des vorherigen Aufrufs übergeben, so wie auch der Dateipfad zur Grundwahrheit, damit diese hochgeladen werden kann, und das Backend Statistiken zur Korrektur zurückgeben kann. Auf diese wird, wie in der vorherigen Funktion, wieder gewartet, es sei denn, es treten serverseitige Fehler auf. Abschließend vergleicht die Funktion, ob die Anzahl der vom Backend vorgenommenen Korrekturen, mit der übergebenen erwarteten Anzahl an Korrekturen übereinstimmt.

call_test:

Die Funktion *call_test* ruft die beiden oben beschrieben Funktionen auf, damit in der *basics.sh*-Skriptdatei nur ein Funktionsaufruf getätigt werden muss. Wenn die Funktion *post_file_and_compare_to_correct_file* erfolgreich durchläuft, werden mittels des Befehls

echo mehrmals Ausgaben an die aufrufende Funktion gemacht. Diese werden abgefangen und ausgegeben. In diesen Ausgaben befindet sich die File-id, welche mit einem *egrep* Befehl und einer passenden Regular Expression aus der Ausgabe herausgesucht wird. Mit dieser File-id wird anschließend die Funktion *get_stats_for_id* aufgerufen. Beide dieser Funktionen haben Ausgaben mittels *echo*, welche von der Funktion abgefangen und erneut ausgegeben werden, damit das aufrufende Skript diese wieder abfangen und dem Benutzer ausgeben kann. (vgl. egrep, 2024)

Genaue Informationen über die vollständige Parameterliste und die Rückgabewerte finden sich dokumentiert in den Skript-Dateien.

Nachdem das Test-Skript diese Funktionen mehrmals für verschiedene Testfälle aufgerufen und den Erfolg dieser Aufrufe evaluiert hat, wird anschließend ausgegeben, wie viele Tests durchgelaufen sind, wie viele davon erfolgreich waren und einzeln, welche davon erfolgreich sind, und welche nicht.

10.1.11.3 Tests für das Dataset

Dieses Testskript soll den gesamten Testdatensatz durchlaufen und alle Dateien ans Backend hochladen und anschließend die Statistiken davon pro Datei in eine gemeinsame Datei speichern.

Datensatz-Struktur:

- dataset
    - original
        - hOCR
            - originale Dateien
    - Ordner mit hOCR-Dateien, welche die Originaldateien in schlechterer Qualität widerspiegeln
        - hOCR
            - hOCR-Dateien
    - <weitere hOCR-Ordner mit anderer Ausgangslage>

Der Originalordner beinhaltet einen hOCR Ordner, in welchem sich wiederherum die Grundwahrheit, also das Dokument, genauso wie es korrekt ist, befindet. In allen anderen Ordnern befindet sich wiederherum ein hocr Ordner, mit Dateien, welche genau so heißen, wie die, die

sich im Originalordner befinden. Die für das Skript unrelevanten Dateien wurden in der obigen Beschreibung ausgelassen.

Das Skript fordert den Benutzer zur Eingabe eines Hosts, also einer IP-Adresse und zur Eingabe eines Ports auf. Danach läuft es alle Ordner, welche nicht der Originalordner sind, durch. Es werden pro Ordner alle Dateien mit dem Originalordner verglichen. Wenn zwei Dateien gleiche Dateinamen haben, wird die Datei aus dem Originalordner als Grundwahrheit hochgeladen, und die aus dem Ordner, der gerade durchlaufen wird, als Datei, welche verbessert werden soll, hochgeladen. Außerdem läuft das Skript über alle Korrektursysteme, welche die API anbietet, und testet alle Dateien für alle Systeme.

Es wird, wie im vorherigen Skript bereits, jede Datei hochgeladen, dann das Ergebnis der Verarbeitung abgefragt. Wenn dies erfolgreich war, dann wird das passende Original hochgeladen und auf die Berechnung der Statistiken gewartet.

Dieses Skript verwendet leicht modifizierte Varianten der Methoden, welche bei den *basics.sh*-Tests verwendet werden, damit auch die Ausgaben für die Batchvariante passend sind. Hierbei ist die Funktion, welche vom „batchwise.sh“-Skript aufgerufen wird, die Funktion *batch_call*.

Das Skript kann in zwei Modi ausgeführt werden: *single* und *multi*.

- *single*:
    - Im single-Modus werden alle Dateien sequenziell hochgeladen. Es wird immer gewartet, bis eine Datei fertig ist, bevor mit der nächsten gestartet wird. In diesem Modus werden auch die Ausgaben der Funktion an den Benutzer ausgegeben.
- *multi*:
    - Im multi-Modus wird jede Sekunde für jede Datei ein neuer Prozess gestartet, in welcher der Funktionsaufruf durchgeführt wird. Diese Prozesse können dann die Rückgaben der aufgerufenen Funktion nicht mehr ausgeben.

```
if [ "$mode" == "multi" ]; then
  # Multi Process
  batch_call $host $port $confidence $min_word_length $file $original_file
$correction_system $foldername >> ./batchwise-output/output.txt 2>&1 &
  sleep 1
else
  # Single process, one after the other
  return_value=$(batch_call $host $port $confidence $min_word_length $file
$original_file $correction_system $foldername)
  echo $return_value
fi
```

Abbildung 10.27 - Verschiedene Modi von batchwise-Skript

Diese Statistiken werden für jede Datei im gesamten Datensatz im JSON-Format in die Datei „batchwise-output/filestats.json“ geschrieben.

Da der Haupt-Prozess nicht weiß, ob bereits alle Prozesse im multi-Modus durchgelaufen sind, ist die Ausgabe jedoch nicht im korrekten JSON-Format. Deswegen ist diese auch im single-Modus nicht korrekt, da sonst zusätzliche Abprüfungen in der Datenanalyse vorgenommen werden müssten. Das Ergebnis ist deshalb eine Datei, wo jede Zeile ein korrektes JSON-Objekt darstellt, jedoch keine eckigen Klammern die gesamte Datei umfassen, und sie somit keine valide JSON-Datei ist. Dies wird in der Datenanalyse berücksichtigt.

10.1.12 Frontend-Tests

10.1.12.1 Allgemein

Für das Testen des Frontends wird ein Skript mit dem Robot-Framework und der Selenium-Library verwendet. Das Robot-Framework übernimmt hier die Ausführung der Befehle, und die Selenium-Library für das Robot-Framework ermöglicht das Aufrufen des Browsers und das Testen in diesem.

10.1.12.2 Test-Skripte

Für das Testen des Frontends wird in Python das Robot-Framework verwendet und die Selenium-Library, damit eine Webseite vom Code aus aufrufbar und testbar ist.

Für dieses Testskript wird ein Treiber für einen Browser benötigt. Das Skript ist prinzipiell auf Google Chrome ausgelegt. Es können jedoch auch andere Browser verwendet werden. Hierfür muss der jeweilige Treiber installiert sein. Die unterstützen Browser beziehungsweise Treiber

und deren Versionen sind auf der Website der Selenium-Library dokumentiert. Die Treiberdatei muss sich in einem Pfad befinden, welcher über die *PATH* -Umgebungsvariable auf Windows, so wie auf Linux unter *$PATH*, zu finden ist.

Bei Start des Skripts wird der Browser geöffnet und auf die URL des OCR-PostProcessing Projekts navigiert. Hier wird anschließend eine Konfidenz eingestellt, die Einstellungen angepasst, und eine .hOCR-Datei hochgeladen. Anschließend wird gewartet bis das Backend eine vollständige Korrektur vorgenommen hat und sich in der Mitte das korrigierte Dokument befindet. Dann wird der Download des korrigierten Dokuments getestet, eine Grundwahrheit hochgeladen und gewartet, bis Statistiken angezeigt werden können.

Zwischen den einzelnen Tests wird jeweils zwei oder drei Sekunden gewartet, damit der Tester die Änderungen im UI auch begutachten kann.

Um die Tests zu entwickeln, wird in Pycharm das IntelliBot @SeleniumLibrary patched Plugin verwendet, damit Pycharm die robot-Dateien erkennen kann, und die IDE die Keywords von Robot und Selenium auch richtig erkennt.

Um einen Test zu starten, muss im Test-Cases Ordner der *robot*-Befehl ausgeführt werden und als Parameter an diesen Befehl der Name der Testdatei übergeben werden.

Zum Beispiel: *robot TC-legacy.robot*

Es gibt zwei Test-Dateien: die *TC-legacy.robot* und *TC-new-frontend.robot*

- *TC-legacy.robot:*
    - Diese Datei ist der Test für das alte Frontend. Dieser Test wurde für das Frontend vor der Überarbeitung während der Diplomarbeit geschrieben.
- *TC-new-frontend:*
    - Diese Datei ist der Test für das neue Frontend

10.1.13 Datenanalyse

Die Datenanalyse dient zur Auswertung der Statistiken für jede Datei. Diese Datenanalyse wurde in Python implementiert und verwendet die Library pandas und deren Dataframes. In der Datenanalyse werden die Daten des *batchwise*-Skripts transformiert und ausgewertet. Implementiert wurden drei verschiedene *pivot_tables*, welche die aggregierten Daten darstellen.

10.1.13.1 Transformation

Wie bereits in Kapitel 10.1.11.3 angesprochen ist die JSON-Datei nicht valide. Deshalb muss sie vor der Datenanalyse in eine valide JSON transformiert werden, damit sie korrekt eingelesen werden kann.

Dazu muss als erstes nach jeder Zeile ein Komma gesetzt werden, damit die einzelnen JSON-Objekte später als Array eingelesen werden können. Bei *lines[-1]*, also bei der letzten Zeile, wird kein Komma eingefügt, da nach dem letzten Element in einem JSON-Array kein Komma benötigt wird.

```
# Function to insert commas between JSON objects
def add_commas(json_str: str):
    lines = json_str.split('\n')
    modified_lines = [line.strip() + ','
                      if line.strip() != '' and line != lines[-1]
                      else line.strip() for line in lines
                      if line.strip()!= '']
    return '\n'.join(modified_lines)
```

Abbildung 10.28 - Transformation Kommas

Die Datei wird eingelesen, und daraufhin wird die Funktion *add_commas* aufgerufen, um den oben bereits beschriebenen Transformationsschritt durchzuführen. Daraufhin wird dem Inhalt der Datei eine eckige Klammer vorangestellt und eine hinten angehängt, damit diese ein valides JSON-Array darstellt. Dieses kann anschließend via *json.loads* geparsed werden und anschließend wird mit *pandas*, welches als *pd* importiert ist, ein *DataFrame* erstellt.

```
import pandas as pd
from pandas import DataFrame

with open(self.__file_path, 'r') as file:
    file_content = file.read()

    # Add commas between JSON objects
    modified_content = add_commas(file_content)

    # Wrap the modified content within brackets to create a valid JSON
array
    json_array_content = f"[{modified_content}]"

    # Parse the JSON array
    data = json.loads(json_array_content)

    # Create a DataFrame from the list of JSON objects
    df = pd.DataFrame(data)
```

Abbildung 10.29 - Transformierung zu Array und Einlesen

Nachdem die Daten in ein valides Format transformiert wurden, können sie innerhalb des *DataFrames* noch weiter transformiert, normalisiert und aufbereitet werden. In jedem JSON-Objekt befindet sich die Antwort der API als *response*. Wenn diese Antwort nicht normalisiert werden würde, dann würde jede Spalte, welche sich aus der Antwort ergibt, ein Präfix „response." besitzen. Deshalb wie diese verschachtelte JSON erst normalisiert, dann die Präfixe gelöscht, und dann anschließend im ursprünglichen *DataFrame* statt der Antwort eingesetzt.

```
# Normalize the nested JSON in the 'response' column and join it with the
original DataFrame
normalized_data = pd.json_normalize(df['response'])

# Remove the "response." prefix from column names
normalized_data.columns = [col.replace('response.', '') for col in
normalized_data.columns]

# Concatenate the original DataFrame with the normalized data
result_df = pd.concat([df, normalized_data], axis=1).drop('response',
axis=1)
```

Abbildung 10.30 - Normalisieren der response

Um die DPI jedes Datensatzes leichter erkennbar zu machen, wird diese aus dem Ordnernamen entnommen und anschließend als eigene Spalte dem *DataFrame* hinzugefügt. Die Dateinamen werden auch gekürzt, um diese leichter lesbar zu machen. Am Ende werden noch drei neue Spalten hinzugefügt, damit diese nicht für jede Auswertung neu berechnet werden müssen.

```
# Create dpi column
result_df['dpi'] =
result_df['folder'].str.split('#').str[0].str[len('dpi_'):]
result_df['dpi'] = result_df['dpi'].astype(int)

# Transform filename
result_df['filename'] = result_df['filename'].apply(
    lambda x: f"{x.split('/')[2].split('#')[0]}/{x.split('/')[-1]}")

# Add the full correct and wrong columns
result_df['CORRECT'] = result_df['CORRECTLY_CORRECTED'] +
result_df['OCR_CORRECT']

result_df['WRONG'] = result_df['WRONGLY_CORRECTED'] +
result_df['OCR_FALSE']

result_df['OVERALL'] = result_df['CORRECT'] + result_df['WRONG']
```

Abbildung 10.31 - Bearbeitung verschiedener Spalten

10.1.13.2 Analyse

Durch die Analyse der Daten lässt sich feststellen, welches System beim Durchlauf des Testdatensatzes am besten abgeschnitten hat. Es wird auch analysiert, welches System bei welchem DPI am besten geeignet ist. Für jede einzelne Datei wird auch berechnet, welches System das Beste für diese Datei war. Das „beste System“ ist in diesem Fall relativ, da ein System insgesamt mehr richtig verbessert haben kann, aber prozentual zu seinen falschen Korrekturen schlechter abgeschnitten hat als ein anderes System. Deshalb wird der prozentuelle Wert für die richtig und falsch korrigierten Wörter ebenfalls angegeben.

In dieser Methode wird eine Ausgabe zu den Statistiken für jedes System vorbereitet. Als erstes wird eine Kopie des *DataFrames* gemacht. Danach werden die Spalten, welche verwendet werden sollen, bestimmt, danach der Pivot-Table erstellt. Da in diesem die Daten bereits nach Korrektursystem aggregiert wurden, welche nun erst die prozentualen Werte bestimmt, da diese sonst nicht pro System wären. Anschließend werden diese dem Pivot-Table angehängt und dieser ausgegeben.

```
def plot_correct_and_wrong_by_system(self):
    df = self.__df.copy()

    columns = ['CORRECTLY_CORRECTED', 'OCR_CORRECT', 'WRONGLY_CORRECTED',
'OCR_FALSE', 'CORRECT', 'WRONG', 'OVERALL']

    pivot_table = df.pivot_table(index=['correction_system']
                                 values=columns, aggfunc='sum')

    pivot_table['CORRECT_%'] = (pivot_table['CORRECT'] /
                                 pivot_table['OVERALL']) * 100

    pivot_table['WRONG_%'] = (pivot_table['WRONG'] /
                                 pivot_table['OVERALL']) * 100

    pivot_table = pivot_table[columns+['CORRECT_%', 'WRONG_%']]

    print(pivot_table)
```

Abbildung 10.32 - Darstellung der Statistiken für jedes System

Diese Methode ist der oben beschriebenen sehr ähnlich. Im Wesentlichen wird nur ‘dpi‘ als weiterer Indizes hinzugefügt, was nun nicht nur nach System, sondern danach nach DPI gruppiert und aggregiert.

```
def plot_stats_by_system_and_dpi(self):
    df = self.__df.copy()

    columns = ['CORRECTLY_CORRECTED', 'OCR_CORRECT', 'WRONGLY_CORRECTED',
'OCR_FALSE', 'CORRECT', 'WRONG', 'OVERALL']
    pivot_table = df.pivot_table(index=['correction_system', 'dpi'],
                                 values=columns, aggfunc='sum')
    # same Code…
```

Abbildung 10.33 - Darstellung der Statistiken für jedes System und DPI

Diese Methode ist der oben beschrieben ebenfalls sehr ähnlich, nur wird hier in erster Linie nach dem Dateinamen und dann nach dem Korrektursystem gruppiert und aggregiert.

```
def plot_stats_by_file_and_system(self):
    df = self.__df.copy()

    columns = ['CORRECTLY_CORRECTED', 'OCR_CORRECT', 'WRONGLY_CORRECTED',
'OCR_FALSE', 'CORRECT', 'WRONG', 'OVERALL']
    pivot_table = df.pivot_table(index=['filename', 'correction_system'],
                                 values=columns, aggfunc='sum')
    # same Code…
```

Abbildung 10.34 – Darstellung der Statistiken für jedes System und DPI

Diese Methoden analysieren das beste System, das beste System pro DPI und das beste System pro Datei. Diese Daten können verwendet werden, um bei einem neuen Korrektursystem zu bestimmen, ob dieses besser ist als die bisher implementierten Systeme.

11. Ergebnis

Das Ergebnis der Diplomarbeit besteht aus sechs Teilen: der Backend Basisklasse, Unit-Tests zum Überprüfen der Funktionalität, den neuen Korrektursystemen, einem Frontend zur Visualisierung der Verbesserung, den Tests und der Datenanalyse.

11.1.1 Backend Basisklasse

Wie in Kapitel 10.1.3 Backend Basisklasse beschrieben, handelt es sich bei der Basisklasse um eine teilweise Implementierung eines Korrektursystems. Diese dient zu einer einfachen Implementierung von neuen Korrektursystemen, da dafür nur die bereits definierten Methoden dieser Klasse überschrieben werden müssen. Des Weiteren wird durch diese Klasse eine Auflistung aller verfügbaren Korrektursysteme zur Verfügung gestellt.

11.1.2 Unit-Tests

Um die dauerhafte Funktionalität des Systems gewährleisten zu können, wurden Unit-Tests implementiert. Diese stellen sicher, dass jede Implementierung der Basisklasse (neues Korrektursystem) alle verfügbaren Methoden richtig überschreibt und gültige Werte bei der Verbesserung zurückgegeben werden. Weitere Informationen zu den Unit-Tests sind im Kapitel 10.1.3 Backend Unit-tests zu finden.

11.1.3 Neue Korrektursysteme

11.1.3.1 bert-base-german-cased

Das bert-base-german-cased Modell basiert auf Künstlicher Intelligenz. Dieses Modell sagt das Wort voraus, welches mit dem [MASK]-Token ersetzt wurde. Die Implementierung ist im Kapitel 10.1.6 zu finden. Es folgt ein Beispiel zur Funktion dieses Modells.

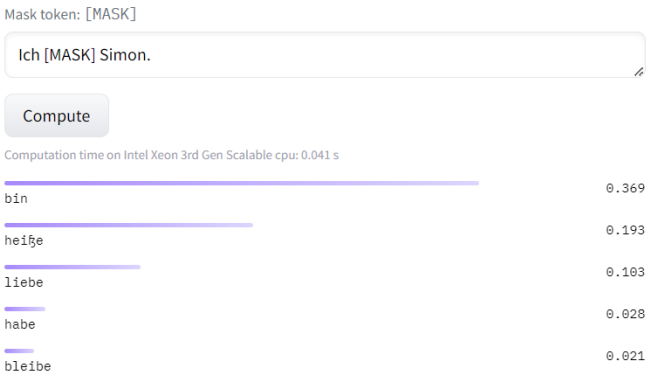


Abbildung 11.1 - bert-base-german-cased

Das Wort, das vorhergesagt werden soll, muss durch einen [MASK]-Token ersetzt werden. Das Modell nimmt den restlichen Satz als Input und kann somit aufgrund des restlichen Satzes, den [MASK]-Token vorhersagen. Als Ergebnis werden immer 5 Wörter zurückgegeben, die absteigend nach der Wahrscheinlichkeit sortiert sind. Also ist das wahrscheinlichste Wort immer das erste.

11.1.3.2 Phunspell

Phunspell ist ein Korrektursystem, welches mithilfe eines Wörterbuches und einer zusätzlichen Affix Datei funktioniert. Dieses Wörterbuch und die Affix Datei bringt es für die deutsche Sprache bereits mit, daher haben wir diese verwendet. Die Implementierung des Korrektursystems mithilfe dieser Bibliothek ist im Kapitel 10.1.5 beschrieben.

Hier folgt ein Beispiel für einen Wörterbucheintrag. Der Wörterbucheintrag für "Ärger" lautet: "Ärger/Sm". Das bedeutet, dass das Suffix "Sm" dem Wort hinzugefügt wird. Das "S" fügt ein "s" hinzu, um es in die Mehrzahl zu setzen, und das "m" ändert die Groß- und Kleinschreibung von Wörtern nach einem Bindestrich. Also wenn das Wort zuvor klein geschrieben war, wird es großgeschrieben, und wenn es zuvor großgeschrieben war, wird es klein geschrieben. Somit

wird vermieden, dass es für jedes Wort unzählige Einträge gibt, es werden einfach die benötigten Affixe angehängt, um die verschiedenen Formen des Wortes zu erreichen. Die Affixe sind im Affix File definiert. (vgl. github/dvwright/phunspell, 2024)

Besonders hervorzuheben ist, dass Phunspell überprüft, ob das Wort, welches verbessert werden soll, zusammengeschrieben ist, wenn es nicht im Wörterbuch ist. Das bedeutet, es kann zum Beispiel das Wort „sollnur“ zu „soll“ und „nur“ verbessern. Somit ist Phunspell das einzige von uns implementierte Korrektursystem, welches Wörter trennen kann.

11.1.4 Frontend

Das Frontend wurde im Rahmen der Diplomarbeit um einige Funktionen erweitert. Die Ausgangslage ist im Kapitel 7.4.2 genauer erläutert. In Abbildung 11.2 ist das erweiterte Architekturdiagramm zu erkennen. Der markanteste Unterschied liegt in der Möglichkeit, dass mehrere Dateien vom Benutzer hochgeladen werden können oder gegebenenfalls auch ein gesamter Datensatz. Es werden nun ebenfalls mehrere unterschiedliche Korrektursysteme vom Backend zu Verfügung gestellt, welche alle in der Frontend Applikation angezeigt und ausgewählt werden können.

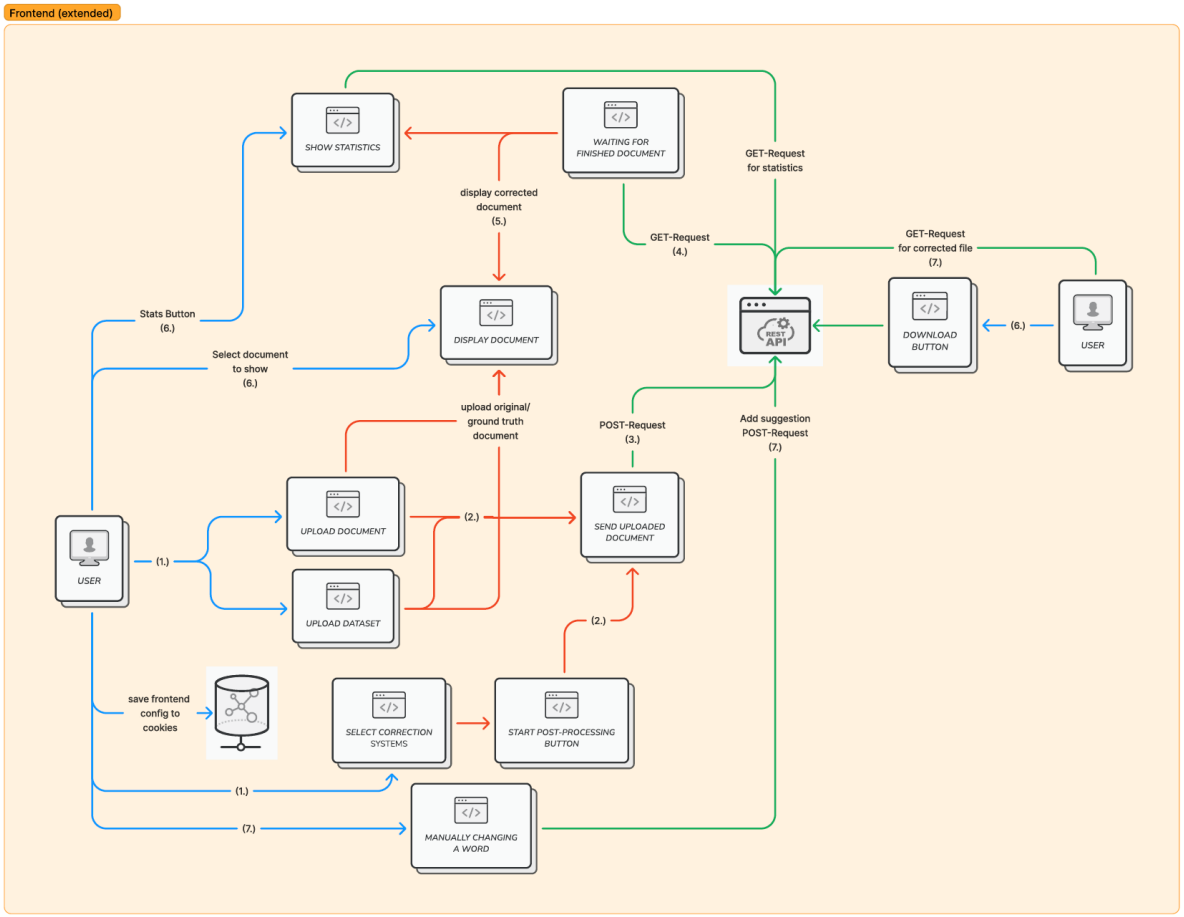


Abbildung 11.2 - Frontend Architektur Extended

11.1.4.1 Fertige Frontend Applikation

In Abbildung 11.3 kann die fertige Frontend-Applikation erkannt werden. In der Abbildung wurde noch kein Dokument hochgeladen, wie in der Sidebar am rechten Bildschirmrand zu erkennen ist.

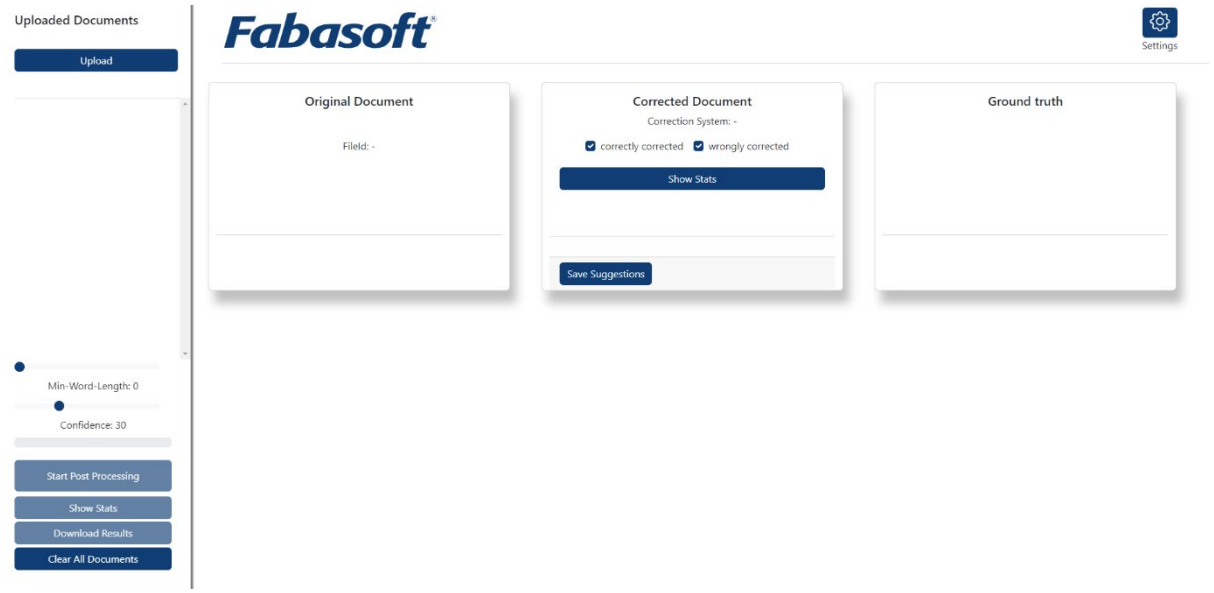


Abbildung 11.3 - Frontend Empty

11.1.4.2 Upload Dialog

Wird der Upload-Button in der Sidebar vom Benutzer geklickt, öffnet sich der Upload Dialog, welcher in Abbildung 11.4 zu erkennen ist. In diesem kann der Benutzer entscheiden, ob er ein einzelnes Dokument, die „korrekte“ Version eines Dokuments oder einen gesamter Ordner hochladen will.

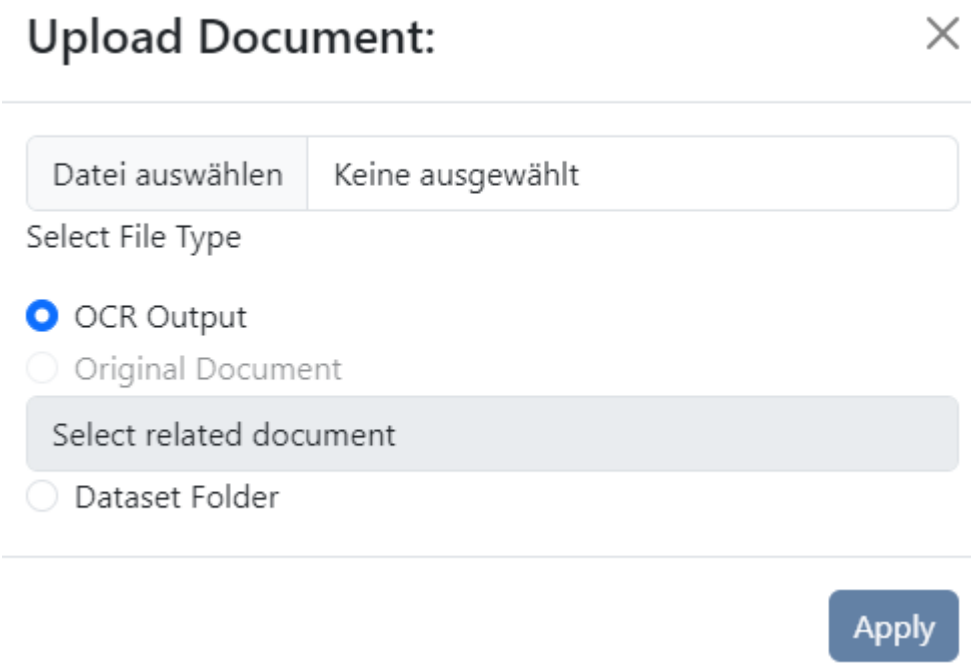


Abbildung 11.4 - Upload Dialog

11.1.4.3 Sidebar

Wurden die Dokumente nun hochgeladen, werden diese in der Sidebar angezeigt (in Abbildung 11.5 zu erkennen). Hier werden nun als oberste Ebene die Namen der hochgeladenen Dokumente angezeigt. Direkt darunter ist die hochgeladene Ground Truth zu erkennen. Unter dieser finden sich die jeweiligen Qualitätsstufen, welche wiederum alle Korrektursysteme untergeordnet haben. Klickt der Benutzer nun auf ein Korrektursystem wird das Dokument in entsprechender DPI-Stufe und die dazugehörige Ground Truth angezeigt. Dies kann in Abbildung 11.6 erkannt werden. In der Sidebar kann ebenfalls die Mindestwortlänge und Mindestkonfidenz für Wörter ausgewählt werden (genauer beschreiben in Kapitel 9.2.4 und 9.2.5).

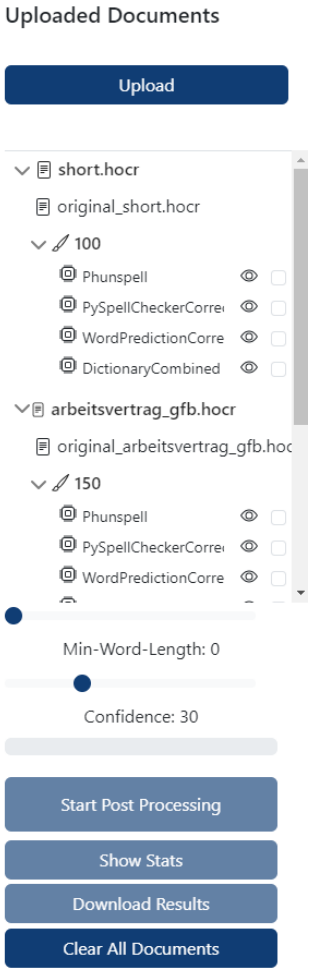


Abbildung 11.5 - Sidebar

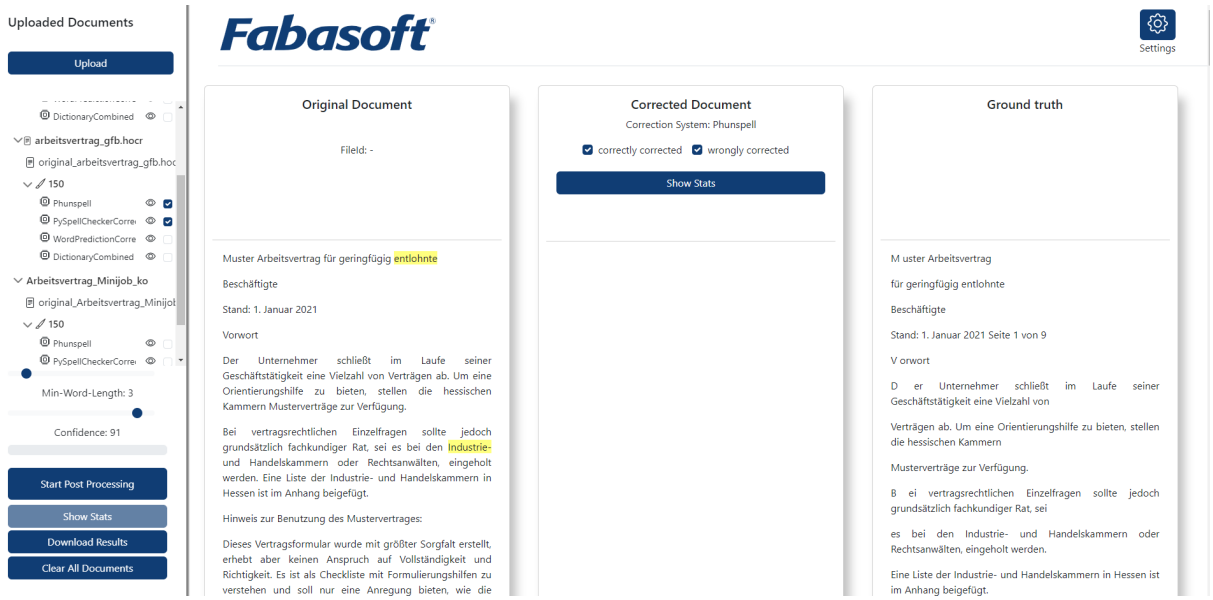


Abbildung 11.6 – Frontend Selected File

11.1.4.4 Verbesserungen Anzeigen

Wurden alle zur Korrektur gewünschten Files mittels Checkbox ausgewählt, kann nun mit drücken des Start Post Processing Buttons der Verbesserungsvorgang gestartet werden (genauer beschrieben im Kapitel zur Implementierung des Post Processing Vorgangs 10.1.9). Die richtig und falsch verbesserten Wörter werden im Anschluss farblich gekennzeichnet, falls eine Ground Truth hochgeladen wurde. In Abbildung 11.7 kann man dies erkennen, wobei in der linken Spalte das originale Dokument und in der rechten Spalte das korrigierte Dokument dargestellt ist. Die grün hinterlegten Wörter „monatlich“ und „entgeltlichen“ wurden vom Korrektursystem laut Ground Truth richtig verbessert. „Aderung“ wurde jedoch falsch verbessert und wird somit rot hinterlegt.

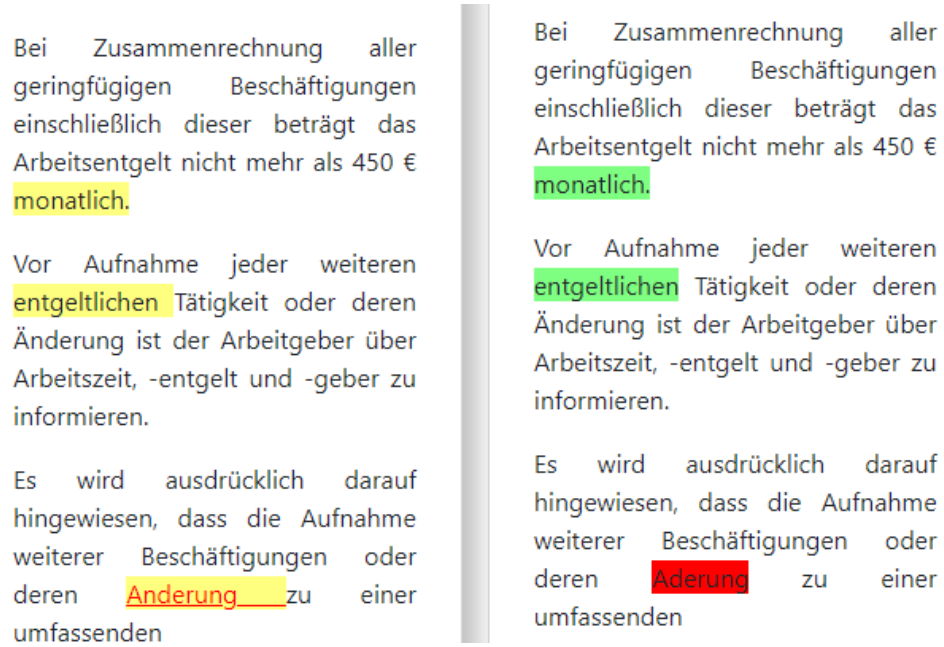


Abbildung 11.7 - Highlighting

Klickt der Benutzer mit einem Rechtsklick auf ein verbessertes Wort, so kann dieser sich auch alternative Vorschläge des Korrektursystems anzeigen lassen, wie in Abbildung 11.8 zu erkennen ist.

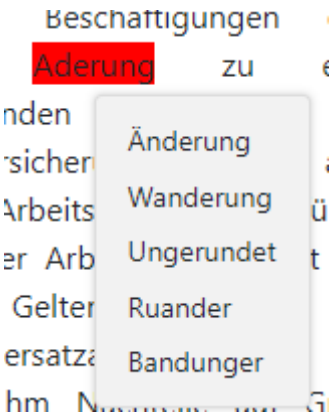


Abbildung 11.8 - Suggestions

11.1.5 Test-Skripte

Um die Funktionalität von sowohl Backend als auch Frontend zu testen, wurden Testskripte geschrieben. Ein Testskript für die Funktionalität des Backends und der API wurde geschrieben, wie in Kapitel 10.1.11.2 beschrieben. Um das Backend weiter zu testen und Daten für die Datenanalyse zu liefern, wurde ein Skript, welches den gesamten Testdatensatz durchläuft, geschrieben, wie in Kapitel 10.1.11.3 beschrieben. Um das Frontend auf die Funktionalität zu testen, wurden Tests für dieses geschrieben, welche in Kapitel 10.1.12 geschildert werden.

11.1.6 Datenanalyse

Um die Qualität der Korrektursysteme zu ermitteln, wurde eine Datenanalyse implementiert, welche die Statistiken aus dem Testdatensatz aggregiert und auswertet. Durch die Datenanalyse kann bestimmt werden, ob ein neues System besser oder schlechter ist als die bereits vorhandenen Systeme. Auf die genaue Implementierung der Datenanalyse wird in Kapitel 10.1.13 detaillierter eingegangen.

12. Resümee

Durch den Projektentwicklungsunterricht konnte die Planung sinnvoll und selbstständig durchgeführt werden. Das Erstellen eines Zeitplans, zuteilen von Aufgaben oder die Definition von Zielen im Projekt stellten so keine Hindernisse dar. Während des Praktikums wurden wöchentlich Meetings gehalten. In diesen Meetings wurde jeweils der Projektstatus präsentiert und diskutiert.

Im Rahmen der Implementierung wurden zahlreiche neue Technologien kennengelernt. Dadurch konnte viel neues Wissen erlangt werden. Hauptsächlich wurde das Wissen in folgenden Themenbereichen vertieft:

- Python
- JavaScript
- Wörterbuchbasierte und KI-basierte Korrektursysteme
- Shell Skripte
- Robot Framework
- React Framework

Abschließend kann gesagt werden, dass die Diplomarbeit erfolgreich abgeschlossen worden ist, und durch das Praktikum konnten viele Erfahrungen in der Berufswelt gesammelt werden.

13. Abbildungsverzeichnis

14. Tabellenverzeichnis

15. Quellenverzeichnis

abbyy. (14. 02 2024). Von abbyy: https://pdf.abbyy.com/de/learning-center/what-is-ocr/#:~:text=Wofür%20steht%20eigentlich%20OCR%3F,bearbeitbare%20und%20durchsuchbare%20Dateien%20ermöglicht. abgerufen

amazon. (2. 14 2024). Von amazon: https://aws.amazon.com/de/what-is/javascript/#:~:text=JavaScript%20ist%20eine%20Programmiersprache%2C%20die,die%20Benutzerfreundlichkeit%20einer%20Website%20verbessern. abgerufen

atlassian. (14. 2 2024). Von atlassian: https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow abgerufen

aws Was ist der Unterschied zwischen Docker-Images und Containern? (17. 2 2024). Von aws: https://www.google.com/search?q=Was+ist+ein+Docker+Container&rlz=1C1ONGR_deAT1066AT1066&oq=Was+ist+ein+Docker+Container&gs_lcrp=EgZjaHJvbWUyBggAEEUYOTIGCAEQIxgnMgYIAhAjGCcyBwgDEAAYgAQyBwgEEAAYgAQyBwgFEAAYgAQyBwgGEAAYgAQyBwgHEAAYgAQyBwgIEAAYgAQyBwgJEAAYgAT abgerufen

biteno. (11. 1 2024). Von biteno: Was ist PyCharm? Eine Einführung.: https://www.biteno.com/was-ist-pycharm/ abgerufen

codeinstitute. (2. 14 2024). Von codeinstitute: https://codeinstitute.net/global/blog/what-is-vs-code/ abgerufen

computerweekly. (14. 02 2024). Von computerweekly: https://www.computerweekly.com/de/definition/Bash-Bourne-again-Shell#:~:text=Bash%20(Bourne%20Again%20Shell)%20ist,wie%20die%20Bearbeitung%20von%20Befehlszeilen. abgerufen

crummy. (19. 2 2024). Von crummy: BeautifulSoup: https://www.crummy.com/software/BeautifulSoup/bs4/doc/ abgerufen

curl . (20. 03 2024). Von curl: https://curl.se/ abgerufen

dublincore. (14. 02 2024). Von dublincore: https://www.dublincore.org/ abgerufen

egrep. (20. 03 2024). Von rgrep: https://www.computerhope.com/unix/uegrep.htm abgerufen

fabasoft.com. (21. 2 2024). Von Fabasoft: https://www.fabasoft.com/de abgerufen

flask logo. (20. 03 2024). Von flask logo: https://flask.palletsprojects.com/en/3.0.x/tutorial/factory/ abgerufen

geeksforgeeks wrapper. (20. 03 2024). Von geeksforgeeks wrapper: https://www.geeksforgeeks.org/function-wrappers-in-python/ abgerufen

github .hOCR-Spezifikation. (14. 02 2024). Von github: https://kba.github.io/hocr-spec/1.2/ abgerufen

github Beispiele OCR-Engines. (14. 02 2024). Von github: https://github.com/kba/awesome-ocr?tab=readme-ov-file#ocr-engines abgerufen

github Tesseract Data. (14. 02 2024). Von github: https://tesseract-ocr.github.io/tessdoc/Data-Files-in-different-versions.html abgerufen

github Tesseract Ergebnis. (14. 02 2024). Von github: https://github.com/tesseract-ocr/tesseract?tab=readme-ov-file abgerufen

github/dvwright/phunspell. (03. 08 2024). Von github/dvwright/phunspell: https://github.com/dvwright/phunspell abgerufen

hubspot. (14. 2 2024). Von hubspot: https://blog.hubspot.com/website/react-js abgerufen

Hugging Face Logo. (2024). Von Hugging Face : https://huggingface.co/brand abgerufen

huggingface. (12. 12 2024). Von huggingface: https://huggingface.co/ abgerufen

hwlang. (13. März 2023). Abgerufen am 13. Februar 2024 von hwlang: Needleman-Wunsch-Algorithmus: https://hwlang.de/algorithmen/pattern/needleman-wunsch.htm

intellibot. (20. 03 2024). Von intellibot: https://plugins.jetbrains.com/plugin/10700-intellibot-seleniumlibrary-patched abgerufen

ionos Flask Framework. (14. 02 2024). Von ionos: https://www.ionos.at/digitalguide/websites/web-entwicklung/flask-framework-im-ueberblick/ abgerufen

ionos HTTP. (14. 02 2024). Von ionos: https://www.ionos.at/digitalguide/server/knowhow/rest-die-http-loesung-fuer-webservices/ abgerufen

jetbrains. (13. 2 2024). Von jetbrains: https://resources.jetbrains.com/storage/products/company/brand/logos/PyCharm.png abgerufen

mobaxterm. (2. 15 2024). Von mobaxterm: https://mobaxterm.mobatek.net/ abgerufen

opc-router rest symbol. (20. 03 2024). Von opc-router rest symbol: https://www.opc-router.com/what-is-rest/ abgerufen

openapi-generator. (13. 02 2024). Von openapi-generator: https://github.com/jimschubert/intellij-openapi-generator abgerufen

pandas. (20. 03 2024). Von pandas: https://pandas.pydata.org/ abgerufen

pixx. (10. 2 2024). Von pixx: https://www.pixx.io/blog/was-bedeutet-dpi#:~:text=Die%20Feinheit%20des%20Rasters%20wird,Raster%2DReihen%22%20zu%20drucken. abgerufen

postman. (13. 02 2024). Von postman: https://www.postman.com/ abgerufen

pypi. (20. 3 2024). Von pypi pyspellchecker: https://pypi.org/project/pyspellchecker/ abgerufen

python. (13. 2 2024). Von python: https://www.python.org/community/logos/ abgerufen

robotframework. (20. 03 2024). Von robotframework: https://robotframework.org/ abgerufen

robothelper mozilla addon. (20. 03 2024). Von robothelper mozilla addon: https://addons.mozilla.org/en-US/firefox/addon/robothelper/?utm_source=addons.mozilla.org&utm_medium=referral&utm_content=search abgerufen

ryte wiki. (13. 4 2015). Von ryte wiki: Python: https://de.ryte.com/wiki/Python abgerufen

selenium. (20. 03 2024). Von selenium: https://www.selenium.dev/ abgerufen

swissq selenium logo. (20. 03 2024). Von swissq selenium logo: https://swissq.it/wp-content/uploads/2023/04/Selenium_Logo-1.png abgerufen

testautomatisierung postman logo. (20. 03 2024). Von testautomatisierung postman logo: https://www.testautomatisierung.org/lexikon/postman-api/postman/ abgerufen

w3schools. (14. 02 2024). Von w3schools: https://www.w3schools.com/html/html_xhtml.asp abgerufen

w3schools. (14. 2 2024). Von w3schools: https://www.w3schools.com/whatis/whatis_npm.asp abgerufen

wikimedia bash logo. (20. 03 2024). Von wikimedia bash logo: https://commons.wikimedia.org/wiki/File:Bash_Logo_Colored.svg abgerufen

wikipedia .hOCR-Standard. (14. 02 2024). Von wikipedia: https://de.wikipedia.org/wiki/HOCR_(Standard) abgerufen

wikipedia Levenshtein_distance. (08. 03 2024). Von wikipedia Levenshtein_distance: https://en.wikipedia.org/wiki/Levenshtein_distance abgerufen

wikipedia Pip_(Python). (17. 2 2024). Von wikipedia: https://de.wikipedia.org/wiki/Pip_(Python) abgerufen

wikipedia robot framework. (20. 03 2024). Von wikipedia robot framework: https://de.wikipedia.org/wiki/Robot_Framework#/media/Datei:Robot-framework-logo.png abgerufen

wikipedia Tesseract. (14. 02 2024). Von wikipedia: https://de.wikipedia.org/wiki/Tesseract_(Software) abgerufen

wikipedia Texterkennung. (14. 02 2024). Von wikipedia: https://de.wikipedia.org/wiki/Texterkennung abgerufen