

Abschlussprüfung Winter 2024
Fachinformatiker für Anwendungsentwicklung

Dokumentation zur betrieblichen Projektarbeit

Email Verwalten

Automatisierte E-Mail-Anhangs Verwaltung

Abgabedatum: Hamburg, den 11.11.2024

Prüfungsbewerber:

Ian Manuel Paniagua Porroa
Beim Andreasbrunnen 6
20249 Hamburg

Ausbildungsbetrieb:

BBQ Bildungs Bauman Qualifizierung
Wendenstraße 25
20097 Hamburg

Praktikumsbetrieb:

Maxedv Beratung GmbH
Boytinstraße 25
22143 Hamburg



Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis	I
Abbildungsverzeichnis	III
Tabellenverzeichnis	IV
Abkürzungsverzeichnis	V
1 Einleitung	6
1.1 Projektumfeld	6
1.2 Projektziel	6
1.3 Projektbegründung	6
1.4 Projektschnittstellen	6
IMAP-Verbindung zu E-Mail-Servern	6
Erkennung neuer E-Mails	6
Dokumentenverarbeitung	6
Dateiverwaltung	6
Terminal	6
Verbindungsstatus	6
1.5 Projektbegrenzung	7
2 Projektplanung	7
2.1 Projektphasen	7
2.2 Abweichungen vom Projektantrag	8
2.3 Ressourcenplanung	8
2.4 Entwicklungsprozess	8
3 Analysephase	9
3.1 Ist-Analyse	9
3.2 Wirtschaftlichkeitsanalyse	9
3.2.1 Make or Buy-Entscheidung	10
3.2.2 Projektkosten	10
3.3 Nutzwertanalyse	11
3.3.1 Nicht-monetärer Nutzen:	11
3.3.2 Wirtschaftlichkeitskoeffizient:	11
3.4 Anwendungsfälle	12
3.4.1 Anwendungsfälle des Projekts:	12
3.4.2 Detaillierter Ablauf des Prozesses:	12
3.4.3 Visualisierung durch ein Aktivitätsdiagramm:	13
3.5 Qualitätsanforderungen	15
3.5.1 Qualitätsanforderungen	15
3.6 Lastenheft/Fachkonzept	16
3.6.1 Projektbeschreibung	16
3.6.2 Anforderungen	16
3.6.3 Detaillierte Anforderungen	16

EMAIL VERWALTEN

Automatisierte E-Mail-Anhangs Verwaltung

Inhaltsverzeichnis

3.6.4	Technische Anforderungen	16
3.6.5	Implementierung und Technologien	17
4	Entwurfsphase	17
4.1	Zielplattform	17
4.1.1	Kriterien zur Auswahl der Zielplattform:	17
4.2	Architekturdesign	18
4.2.1	Beschreibung der gewählten Architektur:	18
4.3	Entwurf der Benutzeroberfläche	18
4.4	Datenmodell	19
4.5	Geschäftslogik	Fehler! Textmarke nicht definiert.
4.6	Maßnahmen zur Qualitätssicherung	20
	Manuelle Tests:	20
	Echte Testumgebung:	20
	Automatisierte Tests:	20
4.7	Pflichtenheft/Datenverarbeitungskonzept	20
4.7.1	Projektbeschreibung	20
4.7.2	Anforderungen	21
4.7.3	Detaillierte Anforderungen	21
4.7.4	Technische Anforderungen	21
4.7.5	Implementierung und Technologien	21
5	Implementierungsphase	22
5.1	Implementierung der Datenstrukturen	22
5.2	Implementierung der Benutzeroberfläche	27
5.3	Implementierung der Geschäftslogik	31
6	Abnahmephase	39
7	Einführungsphase	40
8	Dokumentation	42
9	Soll-/Ist-Vergleich	42
9.1	Lessons Learned	43
9.2	Ausblick	43
	Literaturverzeichnis	45
	Eidesstattliche Erklärung	46

Abbildungsverzeichnis

Abbildung 1: Berechnung geschätzte Durchführungszeit	10
Abbildung 2: Vorher-Nachher-Vergleich.....	11
Abbildung 3: Theoretischen Bewertung	11
Abbildung 4: Aktivitätsdiagramm	14
Abbildung 5: DB_objects.json Beispiel	19
Abbildung 6: GUI Benutze Oberfläche	29
Abbildung 7: GUI Auswahl des Ordners.....	29
Abbildung 8: Konfirmation Fenster	30
Abbildung 9: Ablauf PDFs	38

Tabellenverzeichnis

Tabelle 1: Projektphasen	7
Tabelle 2: Berechnung geschätzte Durchführungszeit	10
Tabelle 3: Vorher-Nachher-Vergleich	11

Abkürzungsverzeichnis

API *Aplication Programing Interface, Aplication Programming Interface*

GUI *Graphical User Interface, Graphical User Interface*

IMAP *Internet Message Acces Protocol*

OCR *Optical Character Recoognition*

SSL/TLS *Secure Sockets Layer / Transport Layer Security*

UID *Unique Identifier*

1 Einleitung

1.1 Projektumfeld

Der Kunde des Projekts ist ein Unternehmen, das den Prozess der Erfassung und Organisation von Rechnungen, die per E-Mail eingehen, automatisieren möchte. Der aktuelle manuelle Prozess führt zu Ineffizienz und Fehlern. Die Firma erhält Rechnungen für verschiedene Eigentümer und benötigt eine automatische Organisation dieser Anhänge. Es handelt sich dabei um ein großes Volumen an Rechnungen.

1.2 Projektziel

Das Hauptziel ist es, die Rechnungen, die per E-Mail eingehen, automatisch herunterzuladen und lokal in Ordnern nach Eigentümer und Verwaltungsnummer zu organisieren. Die E-Mails enthalten Rechnungen für verschiedene Eigentümer derselben Firma. Daher ist es wichtig, die Anhänge nach Eigentümer zu ordnen und klar zu kennzeichnen, ob es sich um eine Rechnung oder einen Lieferschein handelt.

1.3 Projektbegründung

Das Projekt ist sinnvoll, da die Anzahl der E-Mails mit Rechnungen sehr hoch ist. Die manuelle Bearbeitung dieser E-Mails kostet viel Zeit. Mit der Implementierung der Anwendung können Kosten gesenkt, die Arbeitsbelastung reduziert, menschliche Fehler minimiert und ein besserer Überblick über die Rechnungen sichergestellt werden.

1.4 Projektschnittstellen

IMAP-Verbindung zu E-Mail-Servern

Die Anwendung verbindet sich mit E-Mail-Konten, um Anhänge automatisch herunterzuladen. Unterstützt wird der Server von IONOS. Die Anmeldung erfolgt über das Terminal mit E-Mail-Adresse und Passwort. Verarbeitete E-Mails werden als ungelesen markiert, damit eine manuelle Überprüfung möglich bleibt.

Erkennung neuer E-Mails

Eine Funktion überprüft regelmäßig, ob neue E-Mails in der Inbox vorhanden sind. Sie speichert den UID der zuletzt verarbeiteten E-Mail und sucht nach neuen, sowohl gelesenen als auch ungelesenen E-Mails. Anhänge werden in einem festgelegten Ordner gespeichert. Zusätzlich werden Basisdaten wie Datum, Betreff und UID der E-Mails gespeichert. Die E-Mails bleiben in ihrem ursprünglichen Zustand.

Dokumentenverarbeitung

Der Text aus den Anhängen wird extrahiert und mit der JSON-Datenbank abgeglichen, um Übereinstimmungen zu finden.

Dateiverwaltung

Dateien werden basierend auf ihrem Status (verarbeitet, ausstehend, doppelt) in lokale Ordner organisiert.

Terminal

Das Terminal zeigt die heruntergeladenen und kategorisierten Dateien übersichtlich an.

Verbindungsstatus

Das Projekt wurde vom Kunden angefragt und wird dem Leiter von Maxedv, den betroffenen Mitarbeitern und anschließend dem Kunden präsentiert. Die Hauptnutzer sind die Verwaltungsmitarbeiter der Firma.

1.5 Projektbegrenzung

- Das Projekt umfasst nicht die Ausstellung oder Bezahlung von Rechnungen.
- Informationen zu den Beträgen der Rechnungen werden nicht extrahiert oder analysiert.
- Es werden alle Anhänge heruntergeladen, aber derzeit werden nur PDF-Dateien verarbeitet.
- E-Mails ohne Anhänge werden nicht berücksichtigt.
- Die Analyse der Anhänge unterstützt derzeit nur PDF-Dokumente in deutscher und englischer Sprache.
- Die Anwendung ist speziell für die Nutzung unter Windows konzipiert.

2 Projektplanung

2.1 Projektphasen

- Das Projekt wird in maximal 80 Stunden entwickelt.
- Die Dokumentation wird spätestens am 11.12.2024 an die IHK übergeben.
- Das Projekt wird nach den Anforderungen der IHK durchgeführt.
- Die Anwendung muss vor dem 09.12.2024 auf dem Server des Kunden getestet werden.

Tabelle 1: Projektphasen

Projektphasen	Beschreibung	Zeit
Planung und Analyse	Definition der Anforderungen, Auswahl der Technologien wie Python und IMAP, Klärung des Funktionsumfangs	8
Entwicklung - Setup und Basisconfiguration	Dynamische IMAP-Verbindung, Auswahl des Speicherordners, Erstellung von DB_Objects.json	10
Entwicklung der IMAP-Verbindung und E-Mail-Handler	Verbindung mit dem IMAP-Server, Abrufen und Filtern von E-Mails mit Anhängen, Verfolgung der UIDs	12
Verarbeitung der Anhänge	Bearbeitung der Anhänge: Umbenennen, Textextraktion, Verschieben und Abgleich mit DB_Objects.json	10
Refaktorisierung des Codes	Bearbeitung der Anhänge: Umbenennen, Textextraktion, Verschieben und Abgleich mit DB_Objects.json	8
Tests und Fehlerbehebung	Überprüfung des Workflows, Fehlerbehebung und Sicherstellung der Funktionalität unter Windows	8
PyInstaller und reale Programmnutzung	Überprüfung des Workflows, Fehlerbehebung und Sicherstellung der Funktionalität unter Windows	10
Dokumentation und	Erstellung der Projektdokumentation und Vorbereitung der Präsentationsmaterialien	8

EMAIL VERWALTEN

Automatisierte E-Mail-Anhangs Verwaltung

2.2 Abweichungen vom Projektantrag

Es gab Änderungen bei den Prioritäten und Anforderungen des Projekts:

Änderungen der Hauptanforderungen

Ursprünglich war geplant, die Rechnungsnummer zu erkennen und Anhänge zusammenzuführen. Diese Funktionen sind jedoch nicht mehr die oberste Priorität. Stattdessen wurde die Kategorisierung der Rechnungen nach Eigentümern in den Fokus gerückt.

Der Kunde hat uns ein PDF-Dokument mit den Daten der Eigentümer zur Verfügung gestellt, sodass wir feststellen können, ob eine Rechnung einem bestimmten Eigentümer gehört. Dafür war es notwendig, zusätzliche Zeit für die Erstellung einer JSON-Datei mit den Daten der Eigentümer und deren Nutzung einzuplanen, was ursprünglich nicht vorgesehen war.

Änderungen bei der Nutzung der Anwendung

Es wurde klargestellt, dass der Kunde keine Konfiguration oder Bedienung des Programms durchführen muss. Wir haben direkten Zugriff auf den PC des Kunden sowie auf die E-Mail-Zugangsdaten. Daher wurde die grafische Benutzeroberfläche (GUI) durch eine reine Konsolenlösung ersetzt. Diese Anpassung optimiert die Bedienung, da wir die Anwendung direkt konfigurieren und auf dem PC des Kunden installieren.

Diese Änderungen wurden berücksichtigt, um die Anforderungen des Kunden bestmöglich zu erfüllen. Sie führten jedoch zu einer Umverteilung der geplanten Ressourcen und Prioritäten.

2.3 Ressourcenplanung

- **Software-Ressourcen:**
 - **Programmiersprache:** Python and Dokumentation.
 - **Entwicklungsumgebung:** Visual Studio Code (VSCode).
 - **Versionskontrolle:** Git, GitHub
 - **Projektmanagement** und Organisation:
 - Notion zur Strukturierung und Dokumentation des Projektfortschritts
 - Die Perfekte Projektdokumentation: Die Vorlage von dieperfekteprojekttdokumentation.de wurde als Grundlage verwendet, jedoch individuell an die spezifischen Anforderungen dieses Projekts angepasst.
 - **Unterstützende Tools:** ChatGPT wurde verwendet, um technische Fragen zu klären und Lösungsansätze zu optimieren
- **Betriebssystem:**
 - Das System ist für den Einsatz unter Windows konzipiert.
- **Personelle Ressourcen:**
 - Der Projektleiter übernimmt den direkten Kundenkontakt.
 - Dank der Unterstützung eines Kollegen, der Zugriff über **AnyDesk** ermöglicht hat, konnte das Projekt erfolgreich auf dem Computer des Kunden getestet werden.
- **Infrastruktur:**
 - Arbeitsplatz: Büro mit individuell ausgestatteten Arbeitsplätzen.
 - Internetverbindung: Stabile Breitbandverbindung mit mindestens 100 Mbps, um schnelle Downloads von E-Mails und Anhängen zu gewährleisten.

2.4 Entwicklungsprozess

Der Entwicklungsprozess basiert auf einem iterativen Ansatz mit Elementen aus dem **agilen Modell**.

- Der Kunde hat seine Anforderungen über den Projektleiter (den Chef) mitgeteilt.
- Diese Anforderungen wurden an den Entwickler weitergegeben und in der ersten Phase umgesetzt.
- Noch bevor der Kunde das Ergebnis gesehen hat, wurden kleinere Änderungen basierend auf einer neuen Rückmeldung des Kunden vorgenommen.
- Der Entwicklungsprozess wurde so gestaltet, dass spätere Anpassungen möglich bleiben, falls der Kunde weitere Anforderungen hat.

3 Analysephase

3.1 Ist-Analyse

IST : der aktuelle Prozess für das Herunterladen von Rechnungen ist **manuell**.

Der Mitarbeiter muss alle E-Mails öffnen, die Anhänge herunterladen, den Speicherort auswählen und die Dateien nach **Eigentümern** organisieren. Da die Firma mehrere Eigentümer hat, die Rechnungen erhalten, ist dieser Prozess zeitaufwändig und fehleranfällig.

Zu verbessern: um den Prozess zu verbessern, kann die **Automatisierung** eingeführt werden. E-Mails könnten automatisch heruntergeladen und die Anhänge (Rechnungen oder Lieferscheine) entsprechend ihrem Inhalt und Eigentümer organisiert werden.

3.2 Wirtschaftlichkeitsanalyse

Das Projekt ist wirtschaftlich sinnvoll. Durch die Automatisierung des Prozesses wird der Zeitaufwand für das manuelle Öffnen von E-Mails, das Herunterladen von Dateien und die Organisation von Rechnungen deutlich reduziert.

Berechnungen (Schätzung):

1. Aktuelle Situation (manueller Prozess):

- **Zeitaufwand pro Rechnung:** 5 Minuten
- **Rechnungen pro Woche:** 100
- **Gesamtzeit pro Woche:** $100 \times 5 \text{ Minuten} = 500 \text{ Minuten}$ (ca. 8,3 Stunden)
- **Stundensatz eines Mitarbeiters:** 30 €
- **Kosten pro Woche:** $8,3 \text{ Stunden} \times 30 \text{ €} = 249 \text{ €}$

2. Nach der Automatisierung:

- **Zeitaufwand pro Rechnung:** 1 Minute
- **Rechnungen pro Woche:** 100
- **Gesamtzeit pro Woche:** $100 \times 1 \text{ Minute} = 100 \text{ Minuten}$ (ca. 1,7 Stunden)
- **Kosten pro Woche:** $1,7 \text{ Stunden} \times 30 \text{ €} = 51 \text{ €}$

3. Einsparungen:

- **Zeitersparnis pro Woche:** $8,3 \text{ Stunden} - 1,7 \text{ Stunden} = 6,6 \text{ Stunden}$
- **Kosteneinsparung pro Woche:** $249 \text{ €} - 51 \text{ €} = 198 \text{ €}$
- **Kosteneinsparung pro Jahr:** $198 \text{ €} \times 52 \text{ Wochen} = 10.296 \text{ €}$

Vorteile für das Unternehmen:

- **Zeitersparnis:** Mitarbeiter können 6,6 Stunden pro Woche für andere Aufgaben nutzen.
- **Fehlerreduktion:** Automatisierung reduziert Fehler und steigert die Zuverlässigkeit der Daten.
- **Effizienzsteigerung:** Klarer, strukturierter Prozess verbessert die Produktivität.

Fazit

Die Investition in das Projekt amortisiert sich schnell. Bereits innerhalb eines Jahres könnten mehr als **10.000 €** eingespart werden, was die Wirtschaftlichkeit und den Nutzen des Projekts deutlich unterstreicht.

EMAIL VERWALTEN

Automatisierte E-Mail-Anhangs Verwaltung

3.2.1 Make or Buy-Entscheidung

Es gibt andere Produkte wie **Xero**, **Quickbook** oder **Zapier** und andere die ähnliche Funktionen bieten. Diese Programme sind jedoch umfangreicher, haben eine steilere Lernkurve und konzentrieren sich auch auf die Verwaltung von Rechnungsinhalten.

Unsere Anwendung ist hingegen viel **einfacher** und **zielgerichteter**, da sie sich nur auf die Organisation der Dateien konzentriert. Dies erlaubt der Firma, das **Rechnungsprogramm frei zu wählen**, das am besten zu ihren Bedürfnissen passt. Besonders nützlich ist dies, wenn das Unternehmen bereits eine Buchhaltungssoftware verwendet, die sie nicht ersetzen möchten, die aber keine vergleichbare Funktion bietet.

Das Programm wird sinnvoll ungesetzt weil:

- **Maßgeschneiderte Lösung:** Unsere Anwendung ist speziell auf die Anforderungen des Kunden zugeschnitten. Sie bietet genau die Funktionalität, die benötigt wird, ohne unnötige Komplexität.
- **Kosteneffizienz:** Es lohnt sich mehr, eine eigene Anwendung zu entwickeln, als ein externes Produkt zu kaufen, da es sich um eine sehr spezifische Aufgabe handelt.
- **Kundenbindung:** Wir haben eine **langjährige Beziehung** mit dem Kunden und bieten andere Dienstleistungen an, die seinen Erwartungen entsprechen. Dadurch sind wir der bevorzugte Partner für dieses Projekt.
- **Wettbewerbsvorteil:** Unsere Lösung ist **einfacher und flexibler** als die Konkurrenzangebote und passt sich besser an die bestehenden Systeme des Kunden an.

3.2.2 Projektkosten

Die Kosten für die Durchführung des Projekts setzen sich wie folgt zusammen:

- **Personalkosten:**
 1. Da ich als Praktikant an diesem Projekt arbeite, fallen keine direkten Personalkosten an (0 €).
 2. Ein Mitarbeiter, der mich beim Installieren des Programms auf dem PC des Kunden unterstützt hat, hat für **2 Stunden x 69 €** gearbeitet.

Berechnung der Kosten:

- **Gesamtarbeitszeit:**
Geschätzte Durchführungszeit: **80 Stunden**

Tabelle 2: Berechnung geschätzte Durchführungszeit

Kategorie	Stunden	Stundensatz (€)	Kosten (€)
Personalkosten (Praktikant)	80	0,00	0,00
Personalkosten (Mitarbeiter)	2	69,00	138,00
Gesamtkosten			138,00

Abbildung 1: Berechnung geschätzte Durchführungszeit

Fazit:

Die Projektkosten belaufen sich insgesamt auf **138,00 €**, ausschließlich für die Unterstützung durch den Mitarbeiter. Da keine zusätzlichen Kosten anfallen, ist das Projekt äußerst kosteneffizient.

EMAIL VERWALTEN

Automatisierte E-Mail-Anhangs Verwaltung

3.3 Nutzwertanalyse

3.3.1 Nicht-monetärer Nutzen:

Der Nutzen des Projekts geht über reine Kostenersparnisse hinaus. Hier sind einige der nicht-monetären Vorteile, dargestellt als **Vorher-Nachher-Vergleich**:

Tabelle 3: Vorher-Nachher-Vergleich

Kriterium	Vorher (manuell)	Nachher (automatisiert)
Zeitaufwand	Hoher Zeitaufwand: ca. 5 Minuten pro Rechnung	Deutlich reduziert: ca. 1 Minute pro Rechnung
Fehleranfälligkeit	Hohe Fehlerquote durch manuelle Eingaben	Minimale Fehler dank Automatisierung
Flexibilität	Eingeschränkte Anpassung an bestehende Systeme	Integration mit bestehender Buchhaltungssoftware möglich
Benutzerfreundlichkeit	Komplexe und repetitive manuelle Prozesse	Einfache und intuitive Automatisierungslösung
Produktivität der Mitarbeiter	Zeitaufwendige Nebenaufgaben vermindern die Kernarbeit	Mitarbeiter können sich auf wichtigere Aufgaben konzentrieren
Skalierbarkeit	Zeitaufwand steigt proportional zur Anzahl der Rechnungen	Effizient auch bei wachsender Anzahl von Rechnungen

Abbildung 2: Vorher-Nachher-Vergleich

3.3.2 Wirtschaftlichkeitskoeffizient:

Eine qualitative Bewertung der Lösung könnte anhand einer Punkteskala (1–5) erfolgen, um die Verbesserung der einzelnen Kriterien zu quantifizieren.

"Die folgende Nutzwertanalyse basiert auf einer theoretischen Bewertung, die die erwarteten Verbesserungen und Vorteile der Automatisierung aufzeigt. Die Gewichtung und Punktzahlen wurden anhand der bekannten Anforderungen und Ziele des Kunden geschätzt."

Kriterium	Gewichtung (%)	Vorher (Punkte)	Nachher (Punkte)	Gewichtete Verbesserung
Zeitaufwand	30	2	5	$30 \times (5 - 2) = 90$
Fehleranfälligkeit	25	2	4	$25 \times (4 - 2) = 50$
Benutzerfreundlichkeit	20	3	5	$20 \times (5 - 3) = 40$
Produktivität der Mitarbeiter	15	2	4	$15 \times (4 - 2) = 30$
Skalierbarkeit	10	3	4	$10 \times (4 - 3) = 10$
Gesamtwert	100			220

Abbildung 3: Theoretischen Bewertung

Fazit:

Die Nutzwertanalyse zeigt, dass das Projekt nicht nur monetäre Vorteile bringt, sondern auch erhebliche Verbesserungen in Bereichen wie Produktivität, Skalierbarkeit und Fehlerreduzierung. Mit einem gewichteten Nutzwert von 220 Punkten bietet die Automatisierung eine klare Verbesserung gegenüber der vorherigen manuellen Lösung.

3.4 Anwendungsfälle

3.4.1 Anwendungsfälle des Projekts:

Das Projekt automatisiert den gesamten Prozess des Umgangs mit Rechnungen und Lieferscheinen. Es deckt folgende Anwendungsfälle ab:

- **Automatisiertes Herunterladen von E-Mails und Anhängen:**
 1. Verbindung zum E-Mail-Server herstellen (IMAP).
 2. Setzen max_uid von Inbox.
 3. Neue E-Mails ab dem letzten verarbeiteten UID (max_uid) abrufen.
 4. Anhänge aus den E-Mails herunterladen und in einem definierten Verzeichnis speichern.
 - **Kategorisierung und Verarbeitung von Anhängen:**
 1. Extraktion von Text aus den PDF-Dateien.
 2. Identifizieren und Umbenennen von Rechnungen (Re) und Lieferscheinen (Li).
 3. Erstellen einer Arbeitsstruktur (z. B. Ordner "Re_Li_processed").
 - **Zuordnung zu Eigentümern und Organisation:**
 1. Abgleich der extrahierten Daten mit der **DB_objekts.json** (Eigentümer).
 2. Umbenennen der Dateien mit dem **Verwaltungsnummer (verw.nr)** und weiteren relevanten Daten von Eigentümer.
 3. Verschieben der Dateien in eigentümerspezifische Ordner (automatische Erstellung neuer Ordner, falls nicht vorhanden).
-

3.4.2 Detaillierter Ablauf des Prozesses:

- **Programmstart:**
 1. Der Nutzer gibt seine **E-Mail-Credentials (Benutzername und Passwort)** und den **Verzeichnis** ein.
- **Erste Verbindung und Einrichtung:**
 1. Verbindung zum E-Mail-Server (IMAP) herstellen.
 2. Ermitteln der neuesten UID im Posteingang.
 3. Falls keine **max_uid.txt** existiert, wird diese erstellt und die aktuelle UID gespeichert.
- **Regelmäßige Überprüfung und Verarbeitung:**
 1. Alle 3 Minuten:
 1. Neue E-Mails ab der letzten gespeicherten UID abrufen.
 2. Anhänge herunterladen und den **max_uid** aktualisieren.
- **Verarbeitung der heruntergeladenen Dateien:**
 1. Erstellung eines Verzeichnisses namens **Re_Li_processed**.
 2. Identifikation der Dateien als **Rechnung (Re)** oder **Lieferschein (Li)** und Umbenennung.
- **Abgleich und Organisation:**
 1. Text aus den Dateien extrahieren.
 2. Abgleich mit der **DB_objekts.json** (Eigentümer).
 3. Umbenennung der Dateien mit der **Verwaltungsnummer (verw.nr)** des zugeordneten Eigentümers.
 4. Verschieben der Dateien in eigentümerspezifische Ordner (automatische Erstellung bei Bedarf).

EMAIL VERWALTEN

Automatisierte E-Mail-Anhangs Verwaltung

- **Ergebnis:**

1. Alle Dateien mit Eigentümer sind korrekt benannt und in den entsprechenden Ordnern organisiert.

3.4.3 Visualisierung durch ein Aktivitätsdiagramm:

Das oben beschriebene Szenario lässt sich durch ein Aktivitätsdiagramm darstellen. Die Hauptaktivitäten könnten so aussehen:

EMAIL VERWALTEN

Automatisierte E-Mail-Anhangs Verwaltung

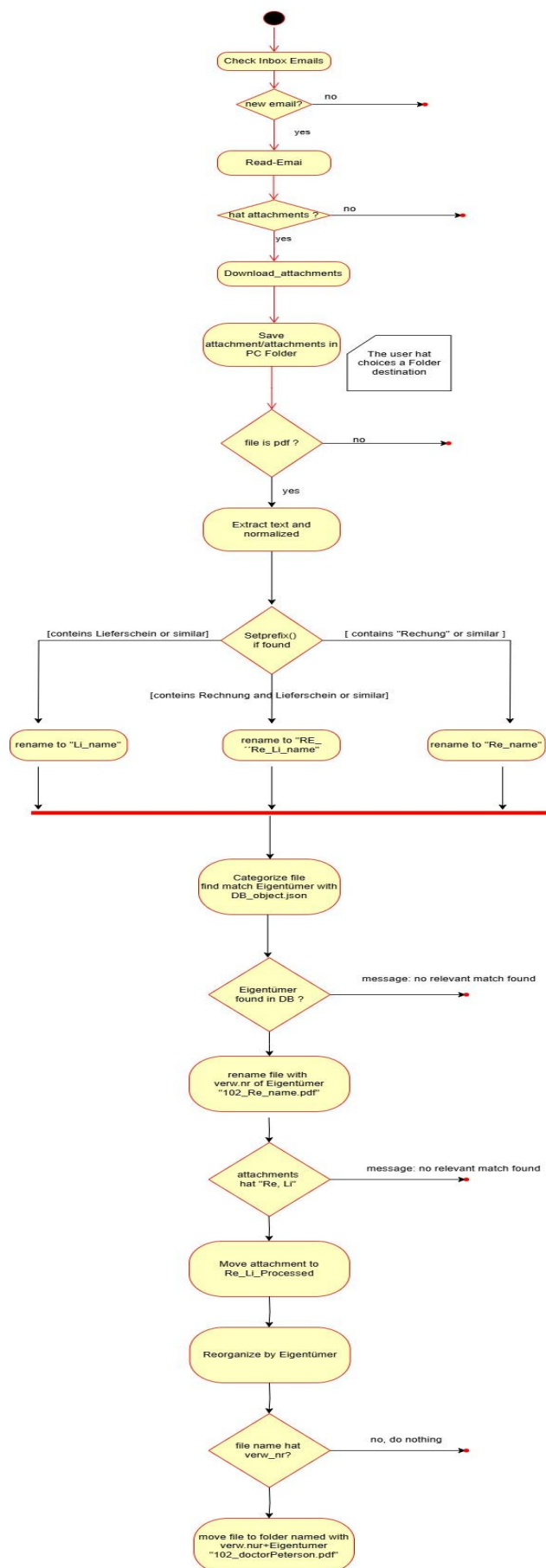


Abbildung 4: Aktivitätsdiagramm

3.5 Qualitätsanforderungen

3.5.1 Qualitätsanforderungen

Performance:

- **Antwortzeit:**
Die Anwendung soll E-Mails und Anhänge effizient verarbeiten. Idealerweise sollte die Verarbeitung einer E-Mail nicht länger als **5 Sekunden** dauern, und das Bearbeiten eines Dokuments nicht mehr als **10 Sekunden**.
- **Skalierbarkeit:**
Die Anwendung muss in der Lage sein, ein erhöhtes E-Mail-Aufkommen zu bewältigen, ohne an Leistung zu verlieren. Eine effiziente Speicher- und Ressourcenverwaltung ist erforderlich.
- **Zuverlässigkeit:**
Das System muss robust sein und Fehler abfangen können, ohne Daten zu verlieren. Im Falle eines Fehlers darf die Anwendung keine E-Mails oder Anhänge verlieren.
- **Mehrsprachigkeit:**
Die Anwendung muss Texte sowohl in **Deutsch** als auch in **Englisch** erkennen können.

Benutzerfreundlichkeit:

- **Einfachheit:**
Der Benutzer soll keinen Kontakt mit dem Code haben müssen. Es reicht aus, seine Daten über die Konsole einzugeben.

Effizienz:

- **Ressourcenverbrauch:**
Die Anwendung soll ressourcenschonend arbeiten und nicht unnötig CPU, Speicher oder Bandbreite beanspruchen. Sie darf andere Systemprozesse nicht beeinträchtigen.
- **Automatisierung:**
Die Prozesse sollen so weit wie möglich automatisiert sein, z. B. das Klassifizieren von Dokumenten, das Extrahieren von Rechnungsdaten und die Integration mit anderen Systemen (z. B. Buchhaltungssoftware).
- **Systemkompatibilität:**
Die Anwendung muss unter **Windows** funktionieren.

Sicherheit:

- **Datenschutz und Zugangsbeschränkungen:**
Sicherstellen, dass nur autorisierte Nutzer Zugriff auf die Daten und die Anwendung haben.

Wartbarkeit :

- **Modularer Code:**
Die Anwendung muss modular aufgebaut sein, um zukünftige Updates, Wartungen und Verbesserungen zu erleichtern.
- **Einfachheit von Updates:**
Der Code soll so gestaltet sein, dass neue Funktionen oder Fehlerbehebungen implementiert werden können, ohne den laufenden Betrieb zu beeinträchtigen.

Zuverlässigkeit:

- **Fehlertoleranz:**
Fehler und Ausnahmen sollen korrekt behandelt werden, mit klaren Meldungen und Optionen zur Wiederherstellung für die Benutzer.
- **Verfügbarkeit:**
Die Anwendung soll kontinuierlich verfügbar sein. Wartungsarbeiten sollten außerhalb der Arbeitszeiten durchgeführt werden, um den Betrieb nicht zu stören.

EMAIL VERWALTEN

Automatisierte E-Mail-Anhangs Verwaltung

3.6 Lastenheft/Fachkonzept

3.6.1 Projektbeschreibung

Ziel des Projekts

Die Anwendung soll folgende Aufgaben automatisieren:

- Funktionieren unter **Windows**.
- Verbindung zu einem E-Mail-Server über **IMAP** (Port 993), um E-Mails mit Anhängen herunterzuladen.
- **Textextraktion** aus Anhängen, insbesondere **PDF-Dateien**.
- Erkennung von Text mit **Flexibilität**, auch bei kleinen Abweichungen (z. B. fehlende Buchstaben, Sonderzeichen oder Akzentzeichen).
- Umbenennung der Dateien je nach Inhalt: Rechnung, Lieferschein oder beides.
- Abgleich des extrahierten Texts mit einer **Eigentümer-Datenbank (DB_objecks.json)**.
- Umbenennung der Dateien mit der **Verwaltungsnummer (verw.nr)** des Eigentümers.
- Organisation der Dateien in **eigentümerspezifischen Ordnern**.

3.6.2 Anforderungen

E-Mail-Verbindung und Verarbeitung:

- **IMAP-Verbindung:**
 - Die Anwendung muss eine Verbindung zu einem E-Mail-Postfach herstellen können (Server: **IONOS**, Port: 993).
- **Herunterladen der Anhänge:**
 - Alle Anhänge aus den E-Mails müssen heruntergeladen und im Ordner gespeichert werden (z. B. `re_`).
- **Textextraktion:**
 - Der Text aus den Anhängen muss mithilfe geeigneter Tools (z. B. `pypdf`) extrahiert werden.
- **Dateiumbenennung:**
 - Dateien werden umbenannt, sobald sie als Rechnung oder Lieferschein erkannt werden. (Prefix)
 - Falls der Text den Namen eines Eigentümers enthält, wird die Datei mit der **Verwaltungsnummer (verw.nr)** des Eigentümers umbenannt.
- **Dateiorganisation:**
 - Dateien werden in den Ordner **Re_Li_Processed** verschoben.
 - Dateien mit einem zugeordneten Eigentümer werden zusätzlich in einen eigentümerspezifischen Ordner verschoben. Falls der Ordner nicht existiert, wird er erstellt.

3.6.3 Detaillierte Anforderungen

E-Mail-Verbindung und Verarbeitung

- **IMAP-Verbindung:**
 - Die Anwendung muss sich mit einem **IMAP-Server (IONOS)** verbinden können, um E-Mails abzurufen

3.6.4 Technische Anforderungen

Systemkompatibilität

- Die Anwendung muss unter **Windows** lauffähig sein.
- Sie soll mit **Python** programmiert werden und gängige Bibliotheken verwenden.

Sicherheitsanforderungen

- **Datensicherheit:**
 - E-Mail-Verbindungen müssen über **SSL/TLS** verschlüsselt sein.
 - Nur relevante Daten dürfen extrahiert und gespeichert werden.

EMAIL VERWALTEN

Automatisierte E-Mail-Anhangs Verwaltung

3.6.5 Implementierung und Technologien

Verwendete Technologien

- **IMAP-Verbindung:** `imaplib` für den Zugriff auf E-Mails.
- **E-Mail-Verarbeitung:** `email`-Bibliothek zur Analyse und Verarbeitung von Anhängen.
- **Textextraktion:** `pyPDF2` für die Extraktion von Text aus PDF-Dateien.
- **Benutzeroberfläche:** Terminal (Textbasiert).

4 Entwurfsphase

4.1 Zielplattform

4.1.1 Kriterien zur Auswahl der Zielplattform:

- **Programmiersprache:**
 - Die Anwendung wird in **Python** entwickelt, da diese Sprache leistungsstarke Bibliotheken zur Verarbeitung von E-Mails, Anhängen und Text bietet.
 - Python ermöglicht eine schnelle Entwicklung und einfache Erweiterbarkeit.
- **Alternative Evaluierung:**
 - Während der Entwicklungsphase wurde auch Google Document AI getestet, insbesondere der Processor Parser Invoice.
 - Dieser Service funktionierte fehlerfrei bei der Analyse und Strukturierung von Rechnungsdaten.
 - Grund für den Ausschluss: Aufgrund der laufenden Kosten für die Nutzung von Google Document AI wurde entschieden, auf eine lokale Lösung mit Python und seinen Bibliotheken umzusteigen, um die langfristigen Kosten zu minimieren.
- **Datenbank:**
 - „Der Kunde stellte ein PDF-Dokument mit Tabellen zur Verfügung, das die Informationen über die Eigentümer enthielt, die in der Firma Rechnungen erhalten. Wir haben das Information in JSON gespeichert.“
 - JSON wird als Datenbankformat verwendet, um Informationen über die Eigentümer (z. B. Verwaltungsnummer, Adresse, Name) zu speichern.
 - JSON ist leichtgewichtig, einfach zu bearbeiten und benötigt keine zusätzliche Datenbank-Infrastruktur. Bietet Persistenz der Daten zu gewährleisten, ohne dass eine separate Datenbank-Infrastruktur erforderlich ist.
- **Client/Server-Architektur:**
 - Die Anwendung ist als Standalone-Client konzipiert, um einfach lokal auf Windows-Systemen betrieben zu werden.
 - Eine Client/Server-Architektur ist nicht erforderlich, da die Anwendung unabhängig auf jedem System laufen kann.
- **Hardware:**
 - Die Zielplattform sind Standard-PCs mit Windows als Betriebssystem.
- **Anforderungen:**
 - Prozessor: Dual-Core oder höher.
 - RAM: Mindestens 4 GB für eine flüssige Ausführung.
 - Speicherplatz: 500 MB für die Anwendung und temporäre Dateien.
- **Netzwerk:**
 - Eine stabile Internetverbindung ist erforderlich, um über IMAP auf E-Mails zugreifen zu können.
 - Port 993 für SSL/TLS-verschlüsselte Verbindungen muss offen sein.
 - Benutzerfreundlichkeit:
 - Die Anwendung ist für den Einsatz durch Endnutzer ohne Programmierkenntnisse konzipiert.
 - Sie läuft im Terminal, wo der Nutzer lediglich seine Zugangsdaten eingeben muss.

Zusammenfassung:

Die Zielplattform wurde so gewählt, dass sie kosteneffizient, einfach einzurichten und für den Nutzer leicht bedienbar ist. Die lokale Verarbeitung der Dateien stellt sicher, dass keine sensiblen Daten über externe Server laufen, was die Sicherheit erhöht.

4.2 Architekturdesign

4.2.1 Beschreibung der gewählten Architektur:

Das Projekt verwendet eine modulare Architektur, bei der jede Funktionalität in einer separaten Struktur organisiert ist. Die Hauptstruktur besteht aus mehreren Ordnern (z. B. config, DB, emails, imap, processing, tests, und utils), die spezifische Aufgaben oder Bereiche abdecken. Diese klare Trennung sorgt für eine übersichtliche Organisation des Codes und erleichtert die Wartung.

Begründung der Architektur:

- **Modularität:** Jede Funktion ist in unabhängigen Modulen organisiert, was die Wiederverwendbarkeit und Lesbarkeit des Codes erhöht.
- **Flexibilität:** Neue Funktionen können leicht hinzugefügt werden, ohne bestehende Module zu beeinträchtigen.
- **Wartbarkeit:** Änderungen oder Fehlerbehebungen sind durch die klare Trennung der Aufgabenbereiche einfacher umzusetzen.

Bewertung und Frameworks:

- Es werden keine speziellen Frameworks wie MVC verwendet, da die modulare Architektur die Anforderungen des Projekts vollständig erfüllt.
- Stattdessen kommen leistungsstarke Python-Bibliotheken wie imaplib, pypdf, und rapidfuzz zum Einsatz, die die Funktionalität der einzelnen Module unterstützen.

4.3 Entwurf der Benutzeroberfläche

Entscheidung für die Benutzeroberfläche:

- Die Anwendung verwendet aktuell die Konsole als Benutzeroberfläche. Dies wurde aus folgenden Gründen gewählt:
 - **Einfachheit:** Der Kunde muss keine Aktionen durchführen. Alle Eingaben (z. B. E-Mail-Zugangsdaten und Speicherpfad für Anhänge) werden von uns direkt über die Konsole eingegeben.
 - **Zugriff:** Wir haben direkten Zugang zum PC des Kunden und können die notwendigen Daten eingeben, ohne den Kunden einzubeziehen.
 - **Keine Notwendigkeit für eine GUI:** Da der Kunde nicht direkt mit der Anwendung interagiert, ist eine grafische Benutzeroberfläche (GUI) nicht erforderlich.

Frühere Überlegungen zur GUI:

Zu Beginn des Projekts wurde angenommen, dass der Kunde die Anwendung selbst installieren und nutzen würde. Daher wurde eine einfache GUI mit Tkinter entwickelt, um dem Kunden die Eingabe von Zugangsdaten und Einstellungen zu erleichtern. Diese Lösung wurde jedoch aufgrund der oben genannten Punkte verworfen.

Zukünftige Möglichkeiten:

Die frühere GUI bleibt in einem separaten GitHub-Repository erhalten, falls sie in Zukunft benötigt wird, z. B. wenn der Kunde mehr Kontrolle über die Anwendung haben möchte.

Fazit:

Die Verwendung der Konsole ist aktuell die effizienteste Lösung, da sie keine zusätzliche Entwicklungszeit für eine GUI erfordert und den Kunden vollständig entlastet. Die Entscheidung, eine einfache und flexible Konsole zu nutzen, passt optimal zu den aktuellen Anforderungen des Projekts.

4.4 Datenmodell

Die Anwendung verwendet eine JSON-Datei (db_objects.json) als Datenbank, um Informationen über die Eigentümer und deren zugehörige Objekte zu speichern. Diese Datei enthält strukturierte Daten, die einfach zugänglich und effizient zu verarbeiten sind.

Die JSON-Datei besteht aus einer Liste von Objekten, die die wichtigsten Informationen zu jedem Eigentümer und seinen Objekten enthalten.

```
[
  {
    "verw_nr": "102",
    "objekt": "EXAMPLE Straße 52",
    "plz": "20148",
    "eigentümer": "Dr. Meirowitz",
    "wo": "1",
    "gew": "3",
    "pp": "1",
    "insgesamt": "4"
  },
  {
    "verw_nr": "103",
    "objekt": "EXAMPLE Straße 22",
    "plz": "20148",
    "eigentümer": "Fastighets Aktiebolaget Flodmynningen",
    "wo": "3",
    "gew": "1",
    "pp": "1",
    "insgesamt": "4"
  }
]
```

Abbildung 5: DB_objects.json Beispiel

Die Daten werden bei der Verarbeitung von Anhängen genutzt, um extrahierte Informationen (z. B. aus Rechnungen) mit den gespeicherten Daten abzugleichen.

Wenn ein Eigentümer erkannt wird, werden die entsprechenden Informationen verwendet, um die Datei umzubenennen und in den richtigen Ordner zu verschieben.

Die Vorteile dieser Datenstruktur:

- Flexibilität: JSON ermöglicht einfache Änderungen und Erweiterungen der Daten.
- Einfacher Zugriff: Die Daten können direkt in Python geladen und verarbeitet werden.
- Leichtgewichtig: Keine zusätzliche Datenbank-Infrastruktur erforderlich.

4.5 Maßnahmen zur Qualitätssicherung

Manuelle Tests:

Die meisten Tests wurden manuell durchgeführt, um sicherzustellen, dass die Kernfunktionen wie E-Mail-Verarbeitung, Textextraktion und Dateiorganisation ordnungsgemäß funktionieren.

Beispiele für manuelle Tests:

- Abrufen von E-Mails mit Anhängen über IMAP.
- Überprüfung der Textextraktion aus realen Rechnungen im PDF-Format.
- Abgleich der extrahierten Daten mit der JSON-Datenbank (db_objects.json).
- Umbenennen und Verschieben von Dateien in die entsprechenden Ordner.

Echte Testumgebung:

Während der Entwicklung wurde ein IONOS E-Mail-Konto verwendet, um die Verbindung, das Abrufen und die Verarbeitung von E-Mails unter realistischen Bedingungen zu testen.

Reale Rechnungen wurden eingesetzt, um sicherzustellen, dass die Texterkennung und die Zuordnung zu Eigentümern korrekt funktionieren. Auf English und Deutsch.

Automatisierte Tests:

Zur Unterstützung der Qualitätssicherung wurden automatisierte Tests mit pytest entwickelt. Diese decken spezifische Bereiche der Anwendung ab:

- attachments/test_data_handler.py: Testet die Verarbeitung und Verwaltung von Anhängen.
- emails/test_file_handler.py: Testet die Handhabung und Verarbeitung von Dateien in E-Mails.
- processing/test_file_handler.py: Testet die Verarbeitung und Organisation von Dateien nach der Textextraktion.

4.6 Pflichtenheft/Datenverarbeitungskonzept

4.6.1 Projektbeschreibung

Ziel des Projekts

Die Anwendung hat das Ziel, die Verarbeitung von E-Mails und Anhängen zu automatisieren. Dabei werden folgende Funktionen umgesetzt:

- **System:** Funktionieren unter **Windows**.
- **E-Mail-Verarbeitung:** Verbindung zu einem **IMAP-Server** (IONOS, Port 993), um E-Mails mit Anhängen herunterzuladen.
- **Textextraktion:** Extraktion von Text aus Anhängen, insbesondere aus **PDF-Dateien**, mithilfe von `pypdf`.
- **Flexible Textanalyse:** Erkennung von Text mit **Flexibilität**, auch bei kleinen Abweichungen (z. B. fehlende Buchstaben, Sonderzeichen oder Akzentzeichen). Dies wird durch den Einsatz von **rapidfuzz** anstelle von regulären Ausdrücken realisiert.
- **Dateiumbenennung:**
 - Umbenennung von Dateien je nach Inhalt (Rechnung, Lieferschein oder beides).
 - Hinzufügen der Verwaltungsnummer (`verw.nr`) des Eigentümers, falls der Text einen entsprechenden Namen enthält.
- **Dateiorganisation:**
 - Dateien werden in den Ordner **Re_Li_Processed** verschoben.
 - Dateien mit einem zugeordneten Eigentümer werden in spezifische **Eigentümer-Ordner** verschoben. Falls der Ordner nicht existiert, wird er automatisch erstellt.

EMAIL VERWALTEN

Automatisierte E-Mail-Anhangs Verwaltung

○

4.6.2 Anforderungen

E-Mail-Verbindung und Verarbeitung:

- **IMAP-Verbindung:**
 - Die Anwendung muss eine Verbindung zu einem IMAP-Server herstellen können (Server: IONOS, Port: 993).
- **Herunterladen der Anhänge:**
 - E-Mails müssen heruntergeladen und im Ordner gespeichert werden.
- **Textextraktion:**
 - Der Text aus den Anhängen wird mithilfe von pypdf extrahiert, um die besten Ergebnisse zu erzielen.
- **Textanalyse:**
 - Die Zuordnung der extrahierten Daten zur JSON-Datenbank erfolgt mit rapidfuzz, um eine höhere Flexibilität und Effizienz bei der Erkennung von Eigentümerdaten zu gewährleisten.
- **Dateiumbenennung:**
 - Dateien werden zunächst basierend auf ihrem Inhalt (Rechnung, Lieferschein oder beide) mit einem Präfix versehen:
 - Rechnung: Re_
 - Lieferschein: Li_
 - Rechnung und Lieferschein: Re_Li_
 - Beispiel: Re_Garner.pdf, Li_Schmidt.pdf, Re_Li_Bauer.pdf
 - Wenn der Text den Namen eines Eigentümers enthält, wird die Verwaltungsnummer (verw.nr) des Eigentümers hinzugefügt.
 - Endergebnis: 102_Re_Garner.pdf
- **Dateiorganisation:**
 - Dateien werden in den Ordner Re_Li_Processed verschoben.
 - Dateien mit einem zugeordneten Eigentümer werden in eigentümerspezifische Ordner verschoben.

4.6.3 Detaillierte Anforderungen

E-Mail-Verbindung und Verarbeitung:

- Die Anwendung muss sich mit einem **IMAP-Server (IONOS)** verbinden können, um E-Mails und Anhänge abzurufen.

4.6.4 Technische Anforderungen

Systemkompatibilität:

- Die Anwendung muss unter **Windows** lauffähig sein.
- Die Entwicklung erfolgt mit **Python** und gängigen Bibliotheken.

Sicherheitsanforderungen:

- **Datensicherheit:**
 - E-Mail-Verbindungen müssen über **SSL/TLS** verschlüsselt sein.
 - Nur relevante Daten dürfen extrahiert und gespeichert werden.

4.6.5 Implementierung und Technologien

Verwendete Technologien:

- **IMAP-Verbindung:** imaplib für den Zugriff auf E-Mails.
- **E-Mail-Verarbeitung:** email-Bibliothek zur Analyse und Verarbeitung von Anhängen.
- **Textextraktion:** pypdf, da es zuverlässiger ist als PyPDF2, das deprecated Funktionen enthält.
- **Textanalyse:** rapidfuzz, um eine effiziente und flexible Zuordnung zwischen extrahierten Texten und der JSON-Datenbank zu ermöglichen.
- **Benutzeroberfläche:** Die Anwendung läuft über eine textbasierte Konsole.

5 Implementierungsphase

5.1 Implementierung der Datenstrukturen

Derzeit wird keine zusätzliche Datenbank verwendet, da die Anhänge direkt auf dem Computer des Kunden gespeichert werden. Es ist aktuell nicht notwendig, weitere Informationen aus den E-Mails zu speichern. Die Zuordnung der Rechnungen erfolgt jedoch mithilfe einer *JSON*-Datei, die Daten der Eigentümer enthält und über ein eigenes Modul verarbeitet wird.

Das Modul `data_loader.py` dient dazu, die *JSON*-Daten zu laden und in einer strukturierten Form zurückzugeben.

```
import json

def load_parameters_from_db(json_file):
    """
    Loads parameters from the JSON file and organizes them by 'verw_nr'.

    Args:
        json_file (str): Path to the JSON file.

    Returns:
        list: A list of dictionaries containing 'verw_nr', 'objekt', and 'eigentümer'.
    """
    try:
        with open(json_file, "r", encoding="utf-8") as f:
            data = json.load(f)
            return [
                {"verw_nr": entry["verw_nr"], "objekt": entry["objekt"], "eigentümer": entry["eigentümer"]}
                for entry in data if "verw_nr" in entry and "objekt" in entry and "eigentümer" in entry
            ]
    except Exception as e:
        print(f"An error occurred while loading parameters from DB: {e}")
        return []
```

Verwendung des Moduls:

Das Modul wird von anderen Komponenten des Projekts aufgerufen, um die Eigentümerdaten in strukturierter Form bereitzustellen. Zum Beispiel:

```
db_file = resource_path("DB/db_objects.json")
print(f"Database file path: {db_file}")
if not os.path.exists(db_file):
    print(f"Database file not found: {db_file}")
    return
```

`resource_path` wird verwendet, um sicherzustellen, dass der Pfad zur *JSON*-Datei unabhängig von der Umgebung (Entwicklung oder kompiliertes Programm) korrekt gefunden wird.

```
# Process attachments
process_attachments(attachment_folder, destination_folder, db_file)
organize_files_by_eigentuemer(destination_folder, db_file)
```


EMAIL VERWALTEN

Automatisierte E-Mail-Anhangs Verwaltung

Das Modul `config/config.py` speichert die vom Benutzer eingegebenen E-Mail-Zugangsdaten (Benutzername und Passwort) sowie den Pfad für den Speicherort der Anhänge in einer JSON-Datei (`config.json`). Diese Datei wird während der ersten Konfiguration automatisch erstellt.

```
# Basic logging configuration
logging.basicConfig(level=logging.INFO)

CONFIG_FILE = os.path.join(os.path.dirname(os.path.abspath(__file__)), "config.json")

def load_config(config_file=CONFIG_FILE):
    """
    Load the configuration from a JSON file, or run initial setup if the file does not exist.

    Args:
        config_file (str): Path to the configuration file.

    Returns:
        dict: The configuration data.
    """
    if not os.path.exists(config_file):
        logging.warning(f"Configuration file not found at {config_file}. Running initial setup...")
        return initial_setup(config_file)

    try:
        with open(config_file, "r", encoding="utf-8") as f:
            return json.load(f)
    except json.JSONDecodeError as e:
        logging.error(f"Error decoding JSON from {config_file}: {e}")
        raise
    except Exception as e:
        logging.error(f"Error loading configuration: {e}")
        raise
```

Funktionsweise:

Wenn die Konfigurationsdatei (`config.json`) nicht existiert, startet die Anwendung einen Einrichtungsprozess. Der Benutzer gibt die Zugangsdaten und den Speicherort für Anhänge ein.

```
def initial_setup(config_file=CONFIG_FILE):
    """
    Perform the initial setup to configure the IMAP credentials and attachment folder.

    Args:
        config_file (str): Path to save the configuration file.

    Returns:
        dict: The created configuration data.
    """
    logging.info("Initial setup: Provide the email credentials and attachment folder path.")
    from config.credentials import input_imap_credentials

    imap_username, imap_password = input_imap_credentials()
    attachment_folder = input("Enter the full path for the attachment folder: ").strip()

    # Validate and create folder
    validate_and_create_folder(attachment_folder)

    config = {
        "imap_username": imap_username,
        "imap_password": imap_password,
        "attachment_folder": attachment_folder,
    }
    save_config(config, config_file)
    return config
```

EMAIL VERWALTEN

Automatisierte E-Mail-Anhangs Verwaltung

Der angegebene Ordner wird überprüft und bei Bedarf erstellt.

Die Daten werden dann in der *JSON*-Datei gespeichert, damit sie bei späteren Starts der Anwendung wiederverwendet werden können.

```
def save_config(config, config_file=CONFIG_FILE):
    """
    Save configuration data to a JSON file.

    Args:
        config (dict): Configuration data to save.
        config_file (str): Path to the configuration file.
    """
    try:
        with open(config_file, "w", encoding="utf-8") as f:
            json.dump(config, f, indent=4)
            logging.info(f"Configuration saved to {config_file}")
    except Exception as e:
        logging.error(f"Error saving configuration: {e}")
        raise
```

In das Modul `tracker.py` wird eine Datei namens `max_uid.txt` verwendet, um die **UID (Unique Identifier)** des zuletzt verarbeiteten E-Mails zu speichern. Dies stellt sicher, dass bei zukünftigen Durchläufen keine E-Mails erneut verarbeitet werden, die bereits bearbeitet wurden.

```
def get_uid_tracker_file():
    """
    Determine the correct path for max_uid.txt based on the environment (PyInstaller or development).
    """
    try:
        # Route PyInstaller
        base_path = sys._MEIPASS
    except AttributeError:
        # Route development
        base_path = os.path.abspath(".")
    return os.path.join(base_path, "max_uid.txt")

UID_TRACKER_FILE = get_uid_tracker_file()
```

EMAIL VERWALTEN

Automatisierte E-Mail-Anhangs Verwaltung

Funktionsweise:

Die Datei `max_uid.txt` wird je nach Umgebung (Entwicklung oder kompiliertes Programm) an der richtigen Stelle abgelegt, um maximale Kompatibilität sicherzustellen.

Speichern und Abrufen des letzten UIDs:

- Beim Start der Anwendung wird der letzte gespeicherte UID aus `max_uid.txt` gelesen.
- Falls die Datei nicht existiert, wird ein Standardwert (z. B. 0) zurückgegeben und die Datei erstellt.
- Nach der Verarbeitung von E-Mails wird der neue höchste UID in der Datei gespeichert.

```
def get_last_saved_uid():  
    """  
    Get the last saved UID from max_uid.txt.  
    Returns 0 if the file does not exist.  
    """  
    if os.path.exists(UID_TRACKER_FILE):  
        with open(UID_TRACKER_FILE, "r") as file:  
            return int(file.read().strip())  
    save_last_uid(0)  
    return 0
```

```
def save_last_uid(uid):  
    with open(UID_TRACKER_FILE, "w") as file:  
        file.write(str(uid))
```

EMAIL VERWALTEN

Automatisierte E-Mail-Anhangs Verwaltung

Die Datei stellt sicher, dass die Verarbeitung effizient ist, indem sie doppelte Bearbeitung von E-Mails vermeidet.

Es wird benutzt in `main.py` erstal in `setup_configuration()`

```
def setup_configuration():
    """
    Ensure the configuration file is set up before running the main program.
    """
    try:
        config = load_config()
        print("Configuration loaded successfully.")
        # Check if max_uid.txt exists; if not, create it with the max UID from the server
        max_uid_file = resource_path("max_uid.txt")
        if not os.path.exists(max_uid_file):
            max_uid = get_max_uid_from_server()
            if max_uid > 0:
                with open(max_uid_file, "w") as f:
                    f.write(str(max_uid))
                print(f"Max UID ({max_uid}) saved to {max_uid_file}.")
            else:
                print("Unable to determine max UID. Starting from 0.")
        else:
            print("Max UID already exists in file.")

    except Exception:
        print("No configuration found. Running initial setup...")
        initial_setup()
```

```
def get_max_uid_from_server():
    """
    Connects to the IMAP server and retrieves the maximum UID.

    Returns:
        int: The maximum UID on the server, or 0 if it cannot be determined.
    """
    try:
        with get_imap_connection() as mail:
            mail.select("INBOX")
            result, data = mail.search(None, "ALL")
            if result == "OK" and data and data[0]:
                max_uid = int(data[0].split()[-1]) # Get the highest UID
                return max_uid
    except Exception as e:
        print(f"Error retrieving max UID from server: {e}")
    return 0
```

EMAIL VERWALTEN

Automatisierte E-Mail-Anhangs Verwaltung

Und dann wird in jedes Cyclo update:

```
while True:
    mail = get_imap_connection()

    # Track last processed UID
    last_uid = get_last_saved_uid()
    print(f"Last saved UID: {last_uid}")

    # Download emails and process attachments
    new_last_uid = process_emails_since(mail, last_uid)
    if new_last_uid > last_uid:
        save_last_uid(new_last_uid)

    # Process attachments
    process_attachments(attachment_folder, destination_folder, db_file)
    organize_files_by_eigentuemer(destination_folder, db_file)

    mail.logout()
    print("Waiting 3 min before the next run...")
    print("new_last_uid", new_last_uid)
    time.sleep(180)
```

Der `max_uid` wird in einer `.txt`-Datei gespeichert, um Persistenz, einfache Leseberechtigungen und eine unkomplizierte Verwaltung ohne zusätzliche Komplexität zu gewährleisten.

5.2 Implementierung der Benutzeroberfläche

Die Benutzeroberfläche der Anwendung wird derzeit über die **Konsole** bereitgestellt, da sich herausgestellt hat, dass das Programm nicht direkt vom Kunden genutzt wird, sondern von uns auf dem Kunden-PC installiert und verwaltet wird.

GUI als vorbereitete Option:

Zu Beginn wurde die Entwicklung einer grafischen Benutzeroberfläche (*GUI*) in Betracht gezogen, da ursprünglich angenommen wurde, dass der Kunde die Anwendung selbstständig nutzen würde. Für die Implementierung der GUI wurde **Tkinter** gewählt, da es aufgrund seiner Einfachheit, der leichten Erlernbarkeit und seiner nahtlosen Integration mit Python sowie dem Betriebssystem eine ideale Lösung darstellt. Die GUI ist in einer separaten Branch vorbereitet und kann bei Bedarf in der Zukunft aktiviert werden.

EMAIL VERWALTEN

Automatisierte E-Mail-Anhangs Verwaltung

Beispielcode der GUI:

Hier ein Ausschnitt aus dem vorbereiteten GUI-Code:

```
def start_app():
    root = tk.Tk()
    root.title("Email PDF Downloader")

    tk.Label(root, text="Email:").grid(row=0, column=0, sticky=tk.W, padx=5, pady=5)
    email_entry = tk.Entry(root, width=30)
    email_entry.grid(row=0, column=1, padx=5, pady=5)

    tk.Label(root, text="Password:").grid(row=1, column=0, sticky=tk.W, padx=5, pady=5)
    password_entry = tk.Entry(root, show='*', width=30)
    password_entry.grid(row=1, column=1, padx=5, pady=5)

    tk.Label(root, text="Email Provider:").grid(row=2, column=0, sticky=tk.W, padx=5, pady=5)
    provider_combobox = ttk.Combobox(root, values=list(EMAIL_PROVIDERS.keys()))
    provider_combobox.grid(row=2, column=1, padx=5, pady=5)

    def submit():
        email_user = email_entry.get()
        email_pass = password_entry.get()
        provider = provider_combobox.get()

        server = EMAIL_PROVIDERS.get(provider)

        if not server:
            print("Invalid provider selected.")
            return

        mail = connect_imap(server, email_user, email_pass)

        if mail:
            re_dir = select_folder()

            # Save the user info (folder, email, provider) to JSON
            save_user_info(re_dir, email_user, provider)

            # Open a new window to display success message
            success_window = tk.Toplevel(root)
            success_window.title("Connection Successful")
            success_window.geometry("300x100")

            # Center the success window on the main window
            x = root.winfo_x() + (root.winfo_width() // 2) - 150 # 150 is half of the success window width
            y = root.winfo_y() + (root.winfo_height() // 2) - 50 # 50 is half of the success window height
            success_window.geometry(f"{x}+{y}")

    # Pack the labels (one below the other)
    email_label.pack(pady=5)
    provider_label.pack(pady=5)

    # Start checking inbox in a new thread
    threading.Thread(target=start_checking_inbox, args=(mail, re_dir, Path("data/email_info.json")), daemon=True).start()

    # Hide the main window
    root.withdraw()

    # Submit button
    submit_button = tk.Button(root, text="Start", command=submit)
    submit_button.grid(row=3, columnspan=2, pady=10)

    # Center the main window on the screen
    root.update_idletasks() # Update "requested size" of the window
    window_width = root.winfo_width()
    window_height = root.winfo_height()

    # Get the screen width and height
    screen_width = root.winfo_screenwidth()
    screen_height = root.winfo_screenheight()

    # Calculate the x and y coordinates for the Tkinter window
    x = (screen_width // 2) - (window_width // 2)
    y = (screen_height // 2) - (window_height // 2)

    # Set the dimensions of the window
    root.geometry(f"{window_width}x{window_height}+{x}+{y}")
```

EMAIL VERWALTEN

Automatisierte E-Mail-Anhangs Verwaltung

Screenshot-Beispiel:

Ein Screenshot der vorbereiteten GUI wird hier eingefügt, um die mögliche Benutzeroberfläche zu visualisieren.

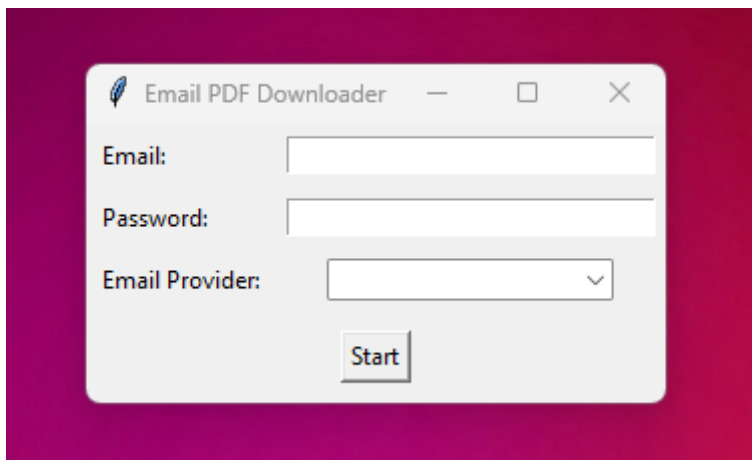


Abbildung 6: GUI Benutze Oberfläche

GUI zur Auswahl des Ordners, in dem Anhänge gespeichert werden sollen:

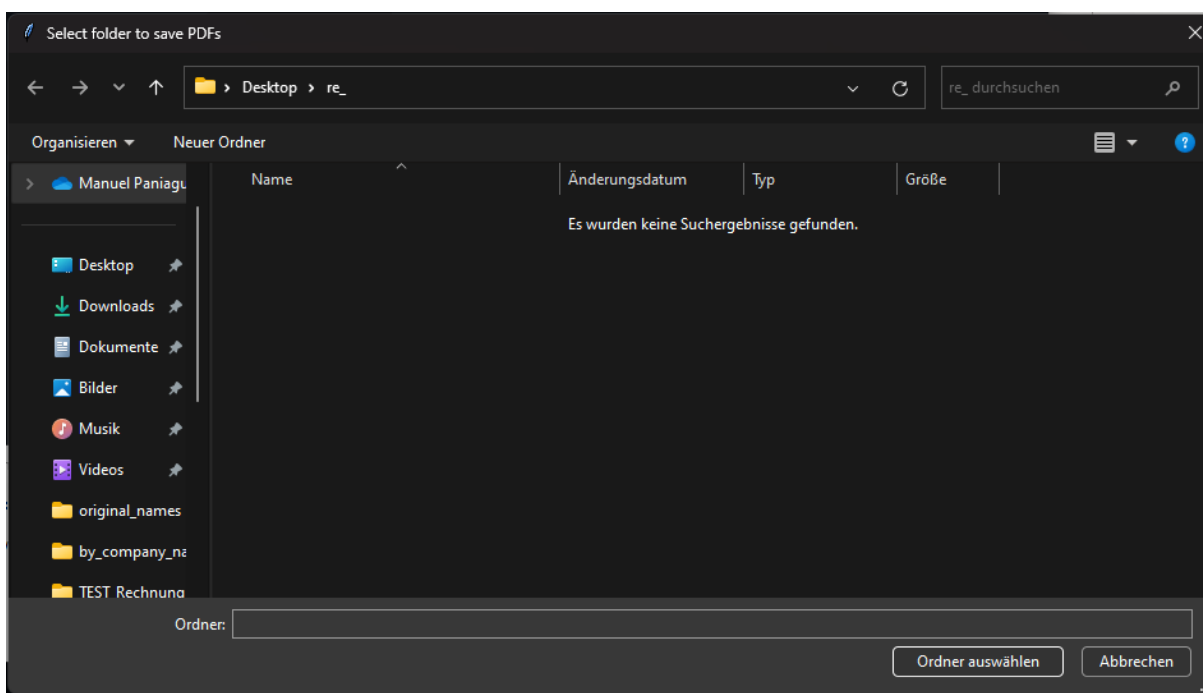


Abbildung 7: GUI Auswahl des Ordners

EMAIL VERWALTEN

Automatisierte E-Mail-Anhangs Verwaltung

Konfirmation Fenster:

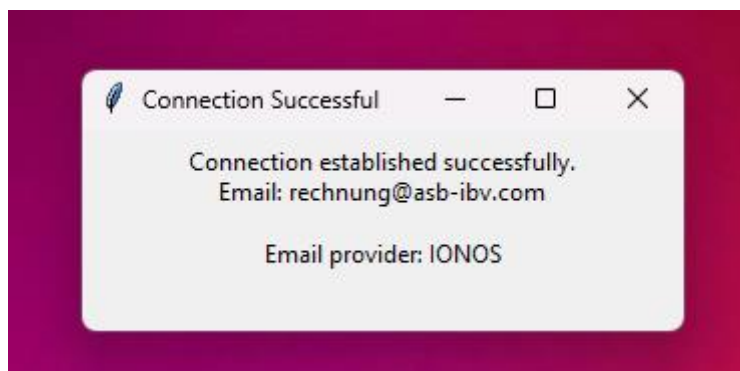


Abbildung 8: Konfirmation Fenster

LOGIC:

```
import tkinter as tk
from tkinter import filedialog, ttk

# Function to select the folder where PDFs will be saved
def select_folder():
    root = tk.Tk()
    root.withdraw() # Hide root window
    folder_selected = filedialog.askdirectory(title="Select folder to save PDFs")
    if not folder_selected:
        print("No folder selected. Exiting.")
        exit()

    return Path(folder_selected)

# Function to save user info (folder, email, and provider) to a JSON file
def save_user_info(folder_selected, email, provider):
    user_info = {
        "folder_selected": str(folder_selected), # Ensure folder path is stored as a string
        "email": email,
        "provider": provider
    }

    # Save the user info to a JSON file
    with open("user_info.json", "w") as f:
        json.dump(user_info, f, indent=4)

# Function to start checking inbox in a separate thread
def start_checking_inbox(mail, re_dir, json_file):
    try:
        while True:
            check_inbox(mail, re_dir, json_file)
            print("Waiting 30 seconds for the next check...")
            time.sleep(30)
    except KeyboardInterrupt:
        print("Exiting script.")
    finally:
        mail.logout()
```


5.3 Implementierung der Geschäftslogik

Die Geschäftslogik der Anwendung wurde so strukturiert, dass sie modular und leicht erweiterbar ist. Sie besteht aus verschiedenen Komponenten, die jeweils spezifische Aufgaben wie das Abrufen von E-Mails, das Verarbeiten von Anhängen und die Organisation von Dateien übernehmen. In diesem Abschnitt beschreiben wir das Vorgehen bei der Implementierung und heben interessante Funktionen hervor.

Vorgehensweise bei der Umsetzung

- **E-Mails abrufen und filtern:**
 - Mithilfe der IMAP-Schnittstelle werden E-Mails aus dem Posteingang basierend auf definierten Kriterien (z. B. UID) abgerufen.
 - Die Implementierung stellt sicher, dass nur neue E-Mails seit dem letzten verarbeiteten UID heruntergeladen werden.
- **Verarbeitung von E-Mails:**
 - Jede gefundene E-Mail wird analysiert, einschließlich Betreff, Absender und Datum.
 - Anhänge werden erkannt, dekodiert und in einem definierten Verzeichnis gespeichert.
- **Speicherung der Daten:**
 - Die Anhänge werden in entsprechenden Ordnern organisiert, basierend auf definierten Kriterien wie Typ oder Absender.

Interessante Funktionen und Algorithmen

1. Abrufen von E-Mails anhand der UID

Die Funktion `fetch_email_by_uid` lädt eine E-Mail basierend auf ihrer eindeutigen UID aus dem Posteingang. Diese Methode gewährleistet, dass gezielt eine bestimmte E-Mail abgerufen wird, was die Verarbeitung effizient macht.

```
def fetch_email_by_uid(mail, uid):  
    try:  
        mail.select("INBOX")  
        status, msg_data = mail.uid("FETCH", uid, "(RFC822)")  
        if status != "OK":  
            print(f"Error fetching email with UID {uid}")  
            return None  
  
        for response_part in msg_data:  
            if isinstance(response_part, tuple):  
                msg = email.message_from_bytes(response_part[1], policy=default)  
                return msg  
  
        return None  
    except Exception as e:  
        print(f"Error retrieving email by UID {uid}: {e}")  
        return None
```

EMAIL VERWALTEN

Automatisierte E-Mail-Anhangs Verwaltung

2. Anhänge speichern

Die Funktion `process_email` analysiert jede E-Mail und speichert deren Anhänge mithilfe einer Hilfsfunktion, die für die Organisation der Dateien zuständig ist.

```
def process_email(msg):
    """
    Processes an email message, including saving its attachments.

    Args:
        msg (EmailMessage): The email message to process.
    """
    if msg:
        print(f"Subject: {msg['subject']}")
        print(f"From: {msg['from']}")
        print(f>Date: {msg['date']}")

        # Process and save attachments
        for part in msg.iter_attachments():
            filename = part.get_filename()
            if filename:
                content = part.get_payload(decode=True)
                save_attachment_into_folder(content, filename)
    else:
        print("Email not found.")
```

3. Verarbeitung aller neuen E-Mails

Die Funktion `process_emails_since` durchsucht den Posteingang nach neuen E-Mails (UIDs größer als die zuletzt verarbeitete) und verarbeitet diese iterativ.

```
def process_emails_since(mail, last_uid):
    """
    Download and process all emails since the last UID.
    """
    try:
        # Search for emails with UIDs strictly greater than last_uid
        uids = search_emails(mail, f"UID {last_uid + 1}:*")
        if not uids:
            print("No new emails found.")
            return last_uid

        # Convert UIDs to integers and filter those greater than last_uid
        new_uids = [int(uid) for uid in uids if int(uid) > last_uid]

        if not new_uids:
            print("No new emails to process.")
            return last_uid

        for uid in new_uids:
            print(f"Processing email with UID {uid}...")
            msg = fetch_email_by_uid(mail, str(uid))
            process_email(msg)

        # Return the maximum UID processed
        return max(new_uids)
    except Exception as e:
        print(f"Error downloading emails: {e}")
        return last_uid
```

EMAIL VERWALTEN

Automatisierte E-Mail-Anhangs Verwaltung

Nach dem Herunterladen der Anhänge beginnt die Organisation der Dateien, die in mehreren Schritten erfolgt:

1. Verarbeitung der Anhänge

Die Funktion `process_attachments` übernimmt die Hauptlogik für die Verarbeitung der heruntergeladenen Anhänge. Hier werden Parameter aus einer *JSON*-Datei geladen, und die *PDF*-Dateien im Anhangs Ordner werden analysiert. Für jede Datei wird geprüft, ob relevante Informationen wie der Eigentümer (**eigentümer**) erkannt werden können. Wenn eine Zuordnung oder ein relevantes Schlüsselwort gefunden wird, wird die Datei entsprechend umbenannt und in einen Zielordner verschoben.

```
def process_attachments(attachment_folder, destination_folder, db_file):
    """
    Process attachments by loading parameters and handling PDFs.
    """
    print("Processing attachments...")
    parameters = load_parameters_from_db(db_file)

    if not parameters:
        print("No parameters loaded. Please check the JSON file.")
        return

    results = process_pdfs_in_folder(attachment_folder, parameters, destination_folder)

    for result in results:
        print(f"\nFile: {result['file_name']}")
        if result['match']:
            print(f"Matched Parameters: Eigentümer='{result['match']['eigentümer']}'")
            if result.get("prefix"):
                print(f"Prefix Applied: {result['prefix']}")
            if result.get("renamed_to"):
                print(f"Renamed To: {result['renamed_to']}")
            else:
                print("No match or relevant keyword found.")
```

EMAIL VERWALTEN

Automatisierte E-Mail-Anhangs Verwaltung

2. Verarbeitung der PDF-Dateien

Die Funktion `process_pdfs_in_folder` durchsucht den Anhangsordner nach PDF-Dateien. Jede Datei wird einzeln verarbeitet, um Text zu extrahieren, relevante Informationen zu finden, und sie bei Bedarf umzubenennen und zu verschieben.

```
def process_pdfs_in_folder(folder_path, parameters, destination_folder):  
    """ ...  
    >  
    results = []  
  
    try:  
        # Ensure the folder exists  
        if not os.path.exists(folder_path):  
            print(f"Folder does not exist: {folder_path}")  
            return results  
  
        # Ensure the destination folder exists  
        os.makedirs(destination_folder, exist_ok=True)  
  
        # Get all PDF files  
        pdf_files = [file_name for file_name in os.listdir(folder_path) if file_name.lower().endswith(".pdf")]  
  
        # Check if there are no PDF files  
        if not pdf_files:  
            print(f"No PDF files found in the folder: {folder_path}")  
            return results  
  
        # Process each PDF file  
        for file_name in pdf_files:  
            pdf_path = os.path.join(folder_path, file_name)  
            result = process_single_pdf(pdf_path, parameters, destination_folder)  
            results.append(result)  
  
    except Exception as e:  
        print(f"An error occurred while processing the folder: {e}")  
  
    return results
```

EMAIL VERWALTEN

Automatisierte E-Mail-Anhangs Verwaltung

3. Verarbeitung einzelner PDF-Dateien

Die Funktion `process_single_pdf` analysiert den Text einer einzelnen *PDF*-Datei, um nach Eigentümerdaten oder Schlüsselwörtern zu suchen. Sie verwendet eine unscharfe Suche (*fuzz matching*), um Übereinstimmungen im Text zu finden, und benennt die Datei entsprechend um.

```
def process_single_pdf(file_path, parameters, destination_folder, threshold=90):
    """ ...
    try:
        print(f"Processing file: {file_path}")

        # Extract text from the PDF
        text = extract_text_from_pdf(file_path)
        normalized_text = normalize_text(text)

        # Skip renaming if the file already starts with a valid prefix (case insensitive)
        file_name = os.path.basename(file_path)
        valid_prefixes = ("re_", "li_", "re_li_")
        if file_name.lower().startswith(valid_prefixes):
            print(f"File already starts with a valid prefix: {file_name}")
            return {"file_name": file_name, "match": None, "prefix": None}

        #TODO: create function for find_match in data_handler and use it here, using fuzz matching
        # Find matches for 'eigentümer' using fuzzy matching
        matched_entry = None
        for entry in parameters:
            eigentümer = normalize_text(entry.get("eigentümer", ""))
            if fuzz.partial_ratio(eigentümer, normalized_text) >= threshold:
                print(f"Match found for parameter '{entry['eigentümer']}'": {eigentümer}")
                matched_entry = entry
                break

        # Determine prefix based on keywords in the text
        prefix = determine_prefix(text)

        # Rename the file if either a match is found or a prefix is determined
        if prefix or matched_entry:
            new_name_parts = []

            # Add 'verw_nr' if a match is found
            if matched_entry:
                new_name_parts.append(matched_entry["verw_nr"])

            # Add the prefix if it exists
            if prefix:
                new_name_parts.append(prefix.rstrip("_"))

            # Add the original file name
            new_name_parts.append(file_name)

            # Construct the new name
            new_name = "_".join(new_name_parts)
            renamed_path = rename_attachment(file_path, new_name)
            print(f"File renamed to: {renamed_path}")

            # Move the file to the destination folder
            moved_path = move_attachment(renamed_path, destination_folder)
            print(f"File moved to: {moved_path}")

            return {
                "file_name": file_name,
                "match": matched_entry,
                "renamed_to": moved_path,
                "prefix": prefix,
            }
        else:
            print(f"No match or relevant keyword found for file: {file_name}")
            return {"file_name": file_name, "match": None, "prefix": None}

    except Exception as e:
        print(f"An error occurred while processing the file {file_path}: {e}")
        return {"file_name": file_name, "match": None, "error": str(e)}
```

EMAIL VERWALTEN

Automatisierte E-Mail-Anhangs Verwaltung

Nachdem die Anhänge heruntergeladen und verarbeitet wurden, wird die Organisation der Dateien durch eine zusätzliche Logik erweitert. Diese stellt sicher, dass die Dateien in **Unterordnern** strukturiert werden, **basierend auf** den Eigentümerdaten (**eigentuemer**) und der Verwaltungsnummer (**verw_nr**).

4. Organisation der Dateien nach Eigentümer

Die Funktion `organize_files_by_eigentuemer` übernimmt die Aufgabe, die bereits verarbeiteten Dateien in Unterordner zu verschieben. Dabei wird die `verw_nr` aus dem Dateinamen extrahiert und mit den Informationen aus der JSON-Datenbank (`db_file`) abgeglichen. Passende Dateien werden in Unterordner mit dem Namen des Eigentümers organisiert.

```
def organize_files_by_eigentuemer(processed_folder, db_file):
    """ ...

    print(f"Organizing files in folder: {processed_folder}")

    # Validate the processed folder
    if not is_valid_folder(processed_folder):
        print(f"Processed folder is not valid or does not exist: {processed_folder}")
        return

    # Load parameters from the database
    parameters = load_parameters_from_db(db_file)
    if not parameters:
        print("No parameters loaded. Please check the database file.")
        return

    # Get a list of all files in the folder
    files = [
        file_name
        for file_name in os.listdir(processed_folder)
        if os.path.isfile(os.path.join(processed_folder, file_name))
    ]

    for file_name in files:
        try:
            # Extract the verw_nr from the file name (if it exists)
            parts = file_name.split("_")
            if not parts or not parts[0].isdigit():
                print(f"Skipping file without verw_nr: {file_name}")
                continue

            verw_nr = parts[0]

            # Find the corresponding entry in the database
            matched_entry = next((entry for entry in parameters if entry.get("verw_nr") == verw_nr), None)
            if not matched_entry:
                print(f"No matching entry found for verw_nr {verw_nr}. Skipping file: {file_name}")
                continue

            eigentuemer = matched_entry.get("eigentuemer")
            if not eigentuemer:
                print(f"No 'eigentuemer' found for verw_nr {verw_nr}. Skipping file: {file_name}")
                continue

            # Create folder name as 'verw_nr_eigentuemer' with underscores instead of spaces
            eigentuemer_folder_name = f"{verw_nr}_{normalize_text(eigentuemer).replace(' ', '_')}"
            eigentuemer_folder_path = os.path.join(processed_folder, eigentuemer_folder_name)

            # Move the file to the eigentuemer folder
            src_path = os.path.join(processed_folder, file_name)
            moved_path = move_attachment(src_path, eigentuemer_folder_path)
            print(f"File {file_name} moved to folder: {eigentuemer_folder_path}")

        except Exception as e:
            print(f"An error occurred while processing file {file_name}: {e}")
```

EMAIL VERWALTEN

Automatisierte E-Mail-Anhangs Verwaltung

Integration in den Hauptprozess

Im Hauptprozess (`main`) wird diese Funktion nach der Verarbeitung der Anhänge aufgerufen, um die Dateien in die entsprechenden Unterordner zu verschieben. Dies stellt sicher, dass die Dateien nicht nur umbenannt, sondern auch übersichtlich organisiert werden.

```
def main():
    setup_configuration() # Ensure configuration is set up
    config = load_config()

    # Load paths from configuration
    attachment_folder = config.get("attachment_folder", "attachments/")
    destination_folder = os.path.join(attachment_folder, "Re_LI_Processed")

    # Ensure folders exist
    os.makedirs(attachment_folder, exist_ok=True)
    os.makedirs(destination_folder, exist_ok=True)

    db_file = resource_path("DB/db_objects.json")
    print(f"Database file path: {db_file}")
    if not os.path.exists(db_file):
        print(f"Database file not found: {db_file}")
        return

    while True:
        mail = get_imap_connection()

        # Track last processed UID
        last_uid = get_last_saved_uid()
        print(f"Last saved UID: {last_uid}")

        # Download emails and process attachments
        new_last_uid = process_emails_since(mail, last_uid)
        if new_last_uid > last_uid:
            save_last_uid(new_last_uid)

        # Process attachments
        process_attachments(attachment_folder, destination_folder, db_file)
        organize_files_by_eigentuemer(destination_folder, db_file)

        mail.logout()
        print("Waiting 3 min before the next run...")
        print("new_last_uid", new_last_uid)
        time.sleep(180)
```

Hier sind ein paar Screenshots zur Veranschaulichung:

EMAIL VERWALTEN

Automatisierte E-Mail-Anhangs Verwaltung

Angehängte Dateien werden hier heruntergeladen und gespeichert.

Zwischenablage	Organisieren	Neu
↑ > Dieser PC > Dokumente > TEST_RE	TEST_RE durchsuche	
Name	Änderungsdatum	Typ
Re_LI_Processed	09.12.2024 11:20	Dateiordner

Anschließend werden pdfs umbenannt und in Re_LI_Processed verschoben.

Schließlich werden sie auf der Grundlage von Eigentümern der Datenbank organisiert, wenn sie nicht draußen bleiben.

> Dieser PC > Dokumente > TEST_RE > Re_LI_Processed

Re_LI_Processed durchsuchen

ts

te

js

Name	Änderungsdatum	Typ	Größe
<div><div></div>102_dr_meiowitz</div>	09.12.2024 11:34	Dateiordner	
<div><div></div>220_fastighets_aktiebolaget_flodmynnin...</div>	09.12.2024 11:20	Dateiordner	
<div><div></div>555_tindur_beteiligungsgesellschaft_mbh...</div>	09.12.2024 11:20	Dateiordner	
<div><div></div>577_georg_caselli-bikafalvi_mathe</div>	09.12.2024 11:20	Dateiordner	
<div><div></div>633_hans-peter_wilhelm_fricke</div>	09.12.2024 11:20	Dateiordner	
<div><div><div></div></div>RE_MAXEDV.pdf</div>	09.12.2024 11:20	Adobe Acrobat-D...	301 KB

Beispielsweise befinden sich in /102_dr_meiowitz alle Rechnungen, die Ihnen entsprechen.

« TEST_RE » Re_LI_Processed » 102_dr_meiowitz

102_dr_meiowitz durch


Name	Änderungsdatum	Typ
 102_RE_hartner.pdf	09.12.2024 11:20	Adobe Acrobat-D...

Abbildung 9: Ablauf PDFs

Am Ende wurde das gesamte Projekt mithilfe von **PyInstaller** zu einer ausführbaren Anwendung verpackt. Der resultierende dist-Ordner wurde komprimiert, um die Weitergabe zu erleichtern. Gemeinsam mit einem Kollegen haben wir unter Verwendung der Software **AnyDesk** das Programm direkt auf dem PC des Kunden installiert und in Betrieb genommen.

6 Abnahmephase

Durchgeführte Tests

Für die Abnahmephase wurden umfangreiche Tests durchgeführt, um sicherzustellen, dass die Anwendung stabil und fehlerfrei funktioniert. Dabei standen verschiedene Testarten im Fokus:

- **Unit-Tests:**
Einzelne Funktionen und Methoden wurden isoliert getestet, um sicherzustellen, dass sie erwartungsgemäß arbeiten.
- **Integrationstests:**
Die Zusammenarbeit verschiedener Module, wie z. B. das Laden von Parametern und das Verarbeiten von Anhängen, wurde getestet.
- **Tests mit realen Daten:**
Ich hatte Zugriff auf ein Test-E-Mail-Konto sowie auf reale Rechnungen und Lieferscheine, die ich für die Tests verwendet habe. Dies stellte sicher, dass die Anwendung unter realen Bedingungen funktioniert.

Verwendung von Pytest

Ich habe mich für **Pytest** als Testframework entschieden, da es eine klarere und besser lesbare Konsolenausgabe bietet als das integrierte **unittest**-Modul von Python. Dies erleichtert die Fehlersuche erheblich und spart Zeit bei der Analyse von Testfehlern.

Bisher habe ich Unit-Tests für folgende Module erstellt:

- `test_data_handler.py`
- `test_file_handler.py`
- `test_email_handler.py`

Es sind jedoch noch weitere Tests für andere Funktionalitäten in Planung, um eine vollständige Abdeckung des Codes zu gewährleisten.

Ergebnisse der Tests

Die Tests haben mir geholfen, notwendige Änderungen im Code zu identifizieren, insbesondere in Bereichen, in denen unerwartete Datenformate oder Eingaben zu Fehlern führten.

Sie haben auch dazu beigetragen, die Fehlermeldungen klarer und verständlicher zu gestalten, damit diese im Falle eines Fehlers hilfreiche Informationen liefern.

Offizielle Abnahme

Die Anwendung wurde erfolgreich auf dem Kunden-PC installiert und getestet. Nach erfolgreicher Prüfung der Funktionalität mit realen Daten wurde die Anwendung offiziell abgenommen. Es wurden keine kritischen Fehler gemeldet, und der Kunde ist mit der Leistung und der Zuverlässigkeit der Anwendung zufrieden.

EMAIL VERWALTEN

Automatisierte E-Mail-Anhangs Verwaltung

Zusammenfassung

Die Kombination aus **Pytest**, realen Daten und klaren Teststrategien hat wesentlich dazu beigetragen, die Qualität der Anwendung zu gewährleisten und eine erfolgreiche Abnahme sicherzustellen. Weitere Tests werden weiterhin implementiert, um die Stabilität und Wartbarkeit der Software zu optimieren.

```
src > tests > processing > test_file_handler.py > ...
9
10 @patch("src.processing.file_handler.load_config")
11 @patch("src.processing.file_handler.os.makedirs")
12 @patch("src.processing.file_handler.open", new_callable=mock_open)
13 def test_save_attachment_into_folder(mock_open_func, mock_makedirs, mock_load_config, mock_config):
14     mock_load_config.return_value = mock_config
15     content = b"test content"
16     filename = "test_file.txt"
17
18     result = save_attachment_into_folder(content, filename)
19
20     # Normalize the expected paths
21     expected_folder = os.path.normpath("test_attachments")
22     expected_file_path = os.path.normpath(os.path.join("test_attachments", filename))
23
24     # Verify that the folder was created
25     mock_makedirs.assert_called_once_with(expected_folder, exist_ok=True)
26
27     # Verify that the file was written
28     mock_open_func.assert_called_once_with(expected_file_path, "wb")
29     mock_open_func().write.assert_called_once_with(content)
30
31     # Verify that the returned result is correct
32     assert os.path.normpath(result) == expected_file_path, f"Expected {expected_file_path}, got {result}"
33
```

```
PROBLEMS 183 OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER 67
Successfully installed colorama-0.4.6 iniconfig-2.0.0 pluggy-1.5.0 pytest-8.3.4
(.venv) PS C:\Users\MaxEDV\Documents\GitHub\automatisierte_email_Anhang> pytest .\src\tests\processing\test_file_handler.py
===== test session starts =====
platform win32 -- Python 3.11.5, pytest-8.3.4, pluggy-1.5.0
rootdir: C:\Users\MaxEDV\Documents\GitHub\automatisierte_email_Anhang
configfile: pytest.ini
collected 1 item

src\tests\processing\test_file_handler.py .

===== 1 passed in 0.27s =====
(.venv) PS C:\Users\MaxEDV\Documents\GitHub\automatisierte_email_Anhang>
```

7 Einführungsphase

Deployment der Anwendung

Das Deployment der Anwendung wurde mithilfe von **PyInstaller** durchgeführt. Dieser Prozess erforderte jedoch einige manuelle Anpassungen, da PyInstaller bestimmte Abhängigkeiten und Daten nicht automatisch erkannt hat.

Beim Erstellen der ausführbaren Datei mussten die `datas` und `hiddenimports` explizit angegeben werden, da PyInstaller diese nicht automatisch erkannt hat.

In `datas` wurde sichergestellt, dass wichtige Ressourcen wie die Datei `db_objects.json` und die Verzeichnisse (`imap`, `config`, `processing`, etc.) korrekt in die ausführbare Datei eingebunden wurden.

In `hiddenimports` wurden alle Module hinzugefügt, die PyInstaller nicht automatisch erkannt.

Probleme mit Importen:

Während der Entwicklung und des **Deployments** traten Probleme bei den Importen zwischen meinen Modulen auf. Insbesondere gab es Konflikte zwischen relativen und absoluten Importen, die angepasst werden mussten.

- Relativer Import wurde für Module innerhalb des gleichen Verzeichnisses bevorzugt.

EMAIL VERWALTEN

Automatisierte E-Mail-Anhangs Verwaltung

- Absoluter Import wurde verwendet, um Konflikte mit anderen Bibliotheken zu vermeiden.

```
a = Analysis(  
    ['src/main.py'],  
  
    pathex=['C:\\Users\\MaxEDV\\Documents\\GitHub\\automatisierte_email_Anhang'],  
    binaries=[],  
    datas=[  
        ('src/DB/db_objects.json', 'DB'),  
        ('src/imap', 'imap'),  
        ('src/config', 'config'),  
        ('src/processing', 'processing'),  
        ('src/emails', 'emails'),  
        ('src/utils', 'utils'),  
    ],  
    hiddenimports=[  
        'json',  
        'imaplib',  
        'processing.folders.folder_utils',  
        'emails.handler',  
        'pypdf',  
        'rapidfuzz',  
        'utils.file_utils',  
        'utils.resource_path'  
    ],  
)
```

Lösung für relative Pfade:

Um sicherzustellen, dass die Anwendung die Datei `db_objects.json` unabhängig vom Installationspfad findet, wurde ein `relative_path.py` erstellt. Dieses Skript garantiert, dass die Anwendung den Pfad zur JSON-Datei korrekt auflöst, unabhängig davon, ob sie in einer Entwicklungsumgebung oder als ausführbare Datei läuft.

Beispiel für `relative_path.py`:

```
import sys  
import os  
  
def resource_path(relative_path):  
    try:  
        # Temporary folder used by PyInstaller  
        base_path = sys._MEIPASS  
    except AttributeError:  
        # Current working directory in development mode  
        base_path = os.path.abspath(".")  
  
    return os.path.join(base_path, relative_path)
```

8 Dokumentation

Die Dokumentation der Anwendung erfolgt derzeit in einer einfachen und direkten Form:

README auf GitHub:

Eine ausführliche README-Datei auf GitHub dient als zentrale Anlaufstelle für alle wichtigen Informationen zur Installation, Verwendung und Wartung der Anwendung. Diese Datei richtet sich sowohl an Entwickler als auch an Administratoren und enthält:

- **Eine Übersicht der Funktionen**
- **Installationsanweisungen**
- **Nutzungshinweise**
- **Troubleshooting-Tipps**
Das README ist in Englisch verfasst, da die Kommunikation zwischen einigen Entwicklern in Englisch stattfindet und so für alle verständlich ist.
- **Kommunikation per E-Mail:**
Zusätzliche Anleitungen, Hinweise und Problemlösungen wurden bisher direkt per E-Mail an die zuständigen Personen kommuniziert. Dies betrifft insbesondere spezifische Fragen oder Konfigurationsanpassungen für die Anwendung.
- **Aktivitätsdiagramm:**
Zur besseren Darstellung des Projektablaufs wurde ein Aktivitätsdiagramm erstellt. Dieses ist ebenfalls in Englisch verfasst, um es für alle beteiligten Entwickler verständlich zu machen.

Zukünftige Pläne

Geplant ist, eine vollständige Dokumentation in Form eines **PDF-Dokuments** zu erstellen. Dies wird sicherstellen, dass alle relevanten Informationen an einem Ort gebündelt sind und leicht zugänglich bleiben.

9 Soll-/Ist-Vergleich

Das Hauptziel des Projekts wurde erreicht: die Verbindung zum *IMAP*-Server wurde hergestellt, und die heruntergeladenen Anhänge (Rechnungen) werden organisiert. Allerdings gab es während des Projekts einige Herausforderungen und Änderungen bei den Anforderungen, die den Verlauf beeinflusst haben.

Verlauf und Herausforderungen

Am Anfang war nicht genau definiert, wie die Anhänge organisiert werden sollten. Ich begann damit, Funktionen zu entwickeln, die Rechnungsnummern mit **Regex** suchen und alle zugehörigen Dateien in einer einzigen PDF-Datei zusammenführen. Diese Funktionen sind weiterhin in einer separaten Branch verfügbar. Das Problem war, dass diese Methode oft ungenau war und die Rechnungsnummer nicht zuverlässig gefunden hat.

Um das Problem zu lösen, habe ich eine **Google-Cloud-OCR-API** integriert, die nahezu perfekt funktionierte. Sie konnte in 99 % der Fälle die Rechnungsnummer erkennen und feststellen, ob die Anhänge zusammengehörten. Mit dieser Lösung konnte ich die Dateien erfolgreich zusammenführen. Allerdings stellte sich später heraus, dass diese API pro 1000 Seiten etwa 30 € kostet, was das Projektbudget überstieg. Da dies nicht mit dem Kunden abgesprochen war, mussten die Anforderungen angepasst werden.

Die neue Anforderung konzentrierte sich darauf, die Rechnungen den Eigentümern aus der Datenbank zuzuordnen, anstatt die Rechnungsnummer zu suchen oder Dateien zusammenzuführen. Dafür habe ich den Code angepasst und viele Teile aus der vorherigen Branch wiederverwendet.

Arbeiten mit Bildern

Während des Projekts wurde klar, dass auch Bilder verarbeitet werden müssen. In einer separaten Branch habe ich dafür **EasyOCR** getestet. Obwohl es funktionierte, benötigte es zu viel CPU-Leistung, was den Prozess verlangsamt und bei großen Datenmengen ungeeignet macht.

Das Hauptziel des Projekts wurde erreicht, da die Anwendung Rechnungen herunterlädt, organisiert und den jeweiligen Eigentümern zuordnet. Dennoch gibt es noch Verbesserungsbedarf, insbesondere

EMAIL VERWALTEN

Automatisierte E-Mail-Anhangs Verwaltung

bei der Verarbeitung von Bildern und der Gruppierung von Anhängen, die zu einer Rechnung gehören.

Ein großer Vorteil ist, dass der Code modular aufgebaut wurde, sodass neue Funktionen und Anforderungen problemlos hinzugefügt werden können. Damit ist das Programm gut für zukünftige Erweiterungen vorbereitet.

9.1 Lessons Learned

Ich habe viel über Python gelernt, vor allem über die Bedeutung, den Code gut zu strukturieren.

Ich habe verstanden, wie wichtig es ist, klare und schriftliche Anforderungen zu haben, um Missverständnisse zu vermeiden. Außerdem habe ich gelernt, dass es entscheidend ist, sich auf die eigentlichen Ziele des Projekts zu konzentrieren. Zum Beispiel habe ich auf eigene Initiative die Google-Cloud-OCR-API verwendet, was mir zwar geholfen hat, OCR zu verstehen, aber gleichzeitig Zeit gekostet hat, die ich in andere Bereiche des Projekts hätte investieren können. Ein großer Fehler war, dass ich die Preise der Google-Cloud-Dokumentation nicht richtig gelesen habe.

Ich habe erkannt, dass sich die Anforderungen des Kunden während des Projekts oft ändern können. Deshalb ist es wichtig, Funktionen so wiederverwendbar wie möglich zu schreiben und eine konstante Kommunikation mit der Person zu führen, die die Aufgaben zuteilt. Wenn ich zu lange brauche, um zu überprüfen, ob ich auf dem richtigen Weg bin, könnte ich unnötig arbeiten und am Ende neu anfangen müssen.

Ich habe auch gelernt, dass die Konfiguration oft länger dauert als erwartet, da immer kleine Probleme auftreten. Zum Beispiel war es notwendig, den `resource_path` anzupassen und die `datas` und `hiddenimports` bei der Verwendung von PyInstaller manuell einzufügen.

Ich habe erkannt, dass künstliche Intelligenz eine große Hilfe sein kann, aber es ist wichtig, über grundlegendes Wissen zu verfügen. Die KI versteht oft nicht den gesamten Kontext eines Projekts. Je besser ich weiß, was ich will und brauche, desto präziser kann ich die KI nach Hilfe fragen. Am Anfang des Projekts konnte ich die Antworten der KI oft nicht verstehen, da mir die Grundlagen in Python fehlten. Nachdem ich mehr Zeit investiert habe, Python zu verstehen, konnte ich die Unterstützung der KI besser nutzen.

Ich habe gelernt, zu akzeptieren, dass kleine Änderungen ein Projekt zeitweise blockieren können, weil sie nicht sofort funktionieren. Es kann frustrierend sein, den Fehler nicht zu finden, aber am Ende gibt es immer eine Lösung. Wenn ich an einem Punkt ankomme, an dem ich zu viel Zeit und Energie in ein Problem investiert habe, wechsle ich zu einer anderen Aufgabe, bevor ich mich zu sehr frustriere. Danach kehre ich mit neuer Energie zurück, um das Problem zu lösen.

9.2 Ausblick

Die nächsten Schritte für das Programm sind:

Wünsche, mitgeteilt am 06.12.12:

- **Wenn eine E-Mail mehrere Anhänge enthält, sollen diese zu einem PDF zusammengefasst werden.** Der Name des Dokuments soll dann wie folgt lauten: Objektnr_RG_Firma_Rechnungsnummer.
- **Wenn eine E-Mail mehrere Rechnungen enthält, dürfen diese nicht zu einem Dokument zusammengefasst werden.**

Die aktuelle Code-Struktur ermöglicht es, diese Änderungen umzusetzen, ohne größere Anpassungen vorzunehmen. Nach meinen Tests mit EasyOCR und der Google-API weiß ich, dass es möglich ist. Dafür muss ich jedoch meine Funktion `findRechnungsnummer()` anpassen. Das Programm muss lernen, in jedem Fall die Rechnungsnummer richtig zu erkennen. Das ist einfach, wenn die Nummer direkt neben der Bezeichnung „Rechnungsnummer“ steht. Es gibt jedoch Fälle, in denen die Nummer im extrahierten Text weit entfernt ist. Das Problem ist lösbar, ich benötige dafür nur etwas Zeit. Das wird auch die Anforderung lösen, die notwendigen Dateien zusammenzuführen. Ich habe den Code dafür, aber damit er richtig funktioniert, muss die Rechnungsnummer problemlos gefunden werden. Das Umbenennen der Dateien wird kein Problem darstellen.

EMAIL VERWALTEN

Automatisierte E-Mail-Anhangs Verwaltung

Nutzung einer API zur Kategorisierung von Dateien

Ein Kollege hat eine API, die Bilder kategorisiert. Sie akzeptiert zwei Parameter und liefert bei Übereinstimmung ein „True“ zurück. Diese API könnte ich eventuell im Projekt nutzen, um Bilder zu verwalten. Die API wurde von mir getestet und funktioniert gut.

Das Projekt befindet sich derzeit in der Testphase auf dem PC des Kunden. Es erfüllt bereits das Hauptziel, und wenn alles gut läuft, werden wir sicherlich daran weiterarbeiten, wie die gespeicherten Dokumente organisiert werden.

Literaturverzeichnis

Anon., 2024. *developer.hosting.ionos.de*. [Online]
Available at: <https://developer.hosting.ionos.de/docs/getstarted>
[Zugriff am 1 12 2024].

Anon., 2024. *drawio*. [Online]
Available at: <https://app.diagrams.net/>
[Zugriff am 10 12 2024].

Anon., 2024. *Maxedv Beratung GmbH*. [Online]
Available at: <https://maxedv-beratung.de/>
[Zugriff am 10 12 2024].

Anon., 2024. *Openai*. [Online]
Available at: <https://openai.com/chatgpt>
[Zugriff am 10 12 2024].

Anon., kein Datum *notion*. [Online]
Available at: <https://www.notion.so/>
[Zugriff am 1 1 2024].

Anon., kein Datum *w3schools*. [Online]
Available at: <https://www.w3schools.com/python/>
[Zugriff am 15 10 2024].

Cloud, G., 2024. *https://console.cloud.google.com/*. [Online]
Available at: <https://console.cloud.google.com/>
[Zugriff am 29 10 2024].

Macke, S., 2024. *dieperfekteprojektdokumentation.de*. [Online]
Available at: <https://dieperfekteprojektdokumentation.de/>
[Zugriff am 25 11 2024].

Eidesstattliche Erklärung

Ich, Ian Manuel Paniagua Porroa, versichere hiermit, dass ich meine Dokumentation zur betrieblichen Projektarbeit mit dem Thema

Email Verwalten – Automatisierte E-Mail-Anhangs Verwaltung

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

Hamburg, den 10.12.2024

IanPaniagua

IAN MANUEL PANIAGUA PORROA

EMAIL VERWALTEN

Automatisierte E-Mail-Anhangs Verwaltung