Software Developer (IHK)

Dokumentation zur Projektarbeit

DOPI - Document Organizer for PDF and Images

Anwendung zur Organisation von PDF- und Bilddateien

Abgabedatum: 26.11.2024

Prüfungsbewerber:

Artur Ikkert

Inhaltsverzeichnis

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

DOPI ..... *Document Organizer for PDF and Images*

exe ..... *executable*

GUI..... *Graphical User Interface*

ID ..... *Identification*

IHK..... *Industrie- und Handelskammer*

OCR..... *Optical Character Recognition*

PDF..... *Portable Document Format*

SQLite..... *Structured Query Language Lite*

1 Einleitung

Die folgende Projektdokumentation beschreibt den Ablauf des IHK-Abschlussprojektes, das im Rahmen des Zertifikatslehrgangs „Software Developer (IHK)“ durchzuführen ist. In dieser Dokumentation werden alle Phasen der Projektentwicklung erläutert, die die Planung, den Ablauf und die Durchführung beschreiben.

1.1 Projektbeschreibung

Es werden eine Vielzahl an Dokumenten auf unterschiedlichen Wegen empfangen, sei es per Post oder als E-Mail. Diese Dokumente werden derzeit in analogen oder digitalen Ordnern abgelegt. Die Suche nach benötigten Dokumenten kann sehr zeitaufwendig und frustrierend sein. Um diese Dokumente effizient zu organisieren und leicht wiederzufinden, wurde nach einer Lösung gesucht, die eine schnelle und einfache Archivierung ermöglicht.

1.2 Projektziel

Ziel ist eine speziell für diese Herausforderung entwickelte Anwendung zur digitalen Dokumentenverwaltung.

Die Anwendung soll es ermöglichen, Dokumente übersichtlich zu archivieren und schnell nach Inhalten zu durchsuchen. Zu den wichtigsten Funktionen der Anwendung gehören die Texterkennung mittels OCR, um Text aus Bilddateien zu extrahieren, sowie das direkte Auslesen von Inhalten aus PDF-Dokumenten. Die Daten werden in einer zentralen SQLite-Datenbank gespeichert, die eine strukturierte Ablage aller Dokumente und ihrer Metadaten ermöglicht. Eine benutzerfreundliche, strukturierte Oberfläche ermöglicht den schnellen Zugriff auf archivierte Dokumente und die gezielte Suche nach relevanten Informationen.

Die Anwendung bietet somit eine umfassende und einfach zu bedienende Lösung für das digitale Dokumentenmanagement. Perfekt für alle, die ihre digitalen Dokumente einfach, schnell und intuitiv organisieren und wiederfinden wollen.

1.3 Projektschnittstellen

Zur Verwaltung und Speicherung der Dokumentinformationen wird eine SQLite-Datenbank verwendet. In dieser Datenbank werden zu jedem Dokument die wichtigsten Informationen wie Dateiname, Datum, relevante Schlagworte und der ausgelesene Textinhalt gespeichert.

Mit Hilfe von SQL-Abfragen kann schnell und gezielt auf diese Daten zugegriffen werden. Dies erleichtert die Suche nach bestimmten Dokumenten oder Inhalten erheblich, da gezielt nach bestimmten Schlagworten oder anderen Kriterien gefiltert werden kann. Die Datenbank

sorgt somit dafür, dass alle wichtigen Informationen sicher und übersichtlich abgelegt sind und jederzeit schnell abgerufen werden können.

2 Projektplanung

Ziel der Projektplanung ist es, einen Überblick und Zeitplan für die Planung und Durchführung der einzelnen Projektphasen zu geben.

2.1 Projektphasen

Für die Durchführung wurde das Projekt in mehrere Phasen unterteilt, die den Entwicklungsprozess der Software abbilden. Für die Durchführung des Projektes stehen 64 bis 80 Stunden zur Verfügung, wobei 10 Stunden für die Planung und Konzeption, 30 Stunden für die Programmierung und 35 Stunden für die Erstellung der Dokumentation vorgesehen sind. Vor Projektbeginn wurde eine grobe Zeitplanung durchgeführt, um die zur Verfügung stehende Zeit auf die einzelnen Projektphasen aufzuteilen. Dieser grobe Zeitplan ist in Tabelle 1 dargestellt.

Projektphase	Geplante Zeit
Analyse	5 h
Entwurf	5 h
Implementierung	30 h
Dokumentation	35 h
Gesamt	**75 h**

Tabelle 1: Grobe Zeitplanung

2.2 Entwicklungsprozess

Zu Beginn der Entwicklung von DOPI stand die Wahl eines geeigneten Entwicklungsprozesses im Vordergrund. Die Wahl fiel auf das Wasserfallmodell mit Rückkopplung, ein lineares und sequenzielles Vorgehensmodell, das besonders geeignet ist, wenn die Anforderungen von Anfang an klar definiert sind und sich im Laufe des Projekts voraussichtlich nicht ändern werden[1]. Für DOPI ist es ideal, da die Hauptanforderungen und

[1] Vgl. (Handbuch der Softwareentwicklung, S. 178f)

die gewünschten Funktionen wie Texterkennung (OCR), PDF-Verarbeitung und eine benutzerfreundliche Oberfläche bereits feststanden.

Das Wasserfallmodell stellt einen linearen und strukturierten Entwicklungsansatz dar, der aus aufeinander folgenden Phasen besteht. Die Entwicklung von DOPI wird daher in klar definierten Schritten umgesetzt. Durch diesen sequenziellen Ablauf wird gewährleistet, dass jede Phase vollständig abgeschlossen ist, bevor die nächste Phase beginnt. Dies sorgt für eine hohe Nachvollziehbarkeit und strukturelle Klarheit in der Entwicklung.

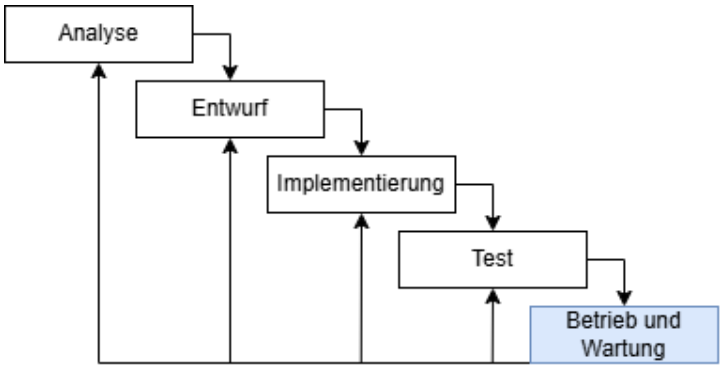


Abbildung 1: Wasserfallmodell mit Rückkopplung

Bei einem Projekt wie DOPI, das auf stabile Kernfunktionen ausgelegt ist, profitiert von dieser Vorgehensweise. Das Wasserfallmodell ermöglicht eine systematische Abarbeitung, ohne dass ständige Anpassungen erforderlich sind. Sollten dennoch Anpassungen notwendig werden, kann auf frühere Phasen zurückgegriffen und Änderungen integriert werden.

3 Analysephase

Nach Abschluss der Projektplanung kann die Analyse durchgeführt werden. Dabei wird der Ist-Zustand betrachtet und die Anforderungen definiert.

3.1 Ist-Analyse

Derzeit ist die Organisation der Dokumente eher unübersichtlich. Eingehende Briefe werden an den Kühlschrank gehängt, auf einen Ablagestapel oder in eine Dokumentenbox gelegt oder bestenfalls in einem Hängeregister aufbewahrt. Die Suche nach einem bestimmten Schreiben kann daher sehr zeitaufwendig und mühsam sein.

Ähnlich unorganisiert geht es bei E-Mails zu. Sie landen unsortiert in Ordnern auf dem PC und haben keine klare Struktur. Zudem gibt es kein Backup, was bedeutet, dass bei einem Datenverlust wichtige Dokumente unwiederbringlich verloren sind.

Insgesamt erschwert diese unsystematische Ablage oft das Auffinden bestimmter Dokumente. Es fehlt eine Lösung, die es ermöglicht, Dokumente digital und übersichtlich zu archivieren, so dass ein schneller Zugriff ohne langes Suchen möglich ist.

3.2 Anforderungsanalyse

Auf Grundlage der Ist-Analyse wurden die folgenden Hauptfunktionen definiert:

- **Texterkennung (OCR)**: Die automatische Erkennung und Extraktion von Text aus Bildern sollten möglich sein, um Inhalte zu erfassen und durchsuchbar zu machen.
- **PDF-Verarbeitung**: Die Anwendung sollte in der Lage sein, PDF-Dateien zu lesen, um Text aus diesen Dokumenten zu erfassen.
- **Dokumentenorganisation**: Die Benutzeroberfläche sollte eine einfache Navigation und Verwaltung der Dateien ermöglichen.
- **Suchfunktion**: Eine gezielte Suche nach Dokumenten und Inhalten soll möglich sein.
- **Archivierungsoption**: Die Dokumente müssen archiviert werden können. Ein Backup sollte so einfach wie möglich sein.

4 Entwurfsphase

Nach der Analysephase folgt die Entwurfsphase. Hier werden das grafische Design und die technische Umsetzung konzipiert.

4.1 Zielplattform

Die Anwendung soll als ausführbare Datei vom Dateityp .exe für Windows-Betriebssysteme erstellt werden. Die Verwendung auf anderen Betriebssystemen ist nicht vorgesehen, da die Anwendung für den privaten Gebrauch auf einem Windows-Gerät bestimmt ist.

Als Programmiersprache für die Entwicklung von DOPI wird Python gewählt, da es leistungsfähige Bibliotheken für die Bild- und Textverarbeitung bietet und Python derzeit meine bevorzugte Programmiersprache ist.

Für die Bildverarbeitung wird die Open Source OCR Engine der Universitätsbibliothek Mannheim[2] in Verbindung mit Pillow[2], einer Python Bildverarbeitungsbibliothek, verwendet. Um diese Engine in Python nutzen zu können, wird noch der zugehörige Wrapper

[2] S. Referenzverzeichnis

pyTesseract[2] benötigt, der die ursprüngliche Tesseract-Engine in eine Python-Bibliothek verpackt, um diese direkt im Programm nutzen zu können. Um die Texte aus den PDF-Dateien extrahieren zu können, wird pypdf[2] verwendet, eine Bibliothek zur Bearbeitung von PDF-Dateien in Python.

Die grafische Benutzeroberfläche wird mit Tkinter[2], einer in Python integrierten GUI-Bibliothek, und CustomTkinter[2], einer auf Tkinter basierenden GUI-Bibliothek, die neue, moderne und vollständig anpassbare Widgets bietet, realisiert.

Als Datenbank wird SQLite[2] verwendet, da diese direkt in die jeweiligen Anwendungen integriert werden kann und keine weitere Serversoftware benötigt. Für die Anbindung der Datenbank wird die Python-Bibliothek sqlite3[2] verwendet.

4.2 Architekturdesign

In der Entwurfsphase wurde entschieden, DOPI als monolithisches System[3] zu entwickeln, da es sich um eine kleine Anwendung handelt. Das bedeutet, dass alle Funktionen der Anwendung, wie Texterkennung (OCR), PDF-Verarbeitung, Datenbankverwaltung und Benutzeroberfläche in einem einzigen Anwendungssystem zusammengefasst sind.

Diese monolithische Architektur bietet folgende Vorteile für das Projekt[4,5]:

- **Einfache Verwaltung**: Da alle Funktionen in einem einzigen System ausgeführt werden, ist die Anwendung einfacher zu verwalten und erfordert keine komplexe Kommunikation zwischen verschiedenen Modulen oder Diensten.

- **Kürzere Entwicklungszeit**: Die Entwicklung und das Testen der Anwendung sind einfacher, da alle Funktionen in einer einzigen Umgebung implementiert und getestet werden können und der Aufwand für die Verwaltung mehrerer Module entfällt.

- **Bessere Performance für kleinere Anwendungen**: Direkte Funktionsaufrufe innerhalb eines einzelnen Prozesses können schneller sein als die Kommunikation zwischen den Modulen.

Die einzelnen Funktionen wie das Einlesen und Analysieren der Dokumente, die Speicherung der Daten in der SQLite-Datenbank und die Benutzeroberfläche werden in einem gemeinsamen Prozess ausgeführt. Dieser Ansatz ermöglicht es, das System

[2] S. Referenzverzeichnis
[3] Vgl. (Handbuch der Softwareentwicklung, S. 266f)
[4] Vgl. (Monolith vs Microservices: Everything You Need To Know)
[5] Vgl. (Discuss the pros and cons of using microservices architecture compared to a monolithic architecture)

übersichtlich zu gestalten und ohne komplexe Schnittstellen vollständig in einer Umgebung zu verwalten.

Neben den Vorteilen bringt die monolithische Architektur auch einige Nachteile mit sich:

- **Skalierbarkeit**: Um die Anwendung zu erweitern, muss die gesamte Anwendung erweitert werden, nicht nur einzelne Teile.

- **Hoher Änderungsaufwand**: Selbst kleine Anpassungen bedeuten oft, dass die gesamte Anwendung überarbeitet und neu bereitgestellt werden muss.

- **Unübersichtlicher Code**: Je größer das Programm wird, desto schwieriger wird es, den Code zu verstehen und zu warten.

- **Ausfallrisiko**: Der Ausfall einer einzelnen Komponente kann zum Ausfall des gesamten Systems führen.

Da das Programm nach seiner Fertigstellung weder geändert noch erweitert werden soll, überwiegen die Vorteile der monolithischen Architektur. Die einfache Verwaltung und die kürzere Entwicklungszeit machen das monolithische System zur idealen Wahl für dieses Projekt. Da keine zukünftigen Anpassungen geplant sind, fallen die Nachteile wie eingeschränkte Skalierbarkeit und hoher Änderungsaufwand kaum ins Gewicht.

4.3 Entwurf der Benutzeroberfläche

Die Benutzeroberfläche von DOPI ist einfach und übersichtlich gestaltet, um die Verwaltung und Suche nach Dokumenten so intuitiv wie möglich zu gestalten. Für die visuelle Gestaltung wurde ein modernes und schlichtes Design gewählt, das mit CustomTkinter realisiert wurde, da es den Fokus auf die wesentlichen Funktionen legt. Die Farbpalette ist bewusst einfach und dunkel gehalten, um Ablenkungen zu minimieren. Die Benutzeroberfläche verwendet hauptsächlich Grau- und Blautöne, die ein angenehmes visuelles Erscheinungsbild bieten.

Hauptansicht der Anwendung:

Die Hauptansicht bietet mehrere Eingabefelder für wichtige Informationen, die in der Datenbank gespeichert und durchsucht werden können. Der Dokumentname und der Inhalt der Datei werden nach dem Einlesen der Datei automatisch in die entsprechenden Textfelder eingefügt. Über einen segmentierten Button kann ausgewählt werden, ob eine PDF-Datei oder ein Bild importiert werden soll.

Entwurfsphase

- **Scannen-Button**: Dieser Button öffnet einen Dateiexplorer, mit dem eine Datei ausgewählt werden kann. Nach der Auswahl wird die Datei automatisch eingelesen und die relevanten Informationen werden in die entsprechenden Felder übertragen.

- **Speichern-Button**: Mit diesem Button werden die eingegebenen Informationen in der Datenbank gespeichert und die Datei im definierten Speicherpfad abgelegt, so dass die Datei sicher und schnell wieder auffindbar ist.

- **Pfad ändern-Button**: Über diesen Button kann der Speicherpfad festgelegt bzw. angepasst werden, um sicherzustellen, dass alle Dateien am gewünschten Speicherort abgelegt werden.

Suchansicht:

Im zweiten Fenster wird eine Tabelle angezeigt, die alle Dokumente aus der Datenbank enthält. Die Tabelle ist so gestaltet, dass die Dokumente übersichtlich dargestellt und gezielt durchsucht werden können.

- **Suchfeld und Suchen-Button**: Durch Eingabe von Suchbegriffen in das Textfeld und Anklicken des „Suchen“-Buttons kann eine SQL-Abfrage gestartet werden, die die Tabelle nach den gewünschten Inhalten durchsucht und die entsprechenden Ergebnisse anzeigt. So können Dokumente schnell gefunden und ausgewählte Informationen übersichtlich dargestellt werden.

Die gesamte Benutzeroberfläche von DOPI ist darauf ausgelegt, die Dokumentenverwaltung so einfach und effizient wie möglich zu gestalten, sodass Nutzer ihre Dateien problemlos speichern, durchsuchen und verwalten können.

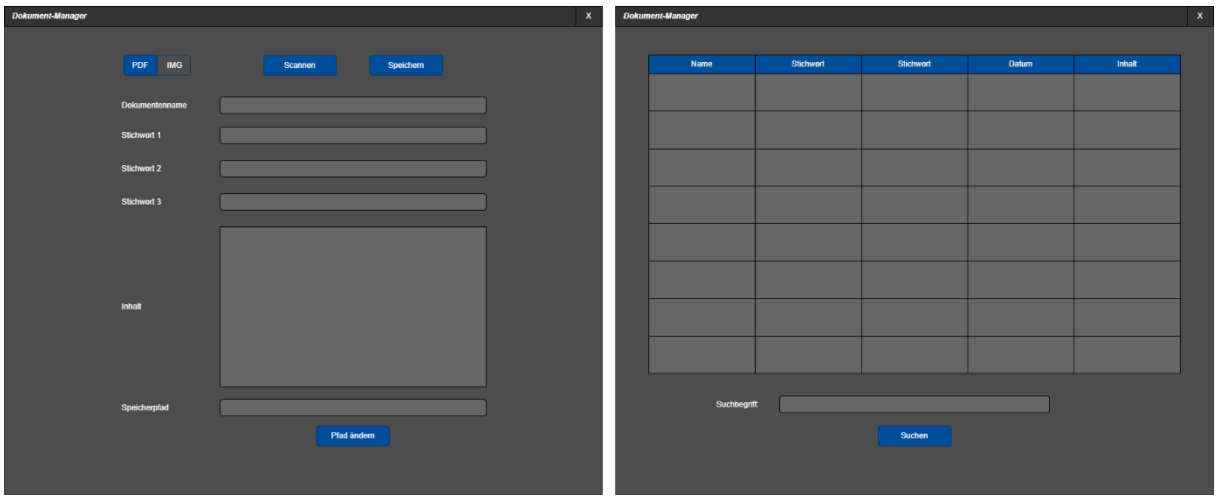


Abbildung 2: Erster Entwurf

4.4 Datenmodell

Das Datenmodell von DOPI bildet die Grundlage für die Speicherung und Verwaltung der Dokumente in der SQLite-Datenbank. Es stellt sicher, dass alle wichtigen Informationen zu jedem Dokument strukturiert abgelegt und später gezielt durchsucht werden können. Die Datenbank besteht aus einer Haupttabelle namens "documents", in der die verschiedenen Dokumenttypen (PDFs und Bilder) sowie deren Metadaten und Inhalte gespeichert werden.

Die Felder der Tabelle sind:

- **id** (INTEGER, PRIMARY KEY): Eindeutige ID für jedes Dokument. Sie dient als Primärschlüssel zur eindeutigen Identifizierung jedes Dokuments.
- **name** (TEXT, UNIQUE): Der Name des Dokuments, der beim Einlesen der Datei automatisch übernommen wird und eindeutig sein muss.
- **keyword1**, **keyword2** (TEXT): Zwei optionale Schlagwörter, die dem Dokument zur Kategorisierung oder für eine gezielte Suche zugeordnet werden können. Diese Felder unterstützen die Organisation und erleichtern die gezielte Suche.
- **date** (TEXT): Datum, an dem das Dokument erstellt oder in die Datenbank eingegeben wurde. Der Typ TEXT wird hier gewählt, wenn es kein eindeutiges Datum gibt und z.B. nur das Jahr gespeichert werden soll.
- **content** (TEXT): Der gesamte Textinhalt des Dokuments, der extrahiert wurde.

Da DOPI auf einer einfachen, monolithischen Architektur basiert, gibt es keine komplexen Beziehungen oder Verknüpfungen zwischen Tabellen. Alle Dokumente und ihre Metadaten werden direkt in der Tabelle documents gespeichert. Die Datenbank vermeidet Datenredundanz, indem jedes Dokument eine eindeutige ID und einen eindeutigen Namen erhält.

4.5 Maßnahmen zur Qualitätssicherung

Um sicherzustellen, dass DOPI zuverlässig funktioniert und die Anforderungen erfüllt, wurden verschiedene Qualitätssicherungsmaßnahmen definiert. Diese Maßnahmen stellen sicher, dass die Anwendung stabil, benutzerfreundlich und frei von kritischen Fehlern ist.

Alle Module und Funktionen werden nach der Implementierung sorgfältig getestet. Durch regelmäßige Code-Reviews, die ich selbst durchführe, werden mögliche Fehlerquellen und ineffiziente Strukturen frühzeitig erkannt und optimiert. Dies trägt zur Qualität und Übersichtlichkeit des Codes bei.

Zur Sicherstellung der Funktionalität werden Funktionstests durchgeführt. Dabei werden die zu importierenden Python-Bibliotheken bereits vor der Implementierungsphase in einem separaten Programm, das nur die Importfunktion enthält, getestet, um sicherzustellen, dass die Funktionen wie erwartet funktionieren und für DOPI verwendet werden können. Nach der Implementierung in das Hauptprogramm wird ein Unit-Test für die Texterkennung (OCR) erstellt. Der Unit-Test überprüft die korrekte Funktionalität. Der Test ist unter A5 zu finden. Zusätzlich werden die Kernfunktionen (Einlesen, Speichern, Suchen und Anzeigen von Dokumenten) aus der Sicht des Benutzers getestet. Damit wird sichergestellt, dass alle Anforderungen an die Anwendung erfüllt werden.

Um die Anwendung benutzerfreundlich zu gestalten, wird DOPI auf eine intuitive Bedienbarkeit hin getestet. Dadurch wird sichergestellt, dass alle Funktionen leicht verständlich und zugänglich sind. Rückmeldungen aus den Tests werden genutzt, um das Design und die Benutzerführung zu verbessern.

5 Implementierungsphase

In der Implementierungsphase wurde DOPI schrittweise auf der Grundlage der Ergebnisse der Entwurfsphase entwickelt. Dabei wurde auf eine Trennung von Datenstrukturen, Logik und Benutzeroberfläche geachtet, um eine stabile und funktionsfähige Anwendung zu erhalten. Jede Funktion wurde direkt nach der Implementierung getestet, um mögliche Fehler frühzeitig zu erkennen und zu beheben.

5.1 Implementierung der Datenstrukturen

Im ersten Schritt wurde die Datenbankstruktur erstellt. Eine SQLite-Datenbank wurde implementiert, um die Dokumentdaten sicher und effizient zu speichern. Die zentrale Tabelle documents wurde so angelegt, dass sie alle relevanten Informationen zu den Dokumenten enthält. Die Felder sind: id, name, keyword1, keyword2, date und content. Für die Anbindung der Datenbank wurde die SQLite-Bibliothek sqlite3 von Python verwendet. Es wurde eine Funktion implementiert, die die Datenbank automatisch erstellt, falls sie noch nicht existiert. Dadurch wird sichergestellt, dass die Anwendung immer mit einer gültigen Datenbankstruktur arbeitet.

5.2 Implementierung der Logik

Nach der Datenbankanbindung wurde mit der Implementierung der Programmlogik begonnen. Dabei wurde zunächst die Grundfunktionalität entwickelt, Daten in die Datenbank zu schreiben und wieder auszulesen. Anschließend wurde eine Funktion integriert, die es ermöglicht, die eingelesene Datei automatisch in den zuvor definierten Speicherpfad zu kopieren. Damit ist sichergestellt, dass die Dokumente zentral abgelegt und bei Bedarf direkt

wiedergefunden werden können. Dazu wird der gewählte Speicherpfad in einer JSON-Datei gespeichert. Beim Start der Anwendung wird diese Datei automatisch eingelesen und der Pfad in das entsprechende Textfeld der Benutzeroberfläche eingetragen. Dadurch bleibt der Speicherpfad auch nach einem Neustart der Anwendung erhalten und erleichtert dem Benutzer die Bedienung. Um diese Funktionen zu testen, wurden temporär einfache Texteingabefelder und Buttons in die Benutzeroberfläche integriert und mit den Funktionen verknüpft.

Die Texterkennung (OCR) wurde mit Hilfe der Bibliothek pytesseract integriert. Für die Verarbeitung von PDF-Dateien wurde die Bibliothek PyPDF2 verwendet, die den Textinhalt aus PDF-Dateien extrahiert. Da diese bereits in der Entwurfsphase getestet wurden, konnten sie problemlos integriert werden. Beide Funktionen sind so konfiguriert, dass sie nach dem Einlesen eines Dokuments automatisch den Namen der Datei und den Inhalt in die Textfelder Dokumentenname und Inhalt eintragen.

Zusätzlich wurde eine Suchfunktion implementiert, die mit Hilfe von SQL-Abfragen die Dokumente in der Datenbank durchsucht. Benutzer können nach Namen, Inhalt, Datum oder Schlagwörtern suchen.

5.3 Implementierung der Benutzeroberfläche

Die Benutzeroberfläche von DOPI wurde mit der Python-Bibliothek Tkinter und der Zusatzbibliothek CustomTkinter entwickelt. Ziel war es, eine einfache und übersichtliche Bedienung zu ermöglichen und alle Funktionen klar zugänglich zu machen.

Der Schwerpunkt lag auf der benutzerfreundlichen Interaktion mit den Funktionen, die zuvor in der Programmlogik implementiert wurden. Die Benutzeroberfläche wurde so entwickelt, dass sie die wichtigsten Aufgaben wie das Hinzufügen, Bearbeiten, Suchen und Verwalten von Dokumenten effizient unterstützt.

Während der Implementierungsphase wurde die Benutzeroberfläche kontinuierlich getestet und verbessert, um die Benutzerfreundlichkeit zu erhöhen. Das Design wurde bewusst einfach gehalten, um sicherzustellen, dass die Anwendung auch für Nutzer ohne technische Vorkenntnisse einfach zu bedienen ist.

Bei der Entwicklung wurde zunächst die Tabelle CTkTable[6] aus der Bibliothek CustomTkinter verwendet. Es stellte sich jedoch heraus, dass diese Tabelle nicht die für das Projekt benötigte Funktionalität und Anpassbarkeit bietet. Aufgrund dieser Einschränkungen wurde beschlossen, die CTkTable aus der Anwendung zu entfernen und durch die von Tkinter

[6] (pypi.org/project/CTkTable/)

bereitgestellte Treeview[7]-Tabelle zu ersetzen. Dieser Austausch ermöglichte die vollständige Integration der gewünschten Funktionen und die Erstellung einer Tabelle, die die Anforderungen des Projekts erfüllen kann.

5.4 Optimierungen während der Implementierungsphase

Trotz der sorgfältigen Analyse- und Entwurfsphase wurden während der Implementierung einige Verbesserungen vorgenommen, um die Funktionalität und Benutzerfreundlichkeit von DOPI weiter zu optimieren. Diese Anpassungen ergaben sich aus den Tests und der praktischen Umsetzung der geplanten Funktionen, die neue Erkenntnisse und Optimierungspotentiale aufzeigten.

Folgende Änderungen wurden während der Implementierungsphase vorgenommen:

- **Optimierung der Button-Anordnung:** Um die Übersichtlichkeit zu verbessern, wurden die Buttons in der Hauptansicht gruppiert und in einem Rahmen auf der linken Seite der Benutzeroberfläche platziert. Im zweiten Fenster wurden die Buttons in einer Zeile mit dem Suchfeld über der Tabelle angeordnet. Diese Änderungen erleichtert es, die wichtigsten Funktionen schnell zu finden und zu verwenden. Durch die klare Trennung der Buttons von den anderen Elementen wird die Benutzeroberfläche übersichtlicher und intuitiver.

- **Datenbankerstellung im Speicherpfad:** Die Datenbank wird nun direkt im definierten Speicherpfad erstellt. Beim ersten Start der Anwendung wird der Benutzer nach dem Speicherpfad gefragt, da bei jeder Änderung des Speicherpfades eine neue Datenbank angelegt wird, wenn noch keine vorhanden ist. Dadurch wird verhindert, dass Daten aus einer Datenbank gelesen werden, die sich nicht im aktuellen Speicherpfad befindet. Ein weiterer Vorteil ist, dass Dokumente in verschiedenen Ordnern abgelegt werden können, um eine noch strukturiertere Archivierung zu ermöglichen.

- **Segmentbutton Kopieren / Verschieben:** Um Redundanzen auf den Festplatten zu minimieren, ermöglicht die Verbesserung das direkte Verschieben der eingelesenen Datei. Dadurch entfällt das manuelle Löschen der Datei. Bei Bedarf kann die Datei weiterhin nur kopiert werden.

- **Button zum Leeren von Schlagwörtern und Datum:** Um die Bearbeitung der Metadaten zu erleichtern, wurde eine Funktion integriert, mit der die Schlagwörter

[7] (pythonassets.com/posts/treeview-in-tk-tkinter)

und das Datum in den entsprechenden Eingabefeldern geleert werden können. Dies erleichtert die Eingabe mehrerer Dokumente mit gleichen Metadaten, da diese nach dem Speichern nicht mehr automatisch geleert werden.

- **Option zum Öffnen und Löschen von Dateien:** Gespeicherte Dateien können direkt aus der Anwendung heraus geöffnet oder gelöscht werden.

- **Erweiterte und flexible Suchfunktion:** Die Abfrage wurde so angepasst, dass sie mehrere, durch Leerzeichen getrennte Suchbegriffe gleichzeitig verarbeiten kann. Außerdem werden die Ergebnisse nahezu in Echtzeit angezeigt.

- **Pop-up mit Suchbegriff-Hervorhebung:** Zur besseren Übersicht wurde ein Pop-up-Fenster hinzugefügt, das per Doppelklick den Inhalt eines Dokuments anzeigt und die Suchbegriffe farblich hervorhebt. Diese Funktion ermöglicht ein schnelles Erkennen der relevanten Inhalte, ohne das gesamte Dokument durchsuchen zu müssen.

- **Anzeige von Infotexten und Fehlermeldungen:** Infotexte informieren in der Hauptansicht über erfolgreiche Aktionen, wie z.B. das Speichern in der Datenbank, oder weisen auf Fehler hin. Dies macht die Anwendung transparenter und erleichtert die Fehlersuche.

Das finale Design des Programmes ist im Anhang A4 zu finden.

5.5 Testen der Anwendung

Nach jedem neu implementierten Teilbereich wurden umfangreiche Tests durchgeführt, um die Stabilität und Funktionalität der Anwendung sicherzustellen. Dabei wurden die einzelnen Module und Funktionen in realistischen Szenarien getestet, um sicherzustellen, dass sie den Anforderungen entsprechen und keine unerwarteten Fehler auftreten.

Der neu integrierte Programmcode wurde einem gründlichen Debugging unterzogen, um mögliche Fehlerquellen frühzeitig zu erkennen und zu beheben. Zur Unterstützung der Fehleranalyse wurden Debug-Meldungen und Print-Ausgaben eingefügt, um Variablen zu überprüfen und über die Konsolenausgabe Hinweise auf Probleme zu geben. Die Meldungen und Print-Ausgaben wurden später im finalen Quellcode wieder entfernt.

Methoden, bei denen Fehler auftreten können, wurden zur Vermeidung von Abstürzen in Try-Except-Blöcke eingebettet. Entsprechende Fehlermeldungen werden direkt in der Anwendung angezeigt. Zusätzlich wurde getestet, ob die Benutzeroberfläche alle

vorgesehenen Aktionen korrekt ausführt und ob die Interaktionen mit der Datenbank reibungslos funktionieren.

Die Testphase trug dazu bei, die Anwendung stabil und benutzerfreundlich zu gestalten, indem mögliche Schwachstellen identifiziert und behoben wurden.

6 Dokumentation

Neben dieser Projektdokumentation, die die Entwicklung und Umsetzung von DOPI beschreibt und gleichzeitig als Entwicklerdokumentation dient, wurde auch eine Benutzerdokumentation erstellt. Diese bietet eine verständliche Anleitung zur Bedienung der Anwendung und soll den Anwendern den Umgang mit DOPI so einfach wie möglich machen.

Die Dokumentation ist ein wichtiger Bestandteil des Projekts, da sie sowohl den Benutzern als auch den Entwicklern hilft, die Funktionalität und den Aufbau der Software zu verstehen. Während die Projektdokumentation technische Details, den Entwicklungsprozess und die Programmlogik abdeckt, konzentriert sich die Benutzerdokumentation darauf, die Bedienung der Software zu erklären.

Die Benutzerdokumentation ist im Anhang unter A6 zu finden und stellt eine klare Orientierungshilfe für alle dar, die mit DOPI arbeiten wollen.

7 Fazit

Die Entwicklung von DOPI war ein erfolgreicher Prozess, der sowohl die definierten Anforderungen erfüllte als auch neue Erkenntnisse und Erfahrungen lieferte. Mit DOPI wurde eine Anwendung geschaffen, die eine effiziente und benutzerfreundliche Lösung für das digitale Dokumentenmanagement bietet. Die Kernfunktionen wie Texterkennung, PDF-Verarbeitung, Such- und Filteroptionen sowie die übersichtliche Benutzeroberfläche wurden vollständig umgesetzt und ermöglichen eine einfache und intuitive Bedienung.

7.1 Soll-/Ist-Vergleich

Der Soll-Ist-Vergleich zeigt, wie die ursprünglichen Zielsetzungen des Projekts im Vergleich zu den tatsächlich erreichten Ergebnissen umgesetzt wurden. In zeitlicher Hinsicht wurde die Entwicklung weitgehend wie geplant durchgeführt, obwohl einige Arbeitsschritte mehr Zeit, als erwartet in Anspruch nahmen. Diese Verzögerungen konnten jedoch durch einen eingeplanten Puffer aufgefangen werden, so dass der Zeitplan insgesamt eingehalten werden konnte.

Wie in Tabelle 2 zu erkennen ist, konnte die Zeitplanung mit geringen Abweichungen eingehalten werden.

Phase	Geplant	Tatsächlich	Differenz
Analyse	5 h	3 h	-2 h
Entwurf	5 h	9 h	+4 h
Implementierung	30 h	29 h	-1 h
Dokumentation	35 h	36 h	+1 h
Gesamt	**75 h**	**77 h**	

Tabelle 2: Soll-/Ist-Vergleich

Trotz geringfügiger Abweichungen in einzelnen Phasen konnte DOPI erfolgreich entwickelt werden und übertraf in einigen Bereichen die ursprünglichen Anforderungen.

7.2 Lessons Learned

Durch die Entwicklung von DOPI konnten wertvolle Erkenntnisse und Erfahrungen gewonnen werden, die sowohl für zukünftige Projekte als auch für die persönliche Weiterentwicklung hilfreich sind.

Eine der wichtigsten Erkenntnisse war die Bedeutung einer soliden Planung in der Analyse- und Entwurfsphase. Obwohl die meisten Anforderungen klar definiert waren, zeigten sich während der Implementierung zusätzliche Optimierungspotentiale, die in der Planungsphase nicht berücksichtigt worden waren. Dies unterstreicht die Wichtigkeit, genügend Zeit für flexible Anpassungen einzuplanen.

Eine klare Trennung von Datenstrukturen, Logik und Benutzeroberfläche in Microservices wäre für die Übersichtlichkeit in diesem Projekt wahrscheinlich von Vorteil gewesen. Auch wenn es mir am Anfang wie ein kleines Programm vorkam, wurden doch mehr Codezeilen benötigt als erwartet, was die Suche nach der richtigen Codezeile oder Funktion am Ende doch erschwerte.

Darüber hinaus hat die Testphase gezeigt, wie wichtig eine frühzeitige und regelmäßige Fehleranalyse ist. Durch gezielte Tests konnten Probleme frühzeitig erkannt und behoben werden, bevor sie größere Auswirkungen hatten. Da die Tabellenerstellung mehr Probleme bereitete als erwartet, werde ich bei zukünftigen Projekten auch diese Module im Vorfeld auf ihre Eignung prüfen, um mögliche Einschränkungen oder Probleme frühzeitig zu erkennen und Alternativen in Betracht ziehen zu können.

Insgesamt hat die Entwicklung von DOPI gezeigt, dass ein flexibles und gut strukturiertes Vorgehen entscheidend für den Erfolg eines Projekts ist. Die gesammelten Erfahrungen werden in zukünftigen Projekten als wertvolle Orientierung dienen.

7.3 Ausblick

DOPI erfüllt bereits die geplanten Anforderungen an ein effizientes und benutzerfreundliches Dokumentenmanagement. Als mögliches Entwicklungsziel wäre die Integration von Cloud-Diensten ein sinnvoller Schritt, um Dokumente nicht nur lokal, sondern auch online verfügbar zu machen. Dadurch könnte die Anwendung ortsunabhängig genutzt werden, was insbesondere dann interessant ist, wenn regelmäßig von verschiedenen Geräten auf die Dokumente zugegriffen werden soll. Bei der Einbindung von Cloud-Diensten wäre auch eine Android-App denkbar, die auf Basis des Programms entwickelt werden könnte, um Dokumente aus E-Mails schnell mit dem Programm zu archivieren.

Referenzverzeichnis

Akascape. (14. 11 2024). *pypi.org/project/CTkTable/.* Von https://pypi.org/project/CTkTable/ abgerufen

Clark, J. A. (15. 10 2024). *pypi.org*. Von https://pypi.org/project/pillow/ abgerufen

Codemy.com. (kein Datum). *Python GUI's With TKinter*. Von https://www.youtube.com/playlist?list=PLCC34OHNcOtoC6GglhF3ncJ5rLwQrLGnV abgerufen

Discuss the pros and cons of using microservices architecture compared to a monolithic architecture. (Juni 2024). Von https://learn.redhat.com/t5/General/Discuss-the-pros-and-cons-of-using-microservices-architecture/td-p/43569 abgerufen

docs.python.org. (14. 11 2024). Von https://docs.python.org/3/library/tkinter.html abgerufen

docs.python.org. (14. 11 2024). Von https://docs.python.org/3/library/sqlite3.html abgerufen

Fenniak, M. (14. 11 2024). *pypi.org*. Von https://pypi.org/project/pypdf/ abgerufen

Hoffstaetter, S. (14. 11 2024). *pypi.org*. Von https://pypi.org/project/pytesseract/ abgerufen

Krypczyk V., Bochkor E. (2022). *Handbuch der Softwareentwicklung* (ISBN 978-3-8362-7977-2 Ausg.). Rheinwerk Verlag.

pythonassets.com/posts/treeview-in-tk-tkinter. (14. 11 2024). Von https://pythonassets.com/posts/treeview-in-tk-tkinter/ abgerufen

Schimansky, T. (14. 11 2024). *customtkinter.tomschimansky.com*. Von https://customtkinter.tomschimansky.com/ abgerufen

Sitnikova, A. (7. September 2021). *Monolith vs Microservices: Everything You Need To Know*. Von https://bambooagile.eu/insights/monolith-vs-microservices abgerufen

sqlite.org. (14. 11 2024). Von https://www.sqlite.org/ abgerufen

Tkinter. com. (kein Datum). *Modern Tkinter Design With CustomTkinter*. Von https://www.youtube.com/playlist?list=PLfZw_tZWahjxJl81b1S-vYQwHs_9ZT77f abgerufen

TurbineThree. (19. 09 2023). *Make Tkinter Look 10x Better in 5 Minutes (CustomTkinter)*. Von https://www.youtube.com/watch?v=Miydkti_QVE abgerufen

UB-Mannheim. (11 2024). *"github.com"*. Von Tesseract Open Source OCR Engine (main repository): https://github.com/UB-Mannheim/tesseract abgerufen

Anhang

A1 Detaillierte Zeitplanung

Analysephase		**3 h**
1. Analyse des Ist-Zustands	0,5 h	
2. Anforderungsanalyse	1,5 h	
3. Erstellen eines Use-Case-Diagramms	1 h	
Entwurfsphase		**9 h**
1. Recherche der benötigten Bibliotheken	1 h	
2. Auswahl des Architektur Design	1 h	
3. Entwurf der Benutzeroberfläche	3 h	
4. Datenbankentwurf	1 h	
5. Testen der zu importierenden Python-Bibliotheken	3 h	
Implementierungsphase		**29 h**
1. Anlegen der Datenbank	1 h	
2. Implementierung der Logik	6 h	
2.1. Datenbankanbindung	0,5 h	
2.2. Funktion zum Kopieren der Datei	1,5 h	
2.3. Speicherpfad in JSON-Datei ablegen und einlesen	1 h	
2.4. Import der Bibliotheken	1 h	
2.5. Suchfunktion erstellen	2 h	
3. Implementierung der Benutzeroberfläche	9,5 h	
3.1. GUI erstellen	2 h	
3.2. CTKtable erstellen	3 h	
3.3. Füllen der Tabelle	2 h	
3.4. Wechsel auf Treeview Tabelle	2,5 h	
4. Optimierungen und Änderungen	12,5 h	
4.1. Neue Buttons und Funktion anpassen	1 h	

4.2. Button-Anordnung ändern	0,5 h	
4.3. Datenbank im Speicherpfad erstellen lassen	0,5 h	
4.4. Suche in Echtzeit-Suche ändern	0,5 h	
4.5. Pop-up-Fenster mit Highlight Funktion	3 h	
4.6. Fehler beheben, Try-Except und Messageboxen einbetten	5 h	
4.7. Anzeige von Infotexten und Fehlermeldungen	2 h	
Erstellen der Dokumentation		**36 h**
1. Erstellen der Benutzerdokumentation	4 h	
2. Erstellen der Projektdokumentation	32 h	
Gesamt		**77 h**

Tabelle 3: Detaillierte Zeitplanung

A2 Use-Case-Diagramm

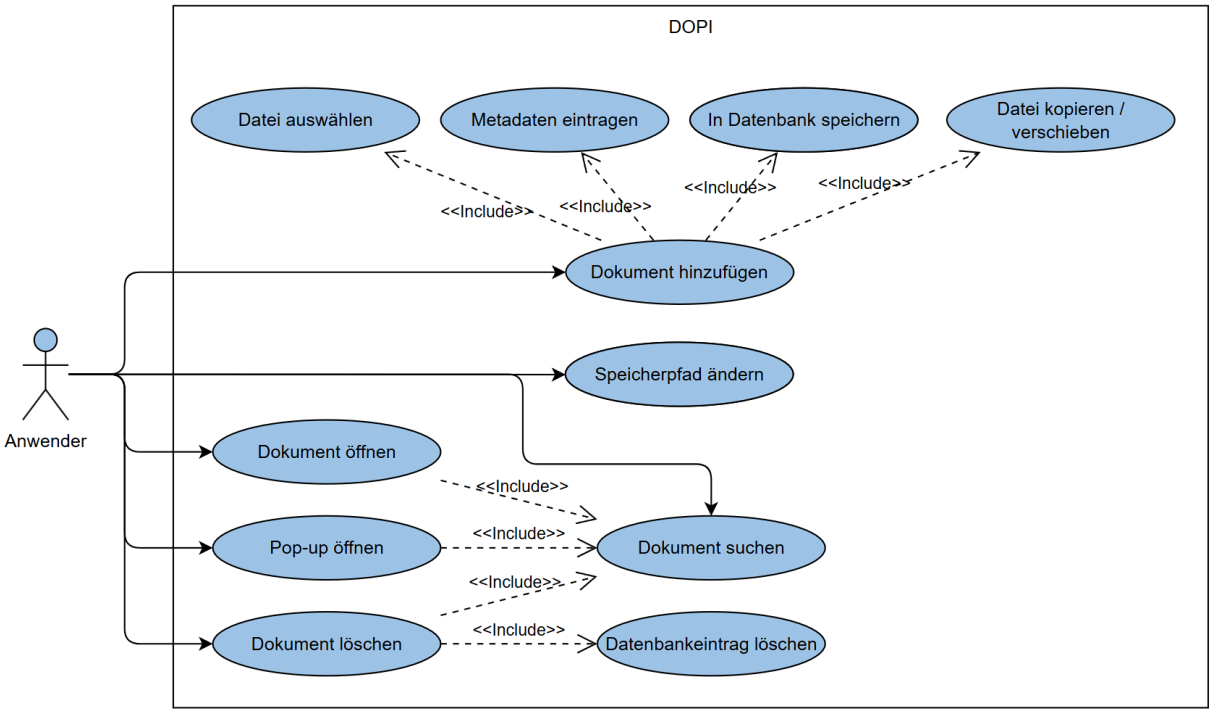


Abbildung 3: Use-Case-Diagramm

A3 Programmablaufplan

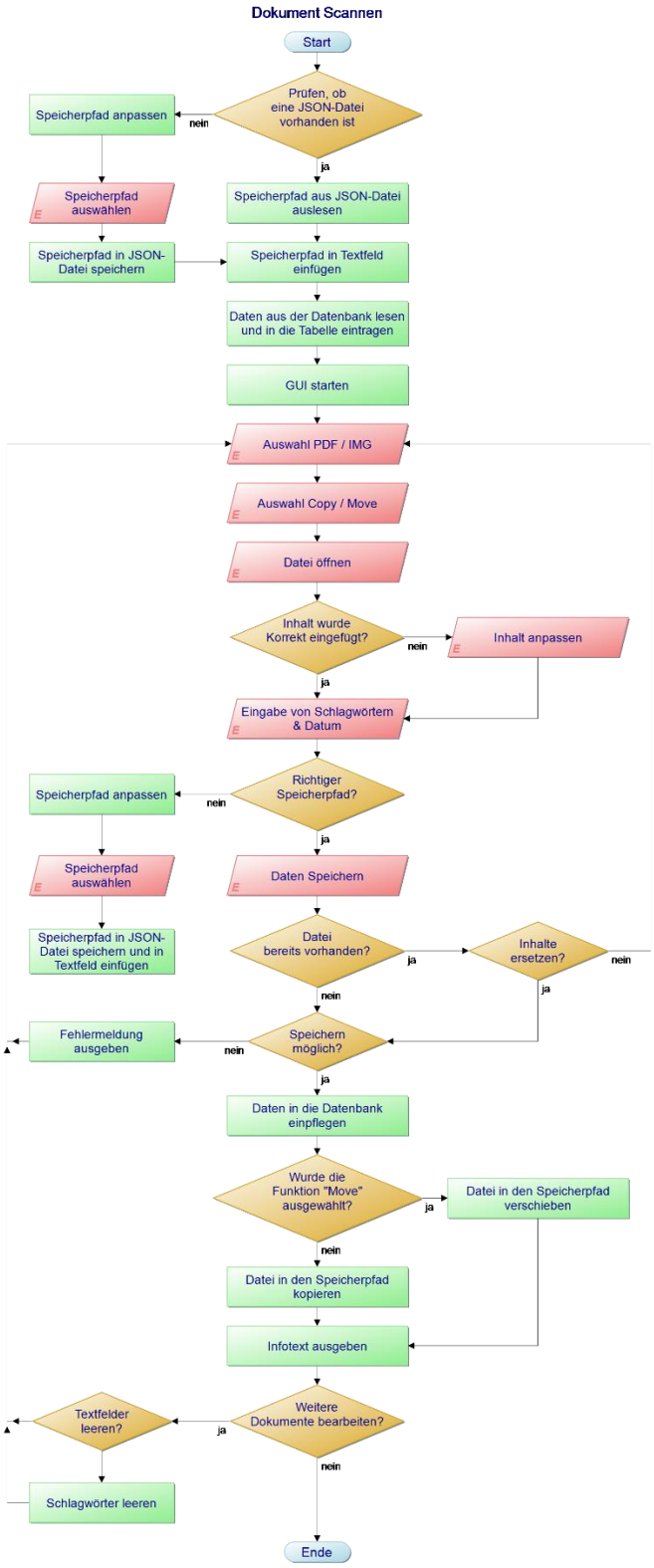


Abbildung 4: Programmablaufplan "Dokument Scannen"

Anhang

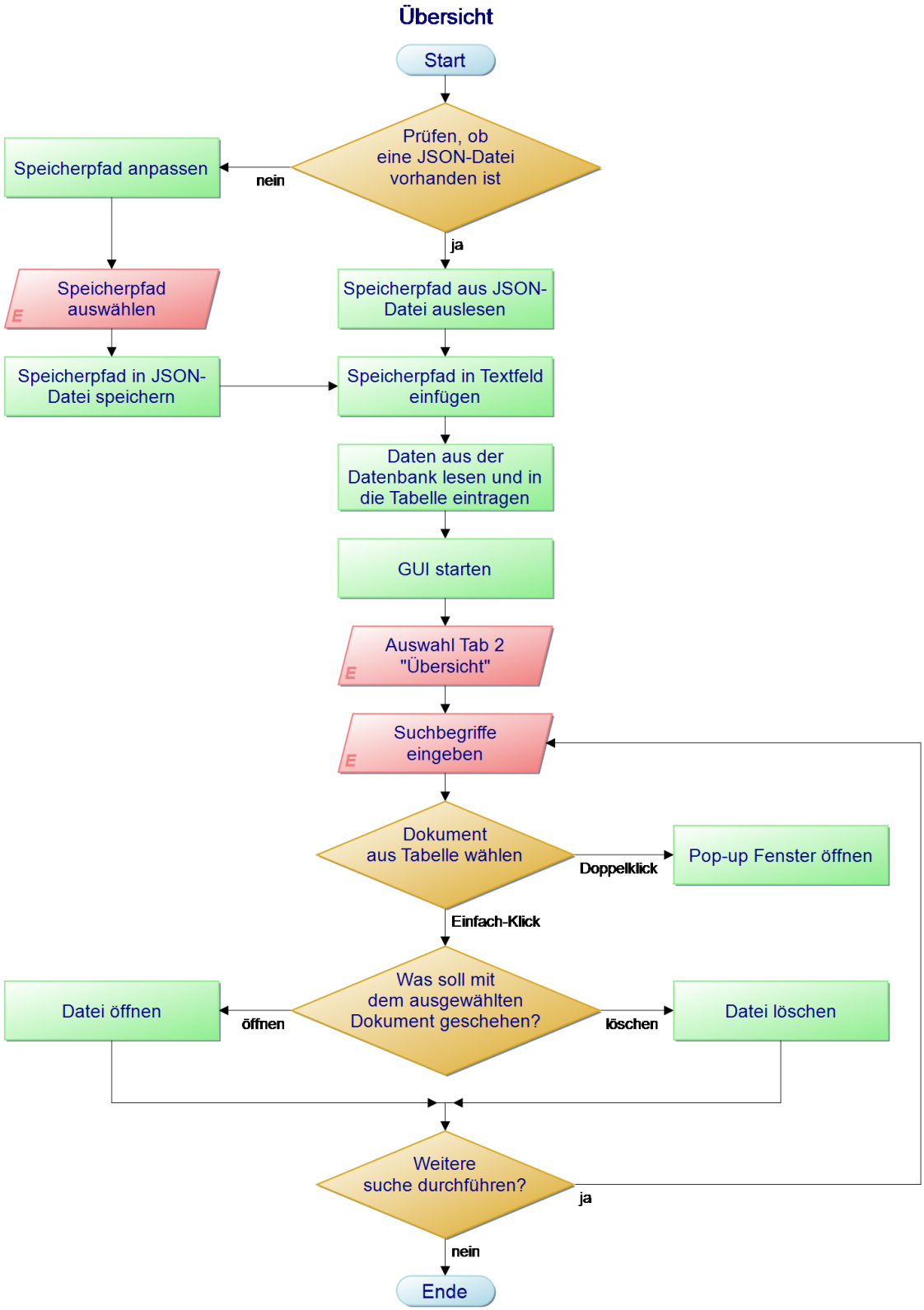


Abbildung 5: Programmablaufplan "Übersicht"

A4 Screenshots der Anwendung

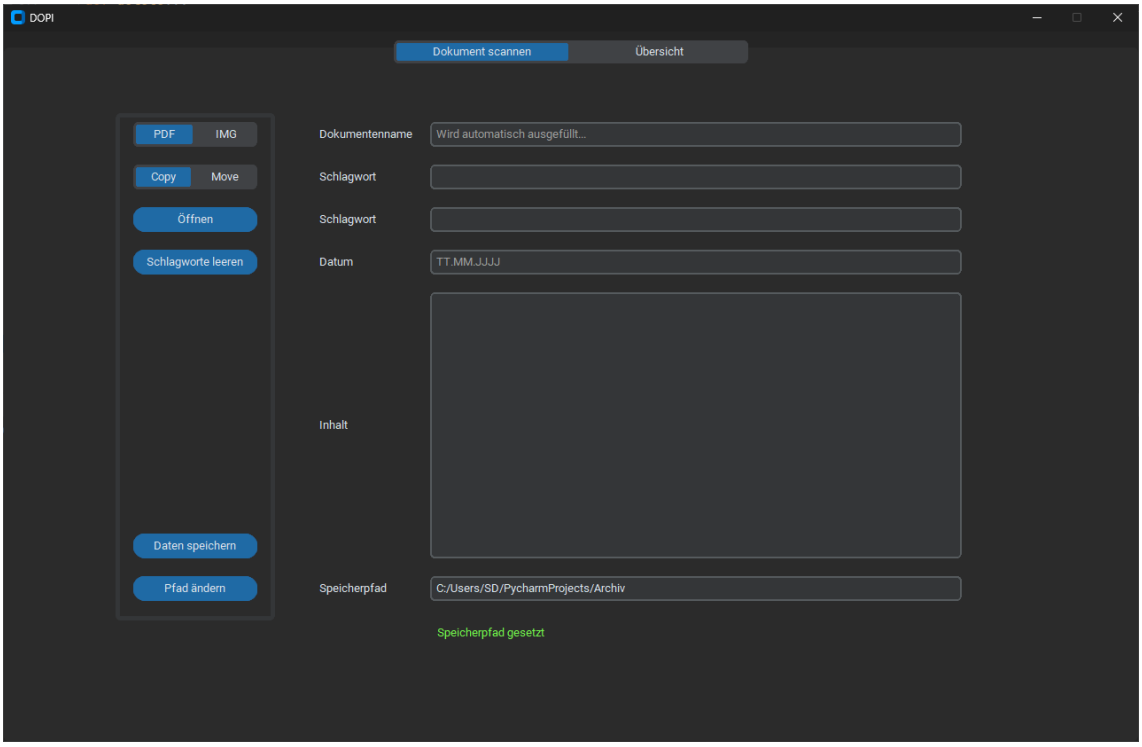


Abbildung 6: Finales Design "Dokument scannen"

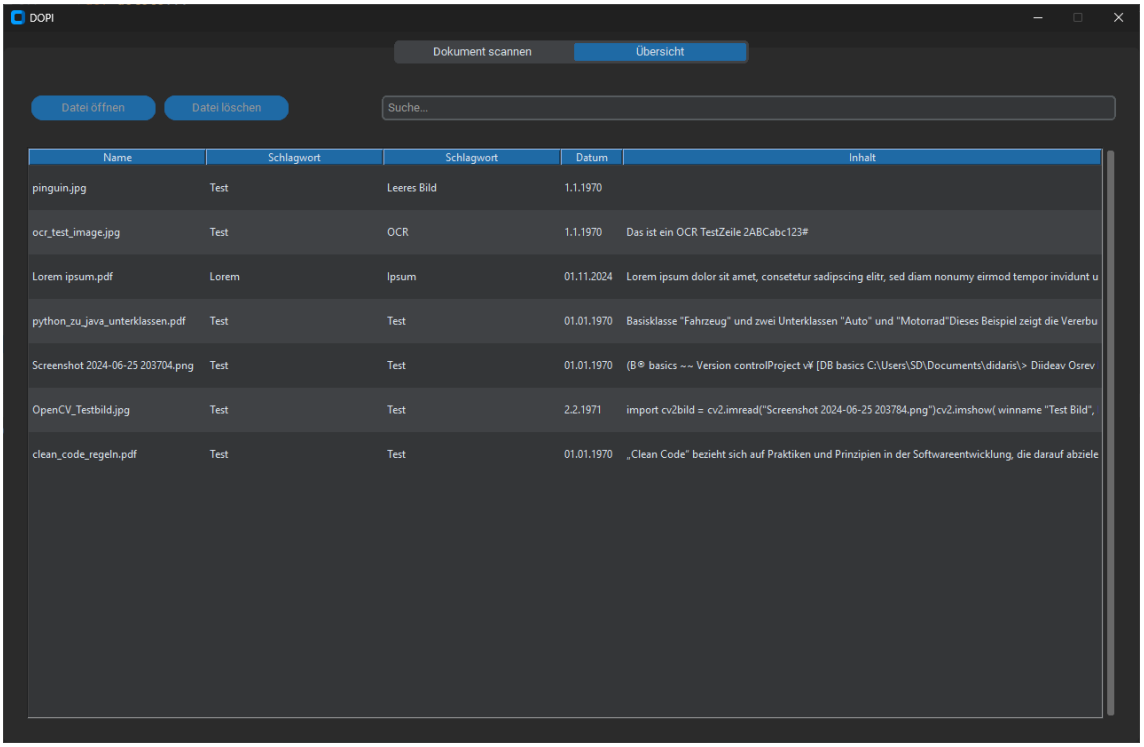


Abbildung 7: Finales Design "Übersicht"

A5 Pytest

```python
# test_ocr.py
from DOPI import open_image
image_path = "ocr_test_image.jpg"
image_path2 = "pinguin.jpg"
def test_read_ocr():
    assert open_image(image_path) == "Das ist ein OCR Test\n\nZeile 2\n\nABCabc123#\n"
    assert open_image(image_path2) == ""
```

Abbildung 8: Code für den pytest

```
(.venv) PS C:\Users\SD\PycharmProjects\DOPI> pytest .\test_ocr.py
============================================================================ test session
starts =============================================================================
platform win32 -- Python 3.11.8, pytest-8.3.3, pluggy-1.5.0
rootdir: C:\Users\SD\PycharmProjects\DOPI
collected 1 item

test_ocr.py .                                                                [100%]

============================================================================ 1 passed in
2.44s ==============================================================================
```

Abbildung 9: pytest im Terminal

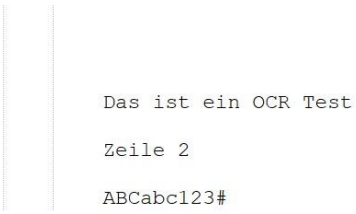


Abbildung 10: Testbild 1 für den pytest

Abbildung 11: Testbild 2 für den pytest

A6 Benutzerdokumentation (Auszug)

1 Einleitung

Dieses Benutzerhandbuch bietet eine Anleitung zur Nutzung von DOPI, einer Software, die Ihnen hilft, Ihre PDFs und Bilder digital zu organisieren. Mit DOPI können Sie Ihre Dateien ganz einfach speichern, durchsuchen und verwalten.

2 Erste Schritte

DOPI muss nicht installiert werden. Legen Sie die DOPI.exe in einem Ordner ab, in dem auch die Config-Datei erstellt werden soll. Starten Sie DOPI mit einem Doppelklick auf DOPI.exe.

Wenn Sie das Programm zum ersten Mal starten, werden Sie gefragt, wo Sie Ihre Dokumente speichern möchten. Wählen Sie einen Ordner auf Ihrem Computer.

3 Dokument hinzufügen

Wählen Sie bei ❶ aus, um welche Art von Dokument es sich handelt.

Wählen Sie bei ❷, ob Sie die Datei kopieren oder verschieben, möchten.

Klicken Sie auf „Öffnen“❸, um eine Datei auszuwählen. Der Dateiname und der Inhalt werden automatisch in die Textfelder eingetragen. Falls der Inhalt nicht korrekt eingelesen wurde, kann er vor dem Speichern bearbeitet werden. Der Name kann nicht geändert werden, da die Datei sonst nicht aus dem Programm heraus geöffnet werden kann.

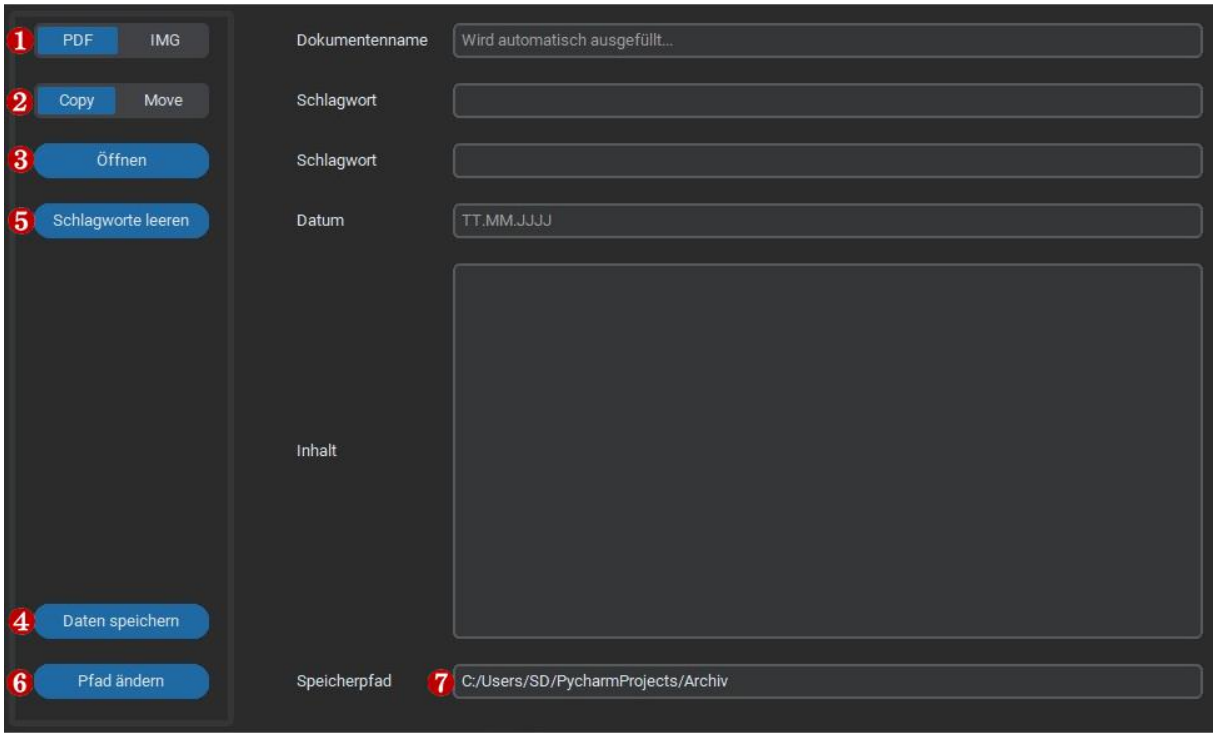
Sie können noch ein Datum und Schlagworte eingeben, um später danach suchen zu können.

Klicken Sie auf „Daten speichern“❹, um die Daten in der Datenbank zu speichern und die Datei zu kopieren/zu verschieben.

Die Textfelder werden nach dem Speichern nicht automatisch geleert, falls Sie mehrere Dokumente mit dem gleichen Datum oder den gleichen Schlagworten erfassen möchten. Durch Klicken auf „Schlagworte leeren“❺ werden das Datum sowie die Schlagworte gelöscht. Der Dateiname und der Inhalt werden beim Öffnen einer neuen Datei automatisch ersetzt.

Wenn Sie den Speicherort der Dokumente ändern möchten, klicken Sie auf „Pfad ändern“❻. Beachten Sie, dass die Suche nur nach Dokumenten aus dem aktuellen Speicherpfad❼

möglich ist. Dies ermöglicht es, Dokumente in verschiedenen Ordnern abzulegen, um eine strukturierte Archivierung zu ermöglichen.



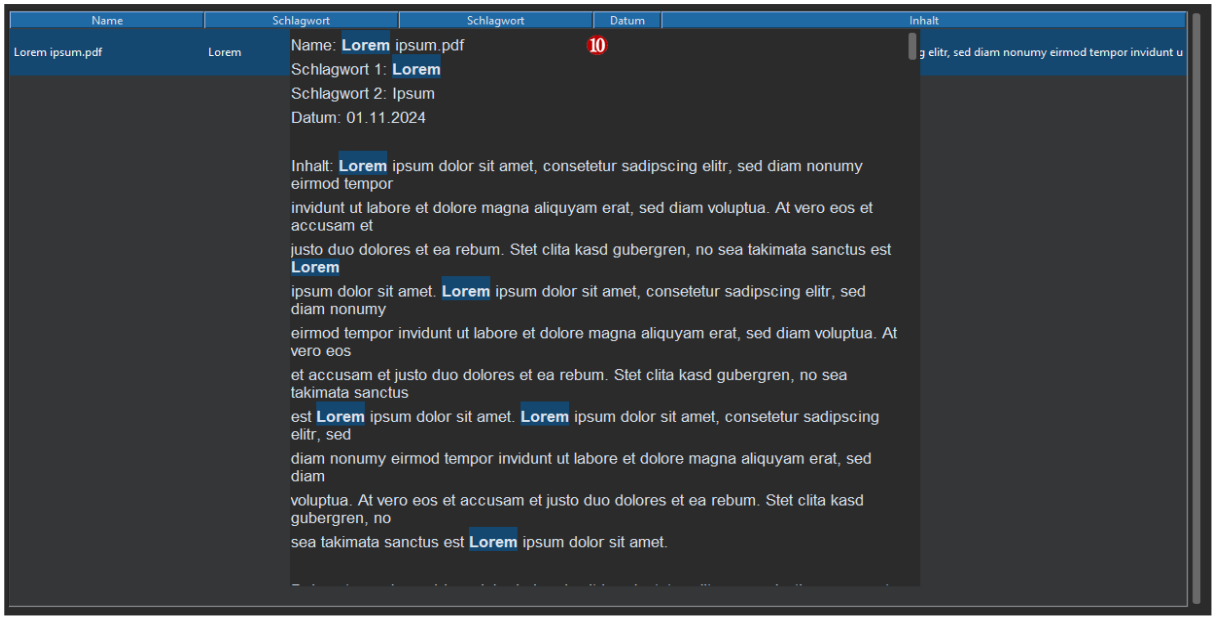
4 Reiter Übersicht

Um nach Dokumenten zu suchen, geben Sie ein oder mehrere Wörter, die im Dokument, im Namen oder in den Schlagwörtern vorkommen, in das Suchfeld❽ in dem Tab „Übersicht“ ein. Die Suche wird nach jeder Eingabe aktualisiert.

Die Buttons „Datei öffnen“ und „Datei löschen“ ❾ sind zunächst ausgegraut und können nicht verwendet werden. Erst wenn eine Zeile in der Tabelle durch Anklicken ausgewählt wurde (blau hinterlegt), werden die Buttons aktiv und können für die ausgewählte Zeile verwendet werden.

Ein Doppelklick auf eines der Suchergebnisse öffnet ein Pop-up-Fenster❿, in dem der Inhalt angezeigt wird. Die Suchbegriffe sind im Pop-up-Fenster blau hervorgehoben, um sie leichter erkennen zu können. Ein Klick außerhalb des Pop-up-Fensters schließt dieses wieder.

Anhang


Name
Schlagwort
Schlagwort
Datum
Inhalt
Lorem ipsum.pdf
Lorem
10
Name: Lorem ipsum.pdf
Schlagwort 1: Lorem
Schlagwort 2: Ipsum
Datum: 01.11.2024
Inhalt: Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor
invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et
justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem
ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy
eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos
et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus
est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed
diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam
voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no
sea takimata sanctus est Lorem ipsum dolor sit amet.

A7 Quellcode (Auszug)

Programmstart:

```python
# Loading and saving the path in the configuration file
def load_path():
    global target_folder
    if os.path.exists("config.json"):
        with open("config.json", "r") as jsonfile:
            data = json.load(jsonfile)
            target_folder = data.get("Storage path", "")
    return target_folder


def save_path():
    path = filedialog.askdirectory()
    if path:
        with open("config.json", "w") as jsonfile:
            json.dump({"Storage path": path}, jsonfile)
        global target_folder
        target_folder = path
        path_entry.configure(state="normal")
        path_entry.delete("0", END)
        path_entry.insert("0", path)
        path_entry.configure(state="readonly")
        message_text = "Speicherpfad gesetzt"
        message.configure(state="normal")
        message.delete(1.0, END)
        message.insert(END, message_text)
        message.configure(text_color="#75F94D", state="disabled")
        create_database()
        read_data()


# Initialisation of the database
def create_database():
    connection = sqlite3.connect(os.path.join(target_folder, "DOPI.db"))
    cursor = connection.cursor()
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS documents (
        id INTEGER PRIMARY KEY,
        name TEXT UNIQUE,
        keyword1 TEXT,
        keyword2 TEXT,
        date TEXT,
        content TEXT)
    ''')
    connection.commit()
    connection.close()


def first_path():
    while not target_folder:
        folder_selection = CTkMessagebox(title="Pfad wählen", message="Vor der ersten Benutzung muss ein Speicherpfad "
                                         "für die Dateien ausgewählt werden.", option_1="Beenden",
                                         option_2="Pfad wählen", width=400, wraplength=280, button_width=100,
                                         cancel_button="none")
        if folder_selection.get() == "Beenden":
            sys.exit()
        save_path()

```

Datenverarbeitung:

```python
# Insert and update data
def insert_data(name, keyword1, keyword2, date, content, segment_archiv):
    connection = sqlite3.connect(os.path.join(target_folder, "DOPI.db"))
    cursor = connection.cursor()
    cursor.execute('SELECT * FROM documents WHERE name = ?', (name,))
    existing_data = cursor.fetchone()

    # Check whether the file already exists
    if existing_data:
        result = CTkMessagebox(title="Konflikt", message=f"Ein Dokument mit dem Namen\n'{name}'\nexistiert bereits. "
                               f"Möchten Sie die Inhalte ersetzen?", option_1="Nein", option_2="Ja", width=450,
                               wraplength=370, button_width=100, cancel_button="none")
        if result.get() == "Nein":
            connection.close()
            return
```

Anhang

```
113
114     # Data is only overwritten if the copy process was successful
115     copied = archive(file, target_folder, segment_archiv)
116     if copied:
117         # If the file already exists, the data is overwritten (Upsert (Update or Insert))
118         cursor.execute('''
119                 INSERT INTO documents (name, keyword1, keyword2, date, content)
120                 VALUES (?, ?, ?, ?, ?)
121                 ON CONFLICT(name) DO UPDATE SET
122                     keyword1 = excluded.keyword1,
123                     keyword2 = excluded.keyword2,
124                     date = excluded.date,
125                     content = excluded.content
126                 ''', (name, keyword1, keyword2, date, content))
127         connection.commit()
128         connection.close()
129         read_data()
130         search_document(search_field_entry)
131     else:
132         message_text = "\nDaten wurden nicht überschrieben!"
133         message.insert(END, message_text)
134         message.configure(text_color="#EB3324")
135         message.configure(state="disabled")
136
137
138 # Open and scan image
139 def open_image():
140     global file
141     file = filedialog.askopenfilename(filetypes=[("Image files", "*.png;*.jpg")])
142     if file:
143         image_file_name = os.path.basename(file)
144         name_entry.configure(state="normal")
145         name_entry.delete("0", END)
146         name_entry.insert("0", image_file_name)
147         name_entry.configure(state="disabled")
148         text = pytesseract.image_to_string(Image.open(file))
149         return text
150
151
152 # Open PDF and extract text
153 def open_pdf():
154     global file
155     file = filedialog.askopenfilename(filetypes=[("PDF files", "*.pdf")])
156     if file:
157         # PDF mit PyPDF öffnen
158         reader = PdfReader(file)
159         pdf_file_name = os.path.basename(file)
160         name_entry.configure(state="normal")
161         name_entry.delete("0", END)
162         name_entry.insert("0", pdf_file_name)
163         name_entry.configure(state="disabled")
164         text = ""
165         for page in reader.pages:
166             text += page.extract_text() + ("\n"*2)
167         return text
168
169
170 # Archive file
171 def archive(source, target_folder, segment_archiv):
172     copied = False
173     if segment_archiv == "Copy":
174         try:
175             shutil.copy(source, target_folder)
176             message.configure(state="normal")
177             message_text = f"Datei erfolgreich kopiert"
178             message.delete(1.0, END)
179             message.insert(END, message_text)
180             message.configure(text_color="#75F94D", state="disabled")
181             copied = True
182         except Exception as e:
183             message.configure(state="normal")
184             message_text = f"Fehler beim Kopieren der Datei: {e}"
185             message.delete(1.0, END)
186             message.insert(END, message_text)
187         return copied
188     else:
189         try:
190             # Extract file name from the source path and create complete target path
191             filename = os.path.basename(source)
192             target_path = os.path.join(target_folder, filename)
193             if os.path.exists(target_path):
194                 os.remove(target_path)
195             shutil.move(source, target_folder)
196             message.configure(state="normal")
197             message_text = f"Datei erfolgreich verschoben"
198             message.delete(1.0, END)
```

Anhang

```
            message.insert(END, message_text)
            message.configure(text_color="#75F94D", state="disabled")
            copied = True
        except Exception as e:
            message.configure(state="normal")
            message_text = f"Fehler beim verschieben der Datei: {e}"
            message.delete(1.0, END)
            message.insert(END, message_text)
        return copied
```

Suche und Pop-up-Fenster:

```
# Search document
def search_document(event):
    global search
    search = search_field_entry.get().lower()
    if search:
        # Separate the search terms with spaces
        search_terms = search.split()
        connection = sqlite3.connect(os.path.join(target_folder, "DOPI.db"))
        cursor = connection.cursor()
        # Create dynamic SQL query for multiple search words
        query = "SELECT * FROM documents WHERE " + " AND ".join(
            ["(LOWER(name) LIKE ? OR LOWER(keyword1) LIKE ? OR LOWER(keyword2) LIKE ? OR LOWER(date) "
             "LIKE ? OR LOWER(content) LIKE ?)"] * len(search_terms)
        ) + " ORDER BY id DESC"
        search_patterns = []
        for term in search_terms:
            search_patterns.extend([f"%{term}%"] * 5)
        # Execute SQL query and retrieve results
        cursor.execute(query, search_patterns)
        data = cursor.fetchall()
        for row in tree.get_children():
            tree.delete(row)
        count = 0
        for row_data in data:
            single_row = []
            for column_data in row_data:
                # Write content in one line and shorten
                column_data = str(column_data).replace("\n", "")[0:300]
                single_row.append(column_data)
            if count % 2 == 0:
                tree.insert("", "end", values=single_row, tags=("evenrow",))
            else:
                tree.insert("", "end", values=single_row, tags=("oddrow",))
            count += 1
        connection.close()
    else:
        read_data()


# Show popup for complete content (Double-Click Event)
def show_popup(content, x, y):
    popup = CTkToplevel(root)
    popup.overrideredirect(True)  # Remove window frames
    popup.geometry(f"{int(x)}+{int(y)}")
    popup.focus_force()  # Sets the focus on the pop-up
    # List of column names (without ID)
    column_names = ["Name", "Schlagwort 1", "Schlagwort 2", "Datum", "Inhalt"]
    # Formatted content: Column name + cell value for each row
    formatted_content = ""
    for row in content:
        for i, value in enumerate(row):
            # Add column name and value
            if value is not None:
                formatted_content += f"{column_names[i]}: {value}\n"
            # Paragraph after Column 3 (Date)
            if i == 3:
                formatted_content += "\n"

    text_box = tk.Text(popup, width=70, height=23, wrap="word", font=("Arial", 13), background="#2B2B2B",
                       foreground="#DCE4EE", border="0")
    scrollbar = CTkScrollbar(popup, command=text_box.yview, fg_color="#2B2B2B")
    scrollbar.pack(side="right", fill="y")
    text_box.configure(yscrollcommand=scrollbar.set)
    text_box.insert("end", formatted_content)
    text_box.configure(state="disabled", spacing1=7)
    text_box.pack(fill="both", expand=True)
    if search:
        highlight_text(text_box, search)

    def close_popup(event):
        popup.destroy()
    popup.bind("<FocusOut>", close_popup)
```