



OCR- DOKUMENTENERKENNUNG

PA 04.03.2025 - 17.03.2025

Lehrling

Hugo Pawlowski

Praktikumsbetrieb

Ajooda AG

Verantwortliche Fachkraft

Lakic Veselin

Experte

Lackner Damian

Nebenexperte

Bassi Diego



1 Inhalt

1	Inhalt.....	1
2	Vorwort.....	4
Teil A.....		5
3	Einleitung	5
3.1	Ausgangslage	5
3.2	Zielsetzung	5
3.3	Mittel und Methoden.....	6
3.4	Vorkenntnisse	6
3.5	Vorarbeiten	6
3.6	Neue Lerninhalte.....	6
3.7	Arbeiten in den letzten 6 Monaten.....	7
3.8	Projektmethode.....	7
4	Teil B.....	8
5	Phase 1: Definitionsphase.....	8
5.1	Anforderungsanalyse	8
5.2	Wissensbeschaffung	9
5.3	Technologieentscheidung	10
5.4	Use Case	11
5.5	Detaillierter Zeitplan	13
5.6	Datensicherung.....	13
6	Phase 2: Planungsphase	13
6.1	Systemarchitektur	13
6.2	Mockup	16
6.3	Testkonzept definieren	21
6.3.1	Unit-Tests.....	21
6.3.2	Integrationstests.....	23
6.3.3	Systemtest & Fehleranalyse	24
6.4	Einrichtung Projektumgebung.....	25
6.4.1	Anforderungen an die Entwicklungsumgebung.....	25
6.4.2	Installation der benötigten Software	25
6.4.3	Einrichtung der Ordnerstruktur	28



7	Phase 3: Realisierungsphase.....	29
7.1	Implementierung der Benutzeroberfläche	29
7.2	OCR-Integration	34
7.2.1	Implementierung der OCR-Verarbeitung.....	34
7.3	Kategorisierung	36
7.3.1	Funktionsweise der Kategorisierung.....	37
7.3.2	Implementierung der Kategorisierung.....	37
7.4	Datei Umbenennen/Sortierung.....	39
7.5	Fehlerbehandlung	42
8	Phase 4: Abschlussphase	44
8.1	Units Test	44
8.2	Integrationstests	46
8.3	Systemtest & Fehleranalyse	47
8.4	Fehlerkorrektur und Code Optimierung	47
8.5	Code Review	48
8.6	Reflexion.....	50
8.7	Fazit	50
9	Anhang.....	51
9.1	Glossar.....	51
9.2	Quellenverzeichnis	52
9.3	Abbildungsverzeichnis	55
9.4	Arbeitsjournal	56
9.4.1	Tag 1, Dienstag, 04.03.2025.....	56
9.4.2	Tag 2, Mittwoch, 05.03.2025	57
9.4.3	Tag 3, Donnerstag, 06.03.2025.....	58
9.4.4	Tag 4, Freitag, 07.03.2025.....	59
9.4.5	Tag 5, Montag, 10.03.2025.....	61
9.4.6	Tag 6, Dienstag, 11.03.2025	62
9.4.7	Tag 7, Mittwoch, 12.03.2025	63
9.4.8	Tag 8, Donnerstag, 13.03.2025.....	64
9.4.9	Tag 9, Freitag, 14.03.2025	65
9.4.10	Tag 10, Montag, 17.03.2025	66



9.5	Projektjournal.....	67
9.6	Quellcode	68



2 Vorwort

Die vorliegende IPA-Abschlussarbeit entstand im Rahmen meiner Ausbildung als Informatiker mit Fachrichtung Applikationsentwicklung. In den vergangenen Jahren habe ich mir umfassende Kenntnisse in der Softwareentwicklung angeeignet, welche ich nun in einem eigenständigen Projekt unter Beweis stellen kann.

Mein Projekt beschäftigt sich mit der Entwicklung eines OCR-Systems zur automatisierten Bilderkennung und Kategorisierung. Der Hintergrund dieses Projekts ist ein häufig auftretendes Problem in Unternehmen: die zeitaufwändige manuelle Verarbeitung von Dokumenten. Um diesen Prozess zu optimieren, soll eine Softwarelösung entwickelt werden, die Dokumente automatisch erkennt, verarbeitet und in vordefinierte Kategorien einordnet.

Während der Arbeit an diesem Projekt konnte ich mein Wissen in Softwarearchitektur, Datenverarbeitung und maschineller Texterkennung (OCR) vertiefen. Die Wahl der geeigneten Technologien und Algorithmen stellte eine besondere Herausforderung dar, ebenso wie die Implementierung einer effizienten und benutzerfreundlichen Lösung.

Ich möchte mich an dieser Stelle bei allen Personen bedanken, die mich in dieser Zeit geholfen haben. Ein besonderer Dank gilt meinem Berufsbildner sowie den Fachpersonen, die mir wertvolle Ratschläge und Feedback gegeben haben.

Diese Arbeit stellt nicht nur den Abschluss meiner Ausbildung dar, sondern ist auch eine wertvolle Erfahrung für meine zukünftige berufliche Laufbahn. Ich hoffe, dass die Ergebnisse meines Projektes eine praxisnahe Lösung für das beschriebene Problem bieten und als Grundlage für zukünftige Weiterentwicklungen genutzt werden wird.

Die Dokumentation ist ausserdem in zwei Teile gegliedert.

Im ersten Teil werden die detaillierte Aufgabenstellung sowie der Ablauf der Arbeit beschrieben. Dabei wird ersichtlich, mit welchen Methoden gearbeitet wurde und welche Vorkenntnisse vorhanden waren.

Der zweite Teil enthält die eigentliche Projektdokumentation. Hier wird die Umsetzung, aufgetretene Probleme sowie die durchgeföhrten Tests erläutert.



Teil A

3 Einleitung

In den heutigen Unternehmen fallen täglich grosse Mengen digitaler Dokumente an, darunter Rechnungen, Bankunterlagen und Versicherungsbelege. Ohne eine effiziente Verarbeitung dieser Dokumente entstehen hohe Zeitaufwände und Fehler. Die Ajooda AG steht vor genau dieser Herausforderung, Dokumente müssen bisher manuell geöffnet und geprüft werden – ein mühsamer Prozess, der mit zunehmenden Dokumenten immer aufwendiger wird.

Um dieses Problem zu lösen, soll ein OCR-System (Optical Character Recognition) implementiert werden, dass die Dokumentenverarbeitung automatisiert. Dadurch lässt sich viel Zeit sparen und das Risiko von Fehlern durch menschliche Unachtsamkeit reduzieren. Studien zeigen, dass OCR-Technologie den Zeitaufwand für manuelle Dateneingaben um bis zu 80 % senken und Fehler um bis zu 50 % verringern können ([Quelle](#)).

3.1 Ausgangslage

Bei der Ajooda AG werden digitale Dokumente derzeit manuell geprüft, sortiert und abgelegt. Dieser Prozess ist nicht nur zeitaufwendig und fehleranfällig, sondern wird durch das Wachstum des Unternehmens zunehmend schwieriger.

Ein weiteres Problem ist die fehlende einheitliche Ablagestruktur. Dadurch kann die Suche nach bestimmten Dokumenten ziemlich lange dauern und ist oft ineffizient. Das Ziel des Projekts ist es deshalb, ein OCR-System einzuführen, das Dokumente automatisch erkennt, sortiert und ablegt. So soll nicht nur die Bearbeitung schneller werden, sondern auch die Fehlerquote sinken und die Ablage nachvollziehbarer werden.

3.2 Zielsetzung

Ziel des Projekts „**Entwicklung eines OCR-Systems zur automatisierten Bilderkennung und Kategorisierung**“ ist die Realisierung einer Anwendung, die den beschriebenen manuellen Prozess ablöst. Konkret soll eine Software entwickelt werden, mit der Benutzer einen **Ordnerpfad** angeben können, woraufhin alle darin enthaltenen Dokumente automatisiert analysiert werden

Die geplante Anwendung liest die Dokumente mittels OCR-Texterkennung aus und kategorisiert sie anschliessend selbstständig. Im Folgenden sind die **Hauptfunktionen** der Anwendung aufgeführt:



- **OCR-Texterkennung:** Auslesen des Inhalts von Dokumenten mittels eines OCR-Dienstes (geplant ist die Nutzung der Google Cloud Vision API). Dadurch werden Bilder (z. B. JPEG/PNG-Format) in durchsuchbaren Text umgewandelt.
- **Automatische Kategorisierung:** Anhand definierter **Schlüsselwörter** im erkannten Text wird jedes Dokument einer Kategorie zugeordnet. Vorgesehene Kategorien sind z. B. "Bank", "Krankenkasse", "Lohnausweis" oder "Kredit"

Die Dateien werden entsprechend umbenannt (mit kategorietypischem Präfix und einer laufenden Nummer) und in einen entsprechenden Ordner eingesortiert.

- **Fehlererkennung und -behandlung:** Das System erkennt **fehlerhafte oder nicht verarbeitbare Dokumente** (etwa wenn kein Text erkannt wird oder das Format ungültig ist). Solche Fälle werden protokolliert und dem Benutzer deutlich gemeldet, damit keine Datei unbemerkt unbehandelt bleibt.
- **Benutzeroberfläche mit Statusanzeige:** Eine benutzerfreundliche GUI zeigt den Fortschritt der Verarbeitung und die Ergebnisse für jedes Dokument an. Der Benutzer kann dort z. B. die erkannte Kategorie, Ausschnitte des extrahierten Textes sowie die Verarbeitungszeit pro Dokument einsehen

Durch diese Funktionen gibt das System einen hohen **Mehrwert** für das Unternehmen. Routineaufgaben werden beschleunigt und standardisiert, was **Arbeitszeit einspart**, und eine Ablage der Dokumente gibt. Mitarbeitende können sich auf die inhaltliche **Analyse der Dokumente** konzentrieren, anstatt Zeit mit Sortieren und Suchen zu verbringen. Zudem erhöht die automatische Kategorisierung die **Transparenz**: Dokumente sind einheitlich benannt und abgelegt, was ihre Wiederauffindbarkeit verbessert. Insgesamt trägt das Projektziel dazu bei, die Prozesse der Ajooda AG im Dokumentenmanagement effizienter und zuverlässiger zu gestalten.

3.3 Mittel und Methoden

Hardware: Lokale Server mit Rechenleistung und Speicherplatz für die Verarbeitung von Dateien.
Software: Google Cloud Vision Programmiersprachen: Node.js für Backend-Entwicklung, JavaScript für clientseitige Skripte, und HTML/CSS

3.4 Vorkenntnisse

Vorkenntnisse in den eingesetzten Technologien wie Node.js, JavaScript, sowie in grundlegende Webentwicklung mit HTML und CSS

3.5 Vorarbeiten

Keyfile von Google Cloud Vision API

3.6 Neue Lerninhalte

Integration und Verwendung von OCR-Software für Bilderkennung.



3.7 Arbeiten in den letzten 6 Monaten

Art der Arbeiten: Entwicklung und Anpassung von Formularen auf WordPress mit Fluent Forms, Elementor und CSS. Behebung von Fehlern in HubSpot. Analyse mit Google Analytics. Erstellung von Dokumentationen. Durchführung von Tests. Eingesetzte Tools: WordPress Fluent Forms Elementor HubSpot Google Analytics ChatGPT Dokumentationen (tutorials) Dokumentationstools (Word, slab) Testumgebungen und Debugging-Tools.

3.8 Projektmethode

Für die Umsetzung meines Projekts habe ich verschiedene Projektmethoden analysiert, darunter das **IPERKA-Modell, Wasserfallmodell, V Modell etc.** Nach Analyse verschiedener Methoden erwies sich das **Vierphasenmodell** als am besten geeignet für dieses Projekt.

Das **Vierphasenmodell** ist eine zuverlässliche Methode in der Softwareentwicklung, die mein Projekt in die folgenden vier klar definierten Phasen unterteilt:

1. **Definitionsphase:** In dieser Phase werden die Anforderungen analysiert und eine detaillierte Planung erstellt. Hierzu gehören die Anforderungsanalyse, die Wissensbeschaffung sowie die Technologieentscheidung.
2. **Planungsphase:** In dieser Phase wird das Projekt strukturiert, Aufgaben und Verantwortlichkeiten werden verteilt, und eine detaillierte Zeitplanung wird erstellt.
3. **Realisierungsphase:** Die eigentliche Implementierung des OCR-Systems findet hier statt. Die zuvor definierten Anforderungen werden umgesetzt, getestet und im Laufe verbessert.
4. **Abschlussphase:** In dieser letzten Phase erfolgt eine abschließende Validierung des Systems, das Testen der gesamten Anwendung sowie die Dokumentation und Vorbereitung der Abgabe.

Der Vorteil des Vierphasenmodells für mein Projekt liegt in seiner **Einfachheit und Struktur**. Es lässt Anpassungen während der Entwicklung zu, ohne die Planung durcheinanderzubringen. Gleichzeitig hilft es, das Projekt übersichtlich zu halten.

Durch diese Methode wird sichergestellt, dass alle notwendigen Schritte von der Planung über die Umsetzung bis hin zur finalen Dokumentation systematisch sind.



4 Teil B

5 Phase 1: Definitionsphase

5.1 Anforderungsanalyse

In der **Anforderungsanalyse** wurde festgelegt, was das System können muss und welche Eigenschaften es haben soll. Dabei werden **funktionale Anforderungen** definiert, die Kernfunktionen der Texterkennung und Kategorisierung. Zusätzlich gibt es **nicht-funktionale Anforderungen**, die sicherstellen, dass das System schnell, benutzerfreundlich arbeitet.

Funktionale Anforderungen

Kategorie	Anforderung
Dateiverarbeitung	Das System muss Ordnerpfade verarbeiten und alle darin enthaltenen Dateien erfassen.
OCR-Erkennung	Texterkennung muss mit Google Cloud Vision API durchgeführt werden, um Text aus Bilddateien zu extrahieren.
Kategorisierung	Erkannter Text wird anhand vordefinierter Schlüsselwörter Kategorien wie Bank, Krankenkasse, Lohnausweis, Kredit zugeordnet.
Datei-Management	Dateien werden basierend auf der erkannten Kategorie automatisch umbenannt und in den passenden Ordner verschoben.
Benutzeroberfläche	Die Benutzeroberfläche zeigt Fortschritt der Verarbeitung, Ergebnisse der Kategorisierung und erkannte Texte an.
Fehlerbehandlung	Fehlerhafte oder nicht verarbeitbare Dateien müssen erkannt, protokolliert und dem Benutzer angezeigt werden.
Systemintegration	Das System soll in bestehende Unternehmenssysteme integrierbar sein (z. B. Cloud-Speicher, lokale Datenbanken).
Protokollierung	Es wird ein Fehler- und Verarbeitungsprotokoll geführt, um die Nachverfolgbarkeit zu gewährleisten.

Nicht-funktionale Anforderungen

Kategorie	Anforderung
Performance	Das System muss in der Lage sein, grosse Mengen von Dokumenten zu verarbeiten.
Benutzerfreundlichkeit	Die GUI muss intuitiv und einfach bedienbar sein, auch für nicht-technische Nutzer.
Skalierbarkeit	Das System sollte so aufgebaut sein, dass zukünftige Erweiterungen (z. B. neue Kategorien) einfach integriert werden können.



Sicherheit	Sichere Verarbeitung von Dokumenten, insbesondere bei der Nutzung von Cloud-Diensten.
Zuverlässigkeit	OCR-Ergebnisse und Klassifizierungen müssen mit hoher Wahrscheinlichkeit korrekt sein, um manuelle Nacharbeit zu minimieren.
Wartbarkeit	Der Code muss gut dokumentiert und modular aufgebaut sein, um einfache Wartung zu ermöglichen.

5.2 Wissensbeschaffung

Während der Projektarbeit habe ich mich mit verschiedenen Technologien auseinandergesetzt, um eine Grundlage für die Umsetzung meines Projekts zu schaffen, und zu schauen, ob es vielleicht sinnvoll wäre eine andere Technologie zu nehmen, die günstiger ist oder sogar eine bessere Performance hat. Dazu gehörte die Recherche über OCR-Technologien, die Integration in Node.js sowie die Entwicklung einer Benutzeroberfläche. Hierbei habe ich offizielle Dokumentationen, Fachartikel und Best Practices analysiert.

Verständnis von OCR-Technologien

Google Cloud Vision API: Ich habe mich mit den Funktionen und Möglichkeiten dieser API beschäftigt, insbesondere mit der Texterkennung. Die offizielle Dokumentation auf [Google Cloud](#) bot mir einen Überblick über die Implementierungsmöglichkeiten und API-Funktionen.

Tesseract OCR: Um eine Alternative zur Cloud-Lösung zu evaluieren, habe ich mich mit Tesseract OCR, einer Open-Source-Engine, befasst. Ich habe deren Stärken und Schwächen angeschaut, insbesondere im Vergleich zur Google Cloud Vision API. Die offizielle Dokumentation auf Tesseract OCR sowie Erfahrungsberichte in Entwicklerforen haben mir geholfen, die richtige Wahl für mein Projekt zu treffen.

Integration von OCR in Node.js

Google Cloud Vision API mit Node.js: Ich habe geschaut, wie ich die API in eine Node.js-Anwendung integrieren kann. Die offiziellen Dokumentationen von Google (Google Cloud Node.js Client) gab mir einen ersten Einblick. Darüber hinaus fand ich praxisnahe Anleitungen auf Plattformen wie [Medium](#) und [YouTube](#), dazu wie ich die API in meine Anwendung implementieren kann.

Entwicklung der Benutzeroberfläche

Für die Benutzeroberfläche habe ich recherchiert, wie eine klassische Web-App mit HTML, CSS und JavaScript entwickelt wird. Dabei habe ich verschiedene Dokumentationen und Videos angeschaut.



Best Practices und Fallstudien

Ich habe gezielt nach Fallstudien und Projekten gesucht, die ähnliche Anforderungen hatten, um aus deren Erfahrungen zu lernen. Besonders hilfreich waren Blogbeiträge und technische Berichte auf [Dev.to](#) sowie Diskussionen in Entwicklerforen wie [Reddit](#). Diese Recherchen halfen mir, bewährte Methoden und Lösungsansätze in mein eigenes Projekt zu übernehmen.

5.3 Technologieentscheidung

Nach ausführlicher Recherche und Analyse habe ich mich für die Nutzung der **Google Cloud Vision API** als OCR-Technologie entschieden. Die Entscheidung basiert auf mehreren Faktoren:

- **Geschwindigkeit und Effizienz:** Die Cloud-Lösung ist deutlich schneller als lokale OCR-Engines wie Tesseract, insbesondere bei grösseren Dokumentenmengen. Durch die Nutzung von Google Cloud-Servern ist die Texterkennung nahezu in Echtzeit. Die API bietet zudem eine hohe Erkennungsgenauigkeit, selbst bei komplexen oder schlecht lesbaren Dokumenten ([Google Cloud Vision API](#)).
- **Zuverlässigkeit und Sicherheit:** Da die Google Cloud Vision API in einer professionellen Umgebung läuft, ist sie stabiler und weniger fehleranfällig als eine lokal betriebene OCR-Lösung. Außerdem erfüllt sie hohe Sicherheitsstandards bei der Datenverarbeitung.
- **Bessere Texterkennung und Analyse:** Die API nutzt maschinelles Lernen zur Verbesserung der Texterkennung und kann auch Layouts, Handschriften und mehrsprachige Texte besser auswerten als herkömmliche OCR-Engines.
- **Dokumentation:** Google Cloud Vision hat viel grössere Dokumentation als andere OCR Anbieter dazu auch noch ein viel grössere Community.

Für die Implementierung habe ich mich für **Node.js** als Backend-Technologie entschieden, da sich die Google Cloud Vision API damit einfach integrieren lässt und es sich gut für serverseitige Anwendungen eignet. Das Frontend wird mit **HTML und CSS** umgesetzt, da diese Technologien plattformunabhängig sind und sich leicht anpassen lassen. Diese Kombination ist weit verbreitet, sodass ich viele Tutorials und Dokumentationen dazu finden kann.



5.4 Use Case

Das folgende Use-Case-Diagramm zeigt die wichtigsten Anwendungsfälle des OCR-Systems, das in diesem Projekt entwickelt wird. Es stellt den Hauptakteur dar, seine Interaktionen mit dem System und die zentralen Funktionen der Anwendung.

Akteure

Benutzer: Der Hauptakteur, der Dokumente hochlädt und die Verarbeitungsergebnisse einsehen kann.

Anwendungsfälle

1. **Dokument hochladen:** Der Benutzer lädt eine oder mehrere Bilddateien (z. B. PNG, JPEG) in das System hoch.
2. **OCR Texterkennung:** Die Google Cloud Vision API extrahiert den Text aus den hochgeladenen Dokumenten.
3. **Dokument kategorisieren:** Das System analysiert den extrahierten Text und ordnet das Dokument einer vordefinierten Kategorie zu (z. B. "Bank", "Krankenkasse", "Lohnausweis", "Kredit").
4. **Fehler erkennen:** Falls die Texterkennung fehlschlägt oder das Dokument nicht lesbar ist, wird eine Fehlermeldung generiert.
5. **Ergebnisse anzeigen:** Der Benutzer kann die Ergebnisse der Texterkennung und Kategorisierung über eine Benutzeroberfläche einsehen.
6. **Dokumente speichern:** Nach erfolgreicher Verarbeitung wird die Datei automatisch umbenannt und in den passenden Ordner verschoben.

Ablauf des Use-Case-Diagramms

1. **Actor (Benutzer) → Ordnerpfad eingeben**
 - Der Benutzer gibt den Pfad des Ordners ein, in dem sich die zu verarbeitenden Dokumente befinden.
2. **Ordnerpfad eingeben → Texterkennung durchführen**
 - Nach der Eingabe des Ordnerpfads startet das Programm nachdem man auf «Evaluieren» gedrückt hat.
3. **Texterkennung durchführen → Google Cloud Vision**
 - Die Texterkennung wird mithilfe des externen Dienstes *Google Cloud Vision* durchgeführt.



4. Texterkennung durchführen → Dokumente klassifizieren

- Der erkannte Text wird analysiert und klassifiziert, um verschiedene Dokumenttypen zu unterscheiden.

5. Dokumente klassifizieren → Speichern und Umbenennen

- Die klassifizierten Dokumente werden in der entsprechenden Struktur gespeichert und umbenannt.

6. Speichern und Umbenennen → Ergebnis anzeigen

- Nach der erfolgreichen Verarbeitung werden die Ergebnisse für den Benutzer sichtbar gemacht.

7. Texterkennung durchführen → Fehler anzeigen

- Falls während der Texterkennung ein Problem auftritt, wird eine Fehlermeldung generiert.

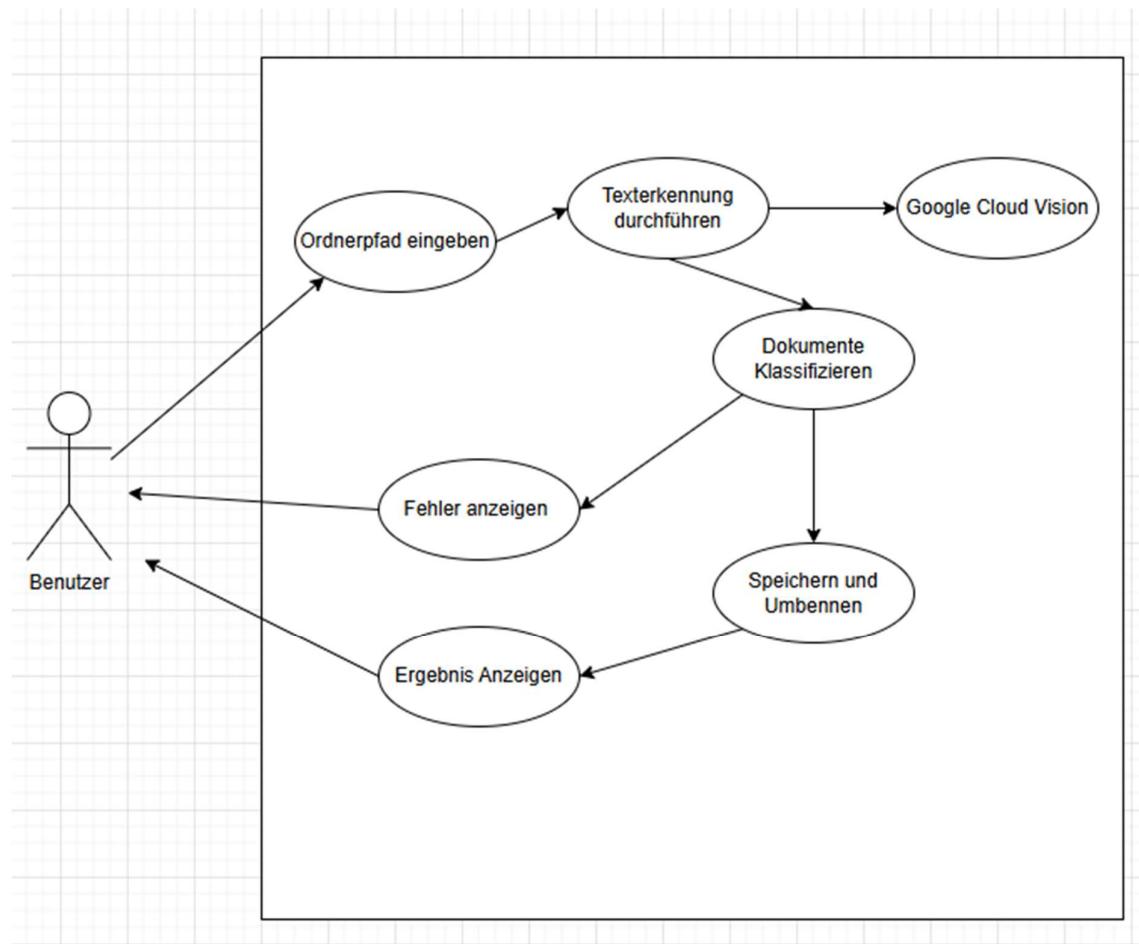
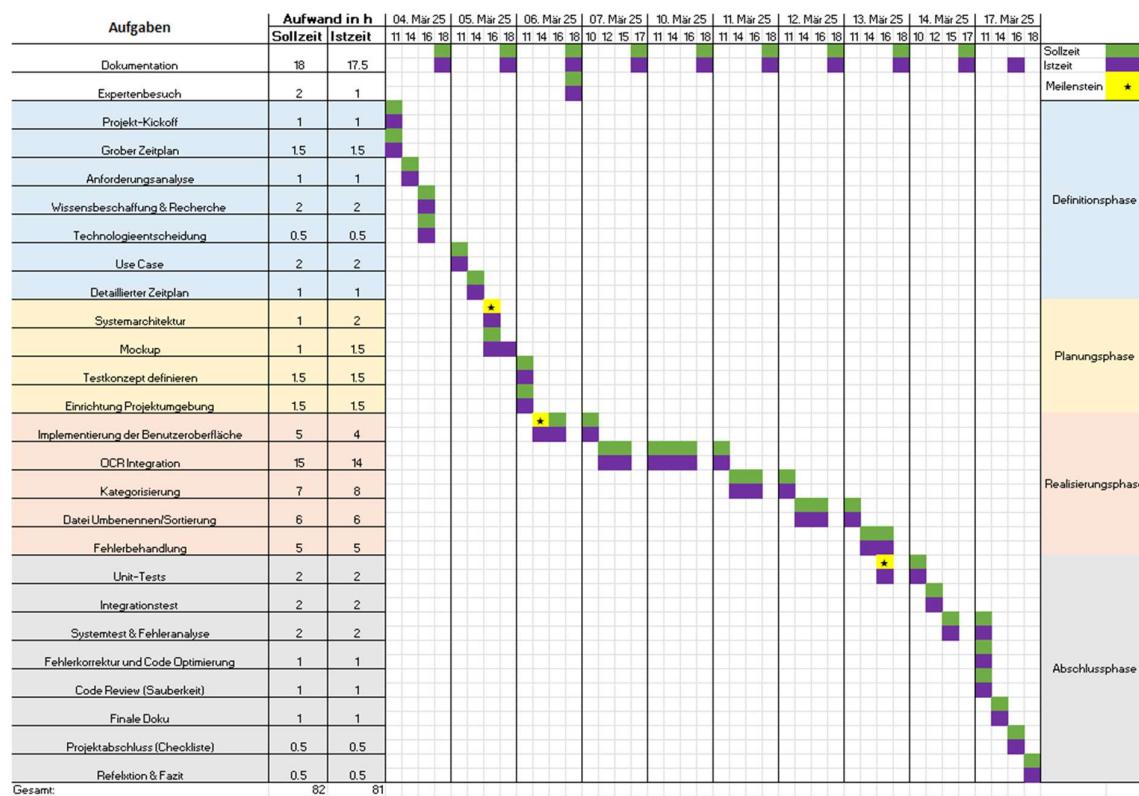


Abbildung 1: Use-Case-Diagramm der OCR-Dokumentenverarbeitung



5.5 Detaillierter Zeitplan



5.6 Datensicherung

Um Datenverluste zu vermeiden, sichere ich meine Projektdateien regelmäßig auf einem **USB-Stick**. Täglich speichere ich alle relevanten Dokumente, den Quellcode sowie die aktuelle Projektversion auf dem Stick. Dies stellt sicher, dass ich im Falle eines technischen Problems jederzeit auf eine aktuelle Version meines Projekts zugreifen kann.

6 Phase 2: Planungsphase

6.1 Systemarchitektur

1. Architekturübersicht

Die Anwendung soll einer **Client-Server-Architektur** und verarbeitet hochgeladene Dokumente durch eine OCR-Erkennung und Kategorisierung. Der grundsätzliche Ablauf ist:

- **Eingabe:** Benutzer laden Dokumente hoch über die Benutzeroberfläche.
- **Verarbeitung:**



- **Express.js (Backend-Framework):** Verwaltung von API-Anfragen und Routing
- **OCR-Modul:** Texterkennung durch Google Cloud Vision API.
- **Kategorisierung:** Analyse des ausgegebenen Textes und Zuweisung einer Kategorie.
- **Fehlermanagement:** Erkennung fehlerhafter Dateien.
- **Ausgabe:**
 - Speicherung der analysierten Dokumente im passenden Ordner.
 - Anzeige der Ergebnisse auf der Benutzeroberfläche.

Verwendete Technologien:

- **Backend:** Node.js mit Express.js zur Verarbeitung von Anfragen und Verwaltung der API.
- **OCR:** Google Cloud Vision API zur Texterkennung.
- **Frontend:** HTML, CSS und JavaScript.
- **Speicherung:** Lokales Dateisystem und Google Cloud Storage für Uploads.

2. Hauptkomponenten des Systems

Referenzieren **Use Case** Anwendungsanfälle, S.8

3. Datenfluss & Prozessablauf

Referenzieren **Use Case** Ablauf, S.8

4. Sicherheits- & Performance-Aspekte

- **Sicherheit:**
 - API-Schlüssel werden sicher in .env-Dateien gespeichert.
 - Fehlerprotokolle helfen, Probleme schnell zu identifizieren.
- **Performance:**
 - **Asynchrone Verarbeitung** von Dokumenten zur Vermeidung von Engpässen.
 - **Batch-Verarbeitung**, um mehrere Dateien gleichzeitig zu analysieren.



- **Caching-Mechanismen**, um wiederholte Anfragen zu optimieren.

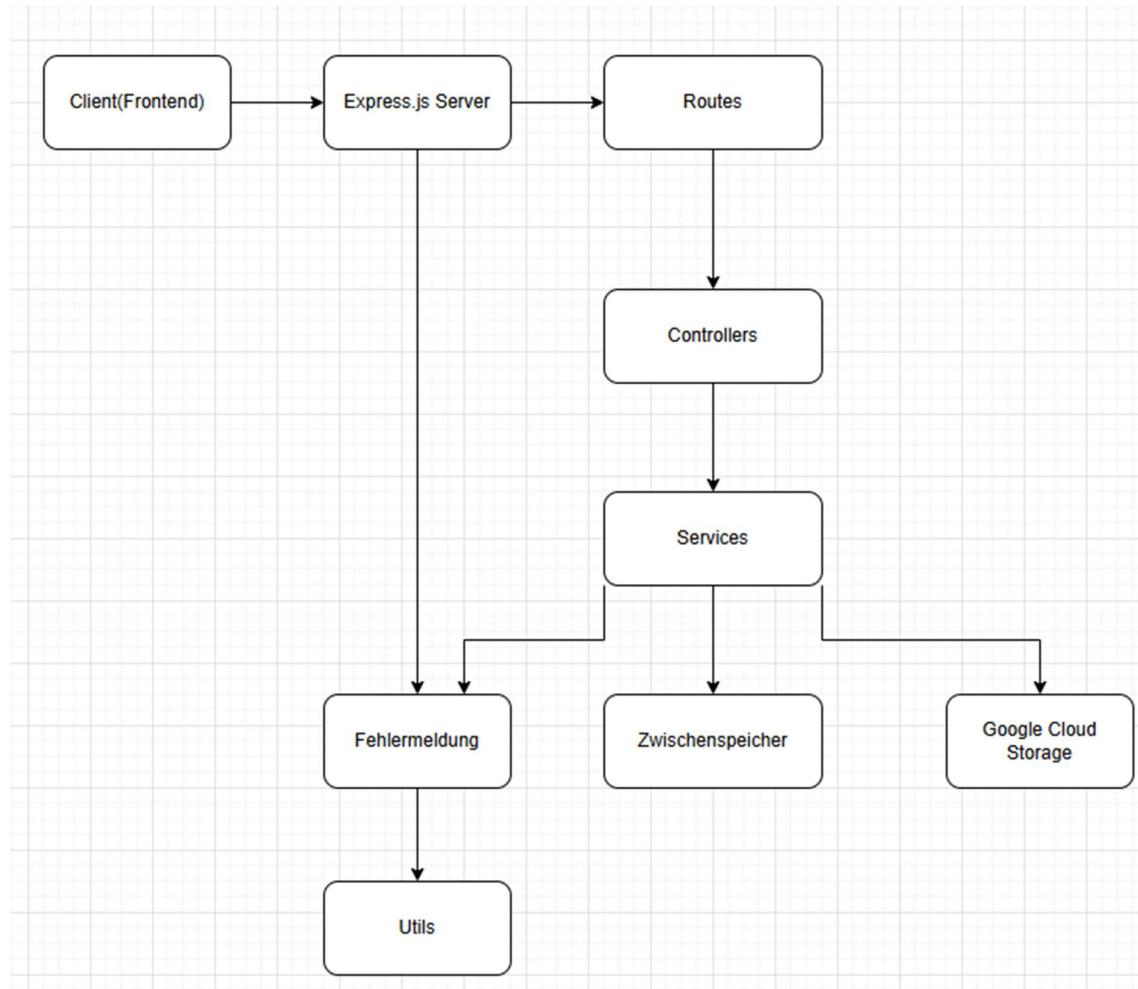


Abbildung 2: Prozessablauf

- **Client (Frontend)**
- **Express.js Server**
- **Routes (fileRoutes, folderRoutes)**
- **Controllers (folderController)**
- **Services (fileService, ocrService, categoryService)**
- **Google Cloud Storage**
- **Datenbank (Metadaten-Speicherung)**
- **Fehlermanagement (Logging, Error Handling)**
- **Utils (config.js, fileUtils.js)**



6.2 Mockup

Mein Mockup ist eine visuelle Darstellung der Benutzeroberfläche, die das geplante Design und die Funktionen des Systems zeigt. Ziel dieses Mockups ist es, eine **benutzerfreundliche Oberfläche** für das OCR-Dokumentensystem zu entwerfen, die eine einfache **Dokumentenerfassung, Verarbeitung und Anzeige der Ergebnisse** zeigt.

Das erste Mockup zeigt die grundlegende Startseite der Anwendung mit der **strukturierten Benutzeroberfläche**. Die Hauptbestandteile sind:

- **Header mit Menüleiste** → Repräsentiert die Anwendung als eine Web-App.
- **Titel „Custom Folder Processing“** → Gibt an, dass der Benutzer hier einen Ordner auswählen und analysieren kann.
- **Zentrale Auswahlfläche „Select Folder“** → Ein abgetrennter Bereich, in dem der Benutzer seinen Ordner für die Texterkennung und Kategorisierung auswählen kann.
- **Zwei Schaltflächen:**
 1. „**Choose Folder**“ → Ermöglicht dem Benutzer, einen lokalen Ordnerpfad einzugeben.
 2. „**Evaluate**“ → Startet die Analyse des gewählten Ordners und die Texterkennung der enthaltenen Dokumente.

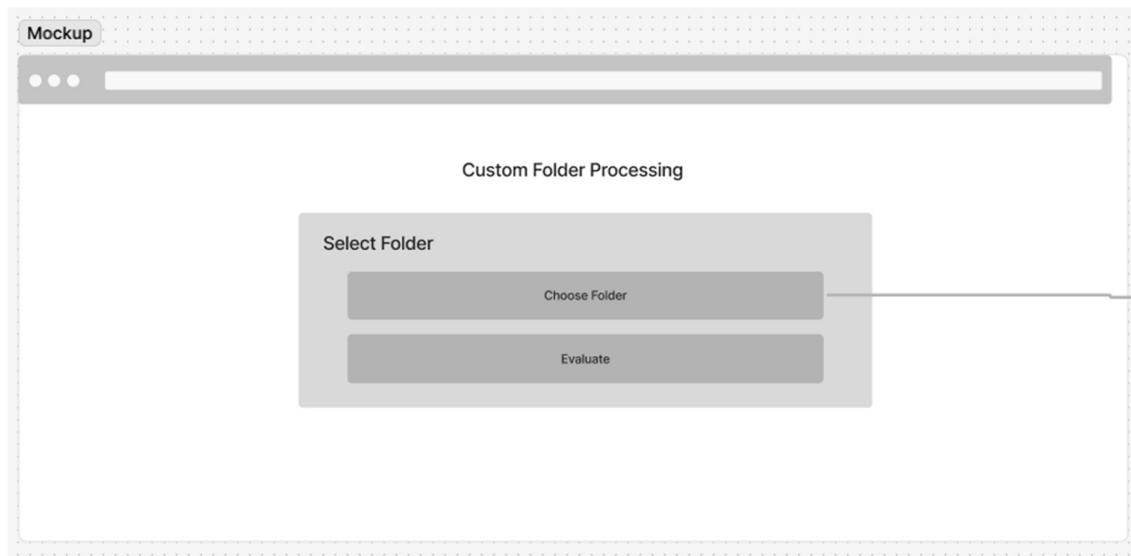


Abbildung 3: Mockup Startseite

Zweiter Schritt: Pop-Up zur Ordnerauswahl

Nachdem der Benutzer auf die Schaltfläche „**Choose Folder**“ geklickt hat, erscheint ein **Pop-Up-Fenster** mit einer **Eingabe für den Ordnerpfad**. Die Hauptsachen dieses Fensters sind:



- **Textfeld „Enter Folder Path“** → Zeigt den aktuellen Pfad des ausgewählten Ordners an.
- **Schaltfläche „OK“** → Bestätigt die Auswahl des Ordners und leitet die Dokumentenverarbeitung ein.
- **Schaltfläche „Cancel“** → Ermöglicht dem Benutzer, den Vorgang abzubrechen und zurück zur Hauptansicht zu kehren.

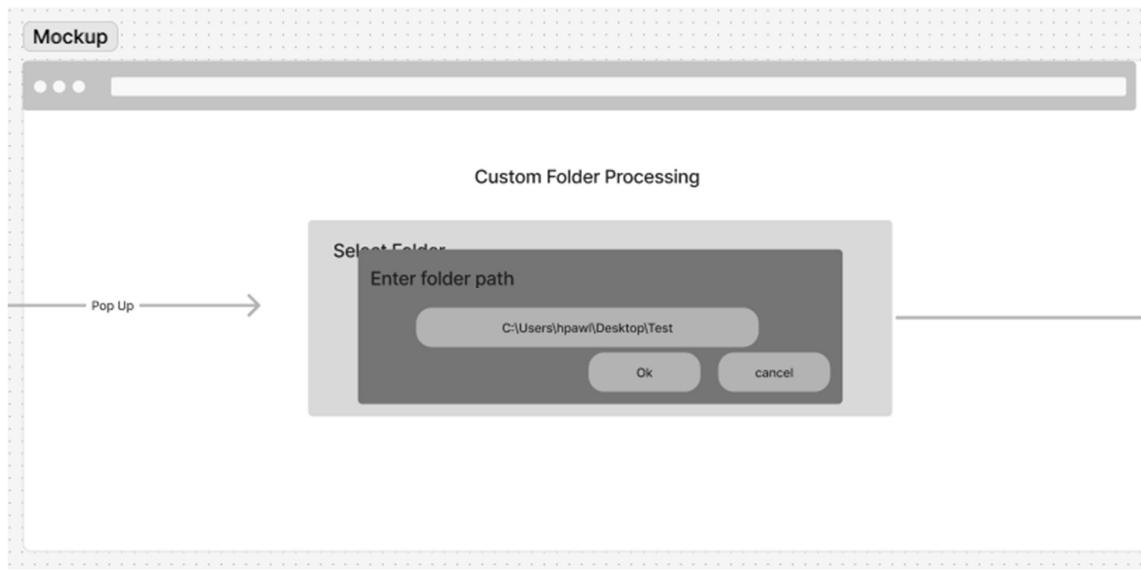


Abbildung 4: Mockup Pop Up

Dritter Schritt: Bestätigung der Auswahl

Nachdem der Benutzer im Pop-Up-Fenster auf „OK“ geklickt hat, wird das Fenster geschlossen und der zuvor ausgewählte Ordnerpfad in der Hauptansicht übernommen. Dies bedeutet:

- Die „Choose Folder“-Schaltfläche ist nun mit dem gewählten Ordner verknüpft.
- Der Benutzer kann nun auf „Evaluate“ klicken, um die OCR-Verarbeitung zu starten.
- Falls nötig, kann der Benutzer erneut auf „Choose Folder“ klicken, um eine andere Auswahl zu treffen.

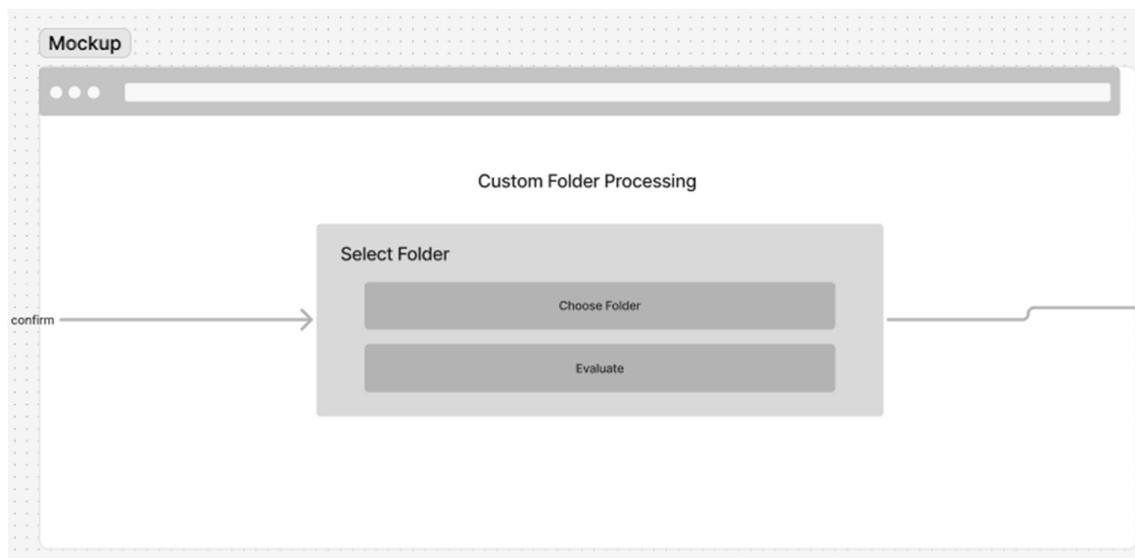


Abbildung 5: Mockup Eingabe Pfad

- **Vierter Schritt: Anzeige der Analyseergebnisse**

Nachdem der Benutzer auf „Evaluate“ geklickt hat, beginnt das System mit der **Analyse der hochgeladenen Dokumente**. Sobald die Verarbeitung abgeschlossen ist, erscheint eine **Ergebnisübersicht**, die folgende Informationen enthält:

- **Tabelle mit den ausgewerteten Dateien** → Listet alle analysierten Dateien aus dem gewählten Ordner auf.
- **Spalten der Tabelle:**
 - **Folder** → Name des analysierten Ordners.
 - **Original File Name** → Ursprünglicher Name der Datei.
 - **New File Name** → Automatisch generierter Name nach der Kategorisierung.
 - **Category** → Die erkannte Dokumentenkategorie (z. B. Bank, Krankenkasse, Lohnausweis).
 - **Detected Text** → Der extrahierte Text aus dem Dokument.
 - **OCR Method** → Gibt an, welche OCR-Technologie verwendet wurde (z. B. Google Cloud Vision).
 - **Processing Time (s)** → Zeigt die Verarbeitungsdauer pro Datei an.
- **Beispielhafte Verarbeitung:**



- Datei „decrypted“ wurde als „**Bank1**“ umbenannt und der Kategorie „**Bank**“ zugeordnet.
 - Datei „file05“ konnte keiner Kategorie zugeordnet werden, da kein Text erkannt wurde.
- **Interaktive Elemente:**
 - „**Show More**“ → Falls ein Dokument viel Text enthält, kann der Benutzer eine detaillierte Ansicht öffnen.

The mockup shows a user interface with a header bar labeled "Mockup". Below it is a table with the following data:

Folder	Original File Name	New File Name	Category	Detected Text	OCR Method	Processing Time (s)
Test	decrypted	Bank1	Bank	Luzern Kantonal Bank Show More	google-cloud-vision	0.31
Test	file05	file05	No category detected	No text found	google-cloud-vision	0.55

Abbildung 6: Mockup Tabellenansicht

Falls ein Dokument viel Text enthält, kann der Benutzer die „**Show More**“-Schaltfläche klicken, um zusätzliche Inhalte anzuzeigen. Dies zeigt eine erweiterte Ansicht in der Spalte „**Detected Text**“, die nun mehr Informationen anzeigt:

- **Erkannter vollständiger Text** → Alle extrahierten Inhalte aus dem Dokument, z. B. IBAN-Nummern, Beträge, Namen oder Adressen.
- **Mehrzeilige Darstellung** → Falls der erkannte Text mehrere Zeilen umfasst, werden diese entsprechend angezeigt.
- **Erneute Reduzierung durch „Show Less“** → Falls die Ansicht zu umfangreich wird, kann der Benutzer den Text durch Klick auf „Show Less“ wieder verkleinern.



Mockup

Folder	Original File Name	New File Name	Category	Detected Text	OCR Method	Processing Time (s)
Test	decrypted	Bank1	Bank	Luzern Kantonal Bank IBAN 60645645 Betrag 5000 CHF AN Werner Lusse Luzern Luzern Musterstrasse Show less	google-cloud-vision	0.31

Abbildung 7: Mockup mehr Text anzeigen



6.3 Testkonzept definieren

Das Testkonzept beschreibt die Strategie zur Qualitätssicherung des OCR-Systems, das zur automatisierten Erkennung und Kategorisierung von Dokumenten eingesetzt wird. Ziel ist es, Fehler frühzeitig zu herauszufinden und eine hohe Genauigkeit der Texterkennung sowie Klassifizierung sicherzustellen.

Testumgebung

- **Betriebssystem:** Windows
- **Software:** Node.js, Express.js, Google Cloud Vision API, Electron.js
- **Testdaten:**
 - **Erlaubte Dateiformate:** JPG, PNG
 - **Nicht unterstützte Formate:** PDF, DOCX, TXT
 - **Unterschiedliche Inhalte:** Verschiedene Kategorien wie Bank, Krankenkasse, Lohnausweise
 - **Schlechte Bildqualität:** Unscharfe Texte, kleine Schriftgrößen

Testziele

Die Hauptziele des Testprozesses sind:

- **Sicherstellung der Funktionalität:** Überprüfung, ob das OCR-System Texte aus Bilddateien korrekt erkennt und diese gemäss den definierten Kategorien klassifiziert.
- **Gewährleistung der Zuverlässigkeit:** Sicherstellen, dass das System unter verschiedenen Bedingungen stabil arbeitet und Fehler angemessen behandelt.
- **Überprüfung der Performance:** Messung der Effizienz des Systems in Bezug auf Verarbeitungsgeschwindigkeit und Ressourcenverbrauch.
- **Validierung der Benutzerfreundlichkeit:** Sicherstellung, dass die Benutzeroberfläche intuitiv ist und der Benutzer den Verarbeitungsstatus sowie die Ergebnisse leicht nachvollziehen kann.

6.3.1 Unit-Tests

Unit-Tests testen **einzelne Funktionen oder Module** des Systems isoliert, um sicherzustellen, dass sie korrekt arbeiten. In diesem Projekt werden Unit-Tests genutzt, um:



- Die **OCR-Verarbeitung** unabhängig zu testen (korrekte Texterkennung für verschiedene Bildformate).
- Die **Kategorisierungslogik** zu überprüfen (Erkennt das System die richtigen Kategorien?).
- Die **Dateiverarbeitung** zu validieren (Werden Dateien korrekt gespeichert und umbenannt?).
- **Fehlerbehandlung:** Sicherstellen, dass das System angemessen auf verschiedene Fehlerquellen reagiert, wie z. B. ungültige Dateiformate oder leere Inhalte

Konzept Unit Testfälle:

Test-ID	Testfall	Beschreibung	Erwartetes Ergebnis
UT-01	OCR-Textverarbeitung	Testet ob ein klares Bild korrekt erkannt wird	Der Text wird vollständig extrahiert
UT-02	Unleserliches Bild	Testet, ob verschwommene Bilder Fehler verursachen	System meldet „Kein Text erkannt“
UT-03	Kategorisierung mit Schlüsselwörtern	Prüft, ob Text korrekt in Kategorien eingeordnet wird	Datei wird richtig klassifiziert
UT-04	Datei-Speicherung	Überprüft, ob Dateien nach der Verarbeitung gespeichert werden	Datei ist im korrekten Ordner zu finden

Best Practices:

- **Isolierung:** Jede Funktion sollte unabhängig getestet werden, ohne Abhängigkeiten zu anderen Teilen des Systems.
informatedigital.com[+4guru99.com](http://4guru99.com)[+4zaptest.com](http://4zaptest.com)[+4](http://4)
- **Regelmässigkeit:** Unit-Tests sollten bei jeder Codeänderung ausgeführt werden, um frühzeitig Fehler zu erkennen.



6.3.2 Integrationstests

Der Integrationstest überprüft, **ob alle Module des Systems korrekt zusammenarbeiten**. Hierbei wird getestet, ob:

- **Datenfluss:** Überprüfung, ob die Daten korrekt vom Hochladen der Bilddatei über die Texterkennung bis zur Kategorisierung und Speicherung fliessen.
- **API-Integration:** Sicherstellen, dass die Kommunikation mit der Google Cloud Vision API reibungslos funktioniert und erwartete Ergebnisse liefert.
- **Benutzeroberfläche:** Testen, ob die Ergebnisse der Verarbeitung korrekt in der Benutzeroberfläche dargestellt werden und der Benutzer den Status nachvollziehen kann.

Konzept Integrationstest:

Test-ID	Testfall	Beschreibung	Erwartetes Ergebnis
IT-01	Upload eines Ordners	Hochladen eines Ordners mit verschiedenen Bildern	Der Text wird vollständig extrahiert
IT-02	OCR-Moduleingabe	Testet, ob die Datei an das OCR-Modul weitergeleitet wird	System beginnt die Texterkennung
IT-03	Kategorisierung	Testet, ob erkannter Text korrekt kategorisiert wird	Dokument wird richtig zugeordnet
IT-04	Dateiablage	Prüft, ob verarbeitete Dateien im richtigen Ordner landen	Datei wird korrekt umbenannt und gespeichert



6.3.3 Systemtest & Fehleranalyse

Der Systemtest überprüft das **gesamte System unter realistischen Bedingungen** und testet verschiedene Szenarien:

- **Normale Nutzung:** Der Benutzer lädt Bilder hoch, die verarbeitet und kategorisiert werden.
- **Grenzfälle:** Test mit sehr grossen oder schwer lesbaren Bildern.
- **Fehlerszenarien:** Systemtest mit fehlerhaften oder ungültigen Dateien (z. B. PDFs, leere Bilder).
- **Performance-Tests:** Überprüfung, wie lange das System für die Verarbeitung von 10, 50 oder 100 Dateien benötigt.

Konzept Systemtest

Test-ID	Testfall	Beschreibung	Erwartetes Ergebnis
ST-01	Test mit unscharfen Bildern	Systemtest mit schwer lesbaren Dokumenten	System erkennt Text mit möglicher Fehlerquote
ST-02	Upload von 50 Bildern	Belastungstest mit grosser Datenmenge	System verarbeitet Dateien ohne Absturz
ST-03	Fehlerszenarien	Hochladen von PDFs oder nicht unterstützten Formaten	System blockiert Datei und gibt eine Fehlermeldung aus



6.4 Einrichtung Projektumgebung

Die korrekte Einrichtung der Projektumgebung ist wichtig, damit die Entwicklung des OCR-Systems einfach und ohne Fehler funktioniert. In diesem Abschnitt wird erklärt, wie die Entwicklungsumgebung eingerichtet wurde, welche Tools und Programme verwendet werden und wie alles für eine einfache Nutzung vorbereitet wurde.

6.4.1 Anforderungen an die Entwicklungsumgebung

Damit das System gut funktioniert, müssen folgende Anforderungen erfüllt sein:

- **Node.js und Express.js** für das Backend
- **Google Cloud Vision API** für die Texterkennung
- **USB-Stick** zur Versionierung des Codes
- **Strukturierte Ordnerverwaltung**, damit Dateien übersichtlich bleiben
- **Windows CMD als Kommandozeile** für die Einrichtung und Verwaltung

6.4.2 Installation der benötigten Software

Damit die Entwicklung starten kann, müssen einige Programme installiert werden.

6.4.2.1 Node.js und npm

Node.js wird benötigt, um das Backend laufen zu lassen.

1. Node.js herunterladen und installieren:

- <https://nodejs.org> besucht und die aktuelle Version heruntergeladen mit Standardeinstellung.

6.4.2.2 Express.js

Express.js ist ein Framework für das Backend.

1. Öffne die CMD und installiere Express:

```
npm install express --save
```

6.4.2.3 Google Cloud Vision API

1. Google Cloud SDK herunterladen:

- Auf <https://cloud.google.com/sdk> gegangen und das Installationsprogramm für Windows heruntergeladen.

2. Google Cloud in CMD initialisieren:

```
gcloud init
```



3. API-Schlüssel einrichten:

- API-Schlüssel in die Datei keyfile.json gespeichert .
- Eine .env-Datei im Projektordner und füge hinzugefügt:

```
cd "C:\Users\hpawl\Desktop\PA Hugo Pawlowski\Code\Version 1"
echo GOOGLE_APPLICATION_CREDENTIALS=C:\Users\hpawl\Desktop\PA
Hugo Pawlowski\Code\Version 1\config\secrets\keyfile.json > .env
```

```
cd "C:\Users\hpawl\Desktop\PA Hugo Pawlowski\Code\Version 1"
```

→ Wechselt in das Projektverzeichnis.

```
echo GOOGLE_APPLICATION_CREDENTIALS=... > .env
```

→ Erstellt eine .env-Datei und setzt die Umgebungsvariable für das Keyfile.

6.4.2.4 Google Cloud Storage

Für die Verwaltung von Dateien und Speicherung

```
npm install @google-cloud/storage
```

6.4.2.5 Multer

Multer wird in diesem System genutzt, um hochgeladene Dateien entgegenzunehmen und temporär im uploads/-Ordner zu speichern. Die hochgeladenen Dateien werden dann von fileService verarbeitet und mit der Google Cloud Vision API analysiert.

```
npm install Multer
```



6.4.2.6 Versionierung des Codes

Ein USB Laufwerk mit genug Speicher dient als Speicherort für die Aufbewahrung von des Codes

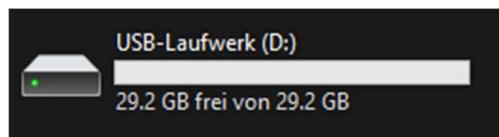


Abbildung 8: Backup Laufwerk

Ordner werden mit der Version (Tag 1.2.3..4 etc) hochgeladen.

Name	Änderungsdatum	Typ	Größe
📁 Tag1	06.03.2025 15:04	Dateiordner	
📁 Tag2	06.03.2025 15:04	Dateiordner	
📁 Tag3	06.03.2025 15:05	Dateiordner	

Den Code ist mit Version 1-8 versioniert wobei Version End der Endcode ist und man es damit auch benutzen kann.

Name	Änderungsdatum	Typ
📁 Version 1	07.03.2025 08:43	Dateiordner
📁 Version 2	10.03.2025 17:57	Dateiordner
📁 Version 3	11.03.2025 17:50	Dateiordner
📁 Version 4	12.03.2025 17:51	Dateiordner
📁 Version 5	13.03.2025 17:59	Dateiordner
📁 Version 6	14.03.2025 16:51	Dateiordner
📁 Version 7	17.03.2025 09:08	Dateiordner
📁 Version End	14.03.2025 08:13	Dateiordner

Abbildung 9:Code Versionierung



6.4.3 Einrichtung der Ordnerstruktur

```
Projekt
|   |-- config/
|   |   `-- secrets
|   |-- controllers/
|   |   |-- list/
|   |   |-- public/
|   |   |-- routes/
|   |   |-- services/
|   |   |-- templates/
|   |   |-- uploads/
|   |   |-- utils/
|   |   `-- package.json
|   |-- packages/
|   |   |-- server.js/
|   |   `-- tests/
|   `-- .env
```



7 Phase 3: Realisierungsphase

7.1 Implementierung der Benutzeroberfläche

Die Benutzeroberfläche (GUI) ist ein kern Bestandteil des OCR-Dokumentenmanagementsystems. Sie richtet es dem Benutzer ein, Dokumente hochzuladen, zu verarbeiten und die Ergebnisse der Texterkennung und Kategorisierung anzuzeigen. Die GUI wurde mit **HTML, CSS und JavaScript** entwickelt.

1. Ziel der Benutzeroberfläche

- **Einfache Bedienung:** Klare Struktur und Navigation.
- **Dokumentenverwaltung:** Möglichkeit zum Hochladen und Verarbeiten von Bildern.
- **Texterkennung & Kategorisierung anzeigen:** Ergebnisse der OCR-Verarbeitung in einer Tabelle darstellen.
- **Fehlermanagement:** Anzeige von Warnungen bei nicht unterstützten Dateiformaten oder fehlerhaften Dokumenten.

2. Aufbau der Benutzeroberfläche

Die Benutzeroberfläche besteht aus mehreren Komponenten:

- **Startansicht („Custom Folder Processing“):**
 - Button „Choose Folder“ → Ermöglicht das Auswählen eines Ordners mit Dokumenten.
 - Button „Evaluate“ → Startet die Texterkennung und Kategorisierung.
- **Popup zur Ordnerauswahl:**
 - Textfeld zur Eingabe des Pfads.
 - „OK“-Button bestätigt die Auswahl.
 - „Cancel“-Button bricht den Vorgang ab.



```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>OCR Folder Processing</title>
7      <link rel="stylesheet" href="styles.css">
8      <script src="script.js" defer></script>
9  </head>
10 <body>
11 <main>
12     <!-- Folder Processing Section -->
13     <section class="folder-section">
14         <h1>Custom Folder Processing</h1>
15         <form id="folderForm">
16             <div class="form-group">
17                 <h2>Select Folder</h2>
18                 <input type="hidden" id="FolderPath" name="FolderPath">
19                 <button type="button" onclick="selectFolder()">Choose Folder</button>
20             </div>
21             <div class="action-group">
22                 <button type="submit" class="primary">Evaluate</button>
23             </div>
24         </form>
25     </section>
26 </main>
27 </body>
28 </html>
```

Abbildung 10: Startseite in Code

- **Ergebnisanzeige nach der Analyse:**

- Tabelle mit den hochgeladenen Dateien.
- Spalten für erkannte Kategorien, extrahierten Text und Verarbeitungszeit.
- Button „Show More“ zeigt den vollständigen erkannten Text an.

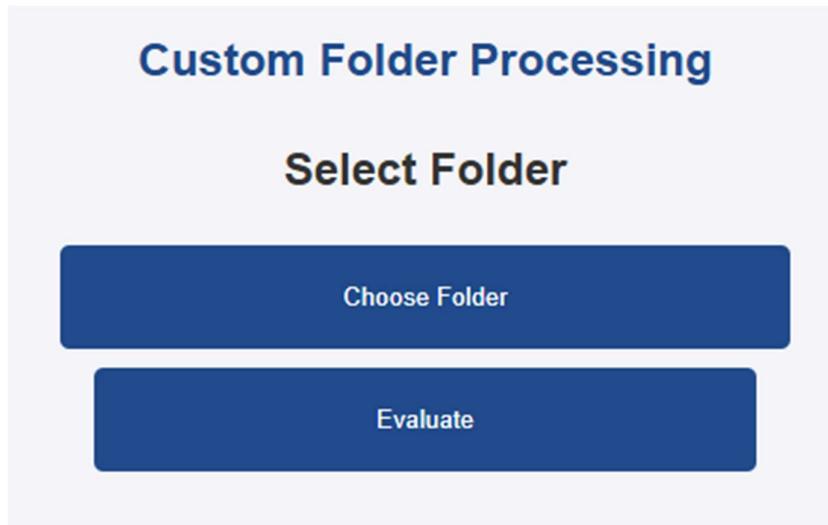
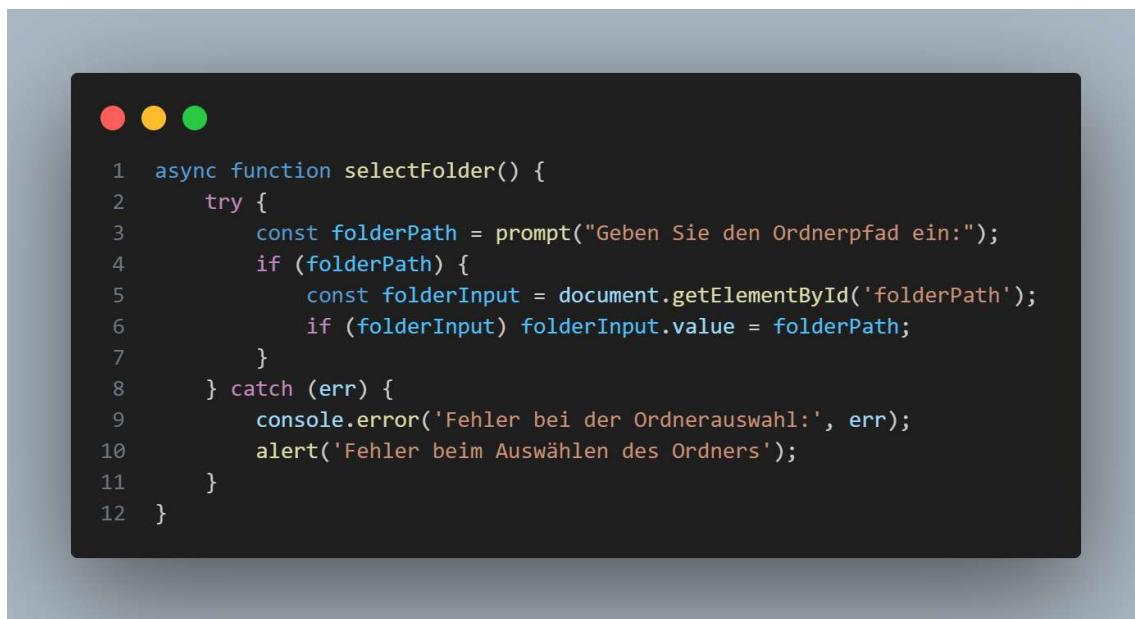


Abbildung 11: Startseite UI



```
1 async function selectFolder() {
2     try {
3         const folderPath = prompt("Geben Sie den Ordnerpfad ein:");
4         if (FolderPath) {
5             const folderInput = document.getElementById('FolderPath');
6             if (folderInput) folderInput.value = folderPath;
7         }
8     } catch (err) {
9         console.error('Fehler bei der Ordnerauswahl:', err);
10        alert('Fehler beim Auswählen des Ordners');
11    }
12 }
```

Abbildung 12: Methode zur Ordnerauswahl in der Benutzeroberfläche

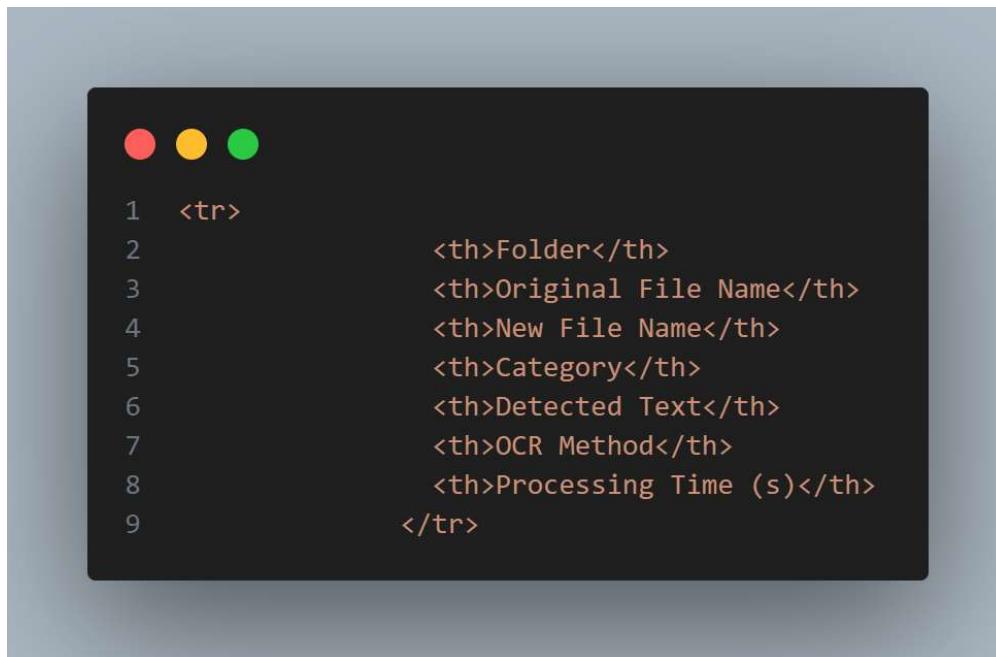
selectFolder() dient dazu, dem Benutzer die Möglichkeit zu geben, einen Ordner auf seinem System auszuwählen. Sie unterstützt zwei verschiedene Umgebungen:



```
1 require('dotenv').config();
2 const express = require('express');
3 const path = require('path');
4
5 const app = express();
6 const PORT = process.env.PORT || 3000;
7
8 app.use(express.static(path.join(__dirname, 'public')));
9
10 app.get('/', (req, res) => {
11   res.sendFile(path.join(__dirname, 'public', 'index.html'));
12 });
13
14 app.listen(PORT);
15
```

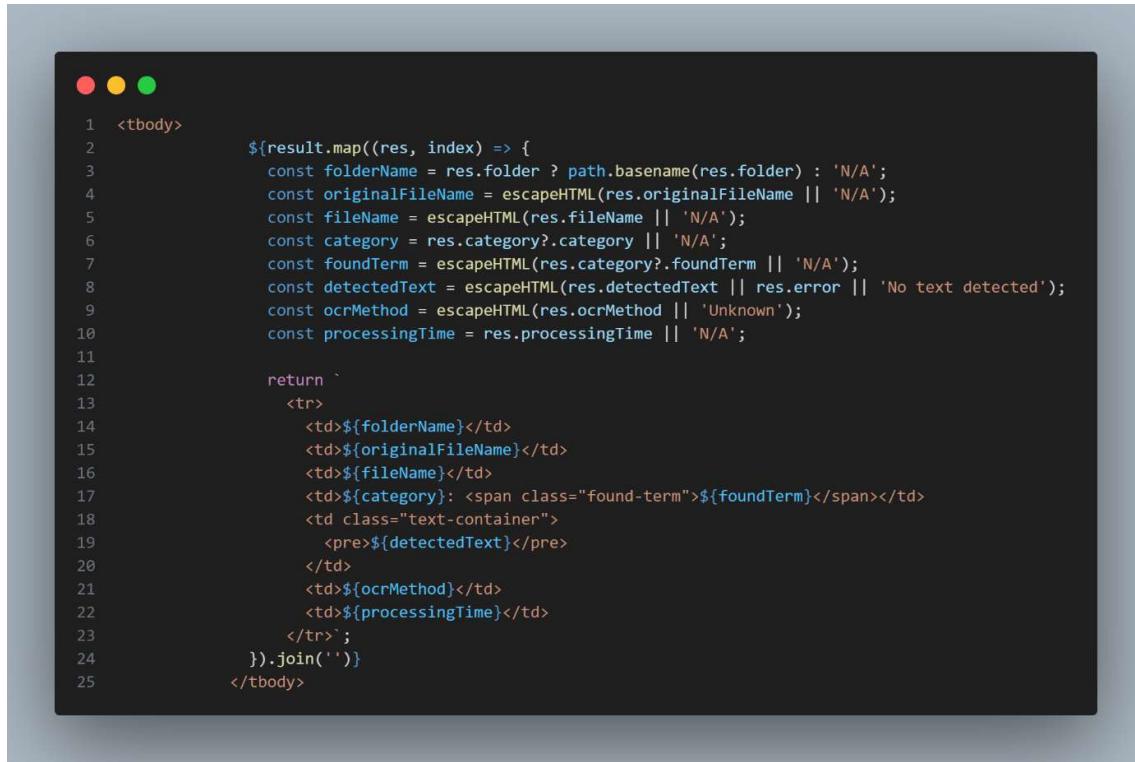
Abbildung 13: Standard-Serverkonfiguration

Diese Methode startet einen einfachen **Webserver** mit **Express.js**, die Dateien bereitstellt und eine HTML-Seite an den Client sendet.



```
1 <tr>
2   <th>Folder</th>
3   <th>Original File Name</th>
4   <th>New File Name</th>
5   <th>Category</th>
6   <th>Detected Text</th>
7   <th>OCR Method</th>
8   <th>Processing Time (s)</th>
9 </tr>
```

Abbildung 14: ResultTemplate Tabelle



```
1  <tbody>
2      ${result.map((res, index) => {
3          const folderName = res.folder ? path.basename(res.folder) : 'N/A';
4          const originalFileName = escapeHTML(res.originalFileName || 'N/A');
5          const fileName = escapeHTML(res.fileName || 'N/A');
6          const category = res.category?.category || 'N/A';
7          const foundTerm = escapeHTML(res.category?.foundTerm || 'N/A');
8          const detectedText = escapeHTML(res.detectedText || res.error || 'No text detected');
9          const ocrMethod = escapeHTML(res.ocrMethod || 'Unknown');
10         const processingTime = res.processingTime || 'N/A';
11
12         return `
13             <tr>
14                 <td>${folderName}</td>
15                 <td>${originalFileName}</td>
16                 <td>${fileName}</td>
17                 <td>${category}: <span class="found-term">${foundTerm}</span></td>
18                 <td class="text-container">
19                     <pre>${detectedText}</pre>
20                 </td>
21                 <td>${ocrMethod}</td>
22                 <td>${processingTime}</td>
23             </tr>;
24     }).join('')
25  </tbody>
```

Abbildung 15:Dynamische Generierung einer Ergebnistabelle

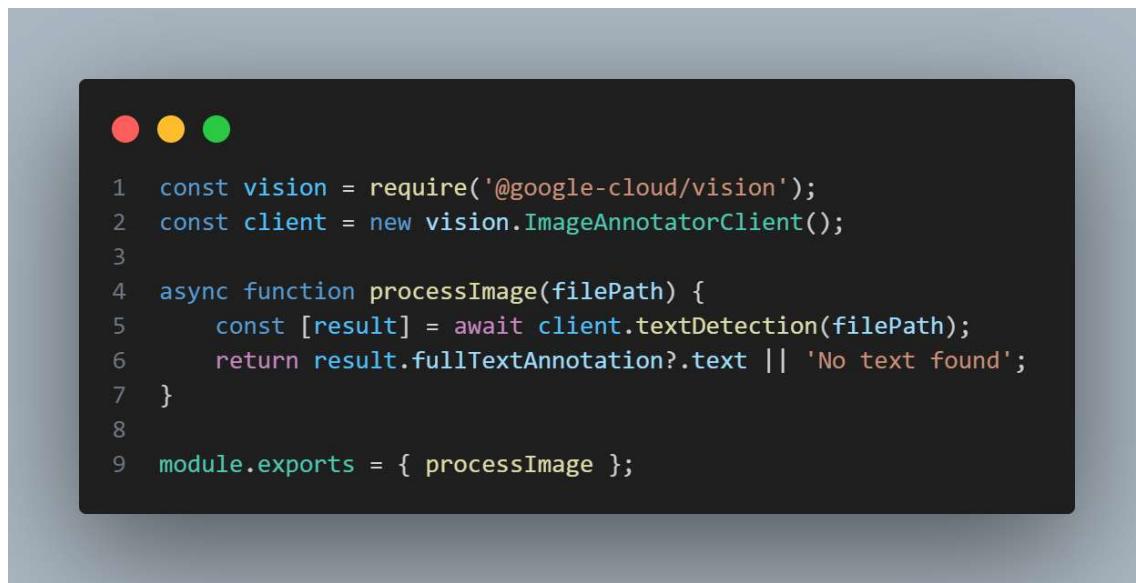
Die Funktion iteriert über die **Erkennungsergebnisse**, extrahiert relevante Informationen und generiert eine **HTML-Tabelle** mit den erkannten Texten, Kategorien und Verarbeitungszeiten. Falls keine Daten verfügbar sind, werden Standardwerte gesetzt.



7.2 OCR-Integration

In diesem Abschnitt wird die Integration der **OCR (Optical Character Recognition)**-Technologie in das System beschrieben. Ziel ist es, Texte aus Dokumenten zu extrahieren und diese automatisch zu kategorisieren.

7.2.1 Implementierung der OCR-Verarbeitung



```
● ● ●

1 const vision = require('@google-cloud/vision');
2 const client = new vision.ImageAnnotatorClient();
3
4 async function processImage(filePath) {
5     const [result] = await client.textDetection(filePath);
6     return result.fullTextAnnotation?.text || 'No text found';
7 }
8
9 module.exports = { processImage };
```

Abbildung 16: Google Cloud Implementation

Ich habe eine Funktion erstellt, die mit der **Google Cloud Vision API** Texte aus Bildern extrahiert. Sie nimmt einen Dateipfad entgegen, führt eine **Texterkennung** durch und gibt den erkannten Text oder eine Fehlermeldung zurück.



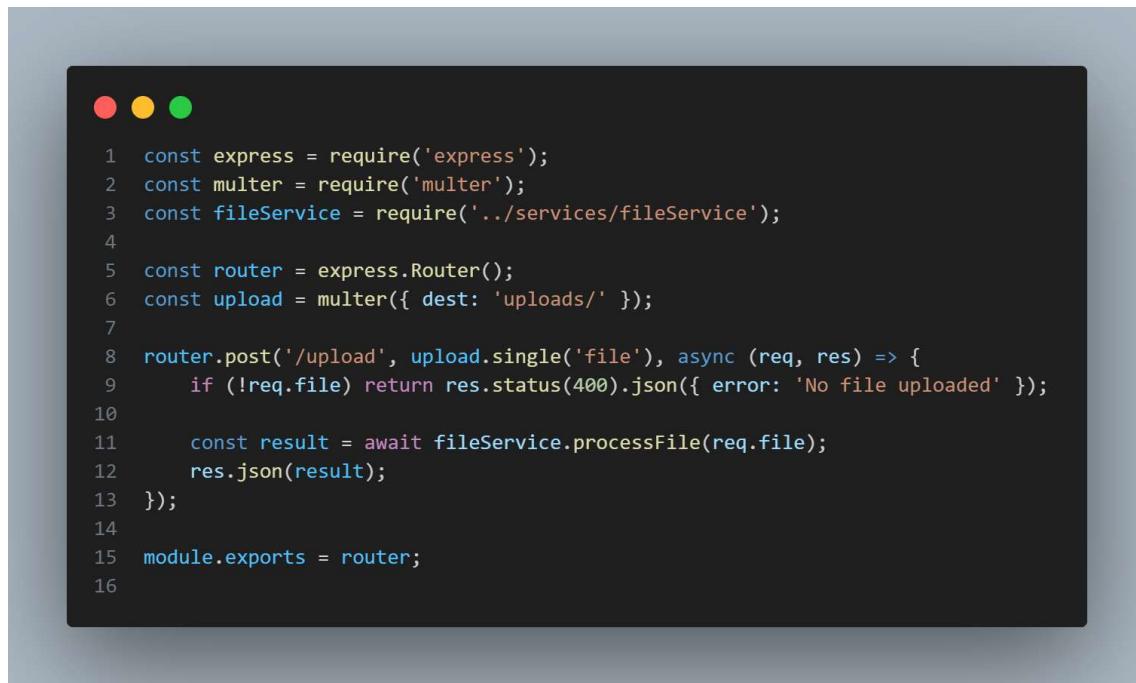
```
● ● ●

1 const ocrService = require('./ocrService');
2
3 async function processFile(file) {
4     const text = await ocrService.processImage(file.path);
5     return { fileName: file.originalname, detectedText: text };
6 }
7
8 module.exports = { processFile };
```

Abbildung 17: Texterkennung in Dateien



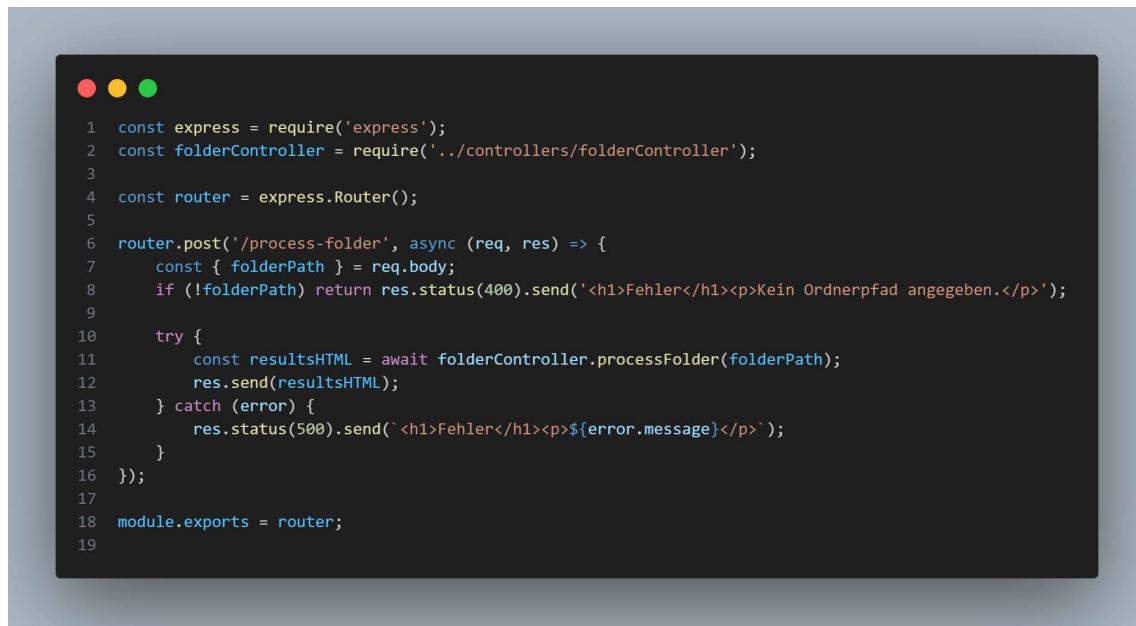
Die Funktion nimmt eine Datei entgegen, ruft den **OCR-Service** auf und gibt den erkannten Text zusammen mit dem Dateinamen zurück. Dies ermöglicht die automatische Texterkennung für hochgeladene Dateien.



```
1 const express = require('express');
2 const multer = require('multer');
3 const fileService = require('../services/fileService');
4
5 const router = express.Router();
6 const upload = multer({ dest: 'uploads/' });
7
8 router.post('/upload', upload.single('file'), async (req, res) => {
9     if (!req.file) return res.status(400).json({ error: 'No file uploaded' });
10
11     const result = await fileService.processFile(req.file);
12     res.json(result);
13 });
14
15 module.exports = router;
16
```

Abbildung 18: API-Route zur Verarbeitung von Datei-Uplads

Die Funktion stellt eine **API-Route** bereit, die Dateien entgegennimmt, speichert und zur **Texterkennung** an den **fileService** weiterleitet. Falls keine Datei hochgeladen wurde, wird eine Fehlermeldung zurückgegeben.

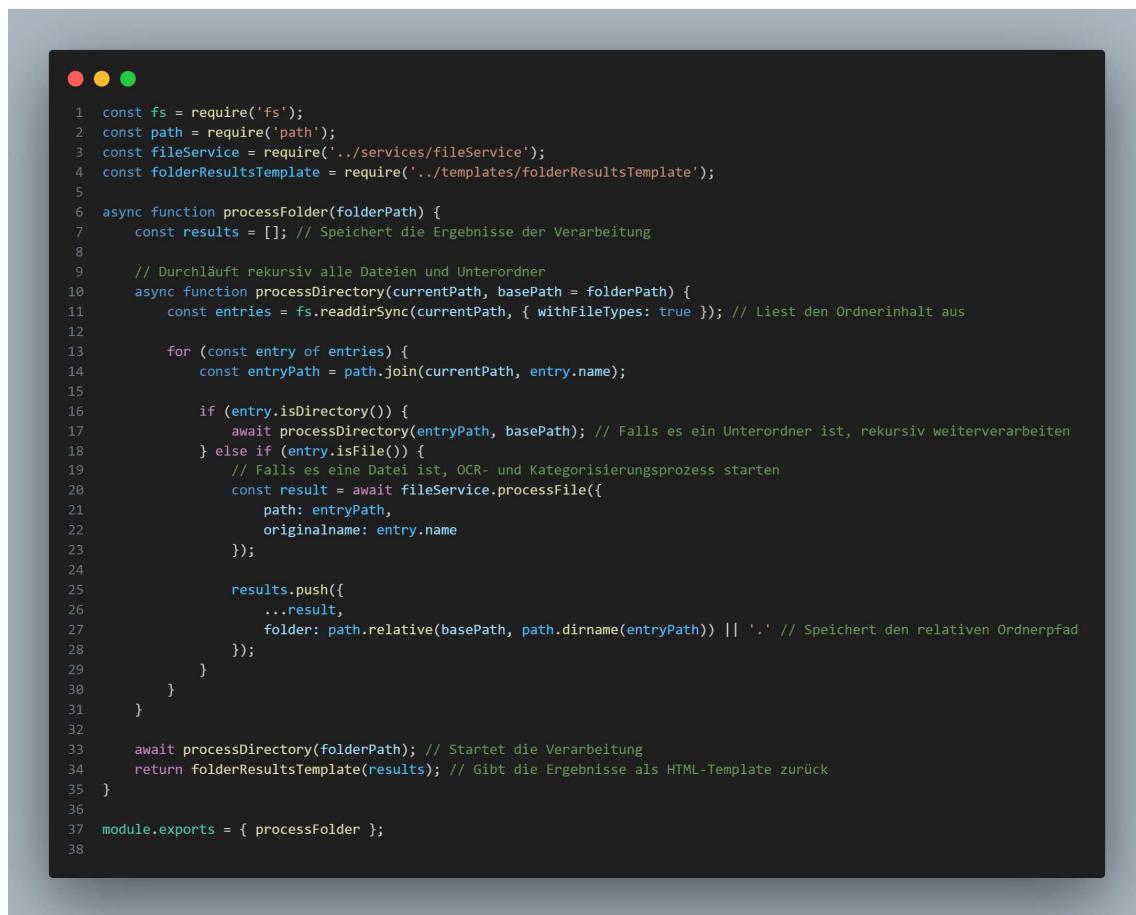


```
1 const express = require('express');
2 const folderController = require('../controllers/folderController');
3
4 const router = express.Router();
5
6 router.post('/process-folder', async (req, res) => {
7     const { folderPath } = req.body;
8     if (!folderPath) return res.status(400).send(`<h1>Fehler</h1><p>Kein Ordnerpfad angegeben.</p>`);
9
10    try {
11        const resultsHTML = await folderController.processFolder(folderPath);
12        res.send(resultsHTML);
13    } catch (error) {
14        res.status(500).send(`<h1>Fehler</h1><p>${error.message}</p>`);
15    }
16 });
17
18 module.exports = router;
19
```

Abbildung 19: API-Route zur Verarbeitung von Ordnerpfaden



Die Funktion empfängt einen **Ordnerpfad** als POST-Anfrage, validiert ihn und leitet ihn an den **folderController** zur Verarbeitung weiter. Falls kein Pfad angegeben oder ein Fehler auftritt, wird eine entsprechende Fehlermeldung zurückgegeben.



```
1 const fs = require('fs');
2 const path = require('path');
3 const fileService = require('../services/fileService');
4 const folderResultsTemplate = require('../templates/folderResultsTemplate');

5
6 async function processFolder(folderPath) {
7     const results = [] // Speichert die Ergebnisse der Verarbeitung
8
9     // Durchläuft rekursiv alle Dateien und Unterordner
10    async function processDirectory(currentPath, basePath = folderPath) {
11        const entries = fs.readdirSync(currentPath, { withFileTypes: true }); // Liest den Ordnerinhalt aus
12
13        for (const entry of entries) {
14            const entryPath = path.join(currentPath, entry.name);
15
16            if (entry.isDirectory()) {
17                await processDirectory(entryPath, basePath); // Falls es ein Unterordner ist, rekursiv weiterverarbeiten
18            } else if (entry.isFile()) {
19                // Falls es eine Datei ist, OCR- und Kategorisierungsprozess starten
20                const result = await fileService.processFile({
21                    path: entryPath,
22                    originalname: entry.name
23                });
24
25                results.push({
26                    ...result,
27                    folder: path.relative(basePath, path.dirname(entryPath)) || '.' // Speichert den relativen Ordnerpfad
28                });
29            }
30        }
31    }
32
33    await processDirectory(folderPath); // Startet die Verarbeitung
34    return folderResultsTemplate(results); // Gibt die Ergebnisse als HTML-Template zurück
35 }
36
37 module.exports = { processFolder };
38
```

Abbildung 20:Funktion zur Verarbeitung eines gesamten Ordners

Die Funktion liest alle **Dateien in einem Ordner** aus, überprüft, ob es sich um gültige Dateien handelt, und übergibt sie an den **fileService** zur Verarbeitung. Die Ergebnisse werden gesammelt und mit einem **Template** formatiert zurückgegeben.

7.3 Kategorisierung

Die Kategorisierung der Dokumente erfolgt in mehreren Schritten. Ziel ist es, anhand des erkannten Textes eine Zuordnung zu einer bestimmten Kategorie vorzunehmen und entsprechende Metadaten für die Weiterverarbeitung bereitzustellen.



7.3.1 Funktionsweise der Kategorisierung

Die Kategorisierung basiert auf einer Kombination aus Texterkennung (OCR) und einer vordefinierten Liste von Schlüsselwörtern. Das System analysiert den extrahierten Text und vergleicht ihn mit einer Liste von Kategorien, um die passende Zuordnung vorzunehmen.

Ablauf der Kategorisierung:

1. **Texterkennung:** Die hochgeladenen Dokumente werden mittels der Google Cloud Vision API analysiert. Der extrahierte Text wird in einer Zeichenkette gespeichert.
2. **Vergleich mit vordefinierten Kategorien:** Der erkannte Text wird mit einer vordefinierten Liste von Schlüsselwörtern verglichen. Diese Liste umfasst Begriffe aus den Kategorien "Bank", "Krankenkasse", "Lohnausweis" und "Kredit".
3. **Zuordnung der Kategorie:** Falls ein oder mehrere Schlüsselwörter erkannt werden, wird die Datei in die entsprechende Kategorie eingeordnet.
4. **Speicherung der Ergebnisse:** Die kategorisierten Dateien werden in einem strukturierten Format gespeichert, entweder durch Umbenennung oder durch Ablage in spezifischen Ordnern

7.3.2 Implementierung der Kategorisierung

Definition der Kategorien

Die Liste der Kategorien wird in liste.js definiert



```
 1 module.exports = {
 2   bankenListe: ["UBS", "Credit Suisse", "Raiffeisen", "ZKB"],
 3   krankenkassenListe: ["CSS", "Helsana", "Sanitas"],
 4   lohnAusweisListe: ["Lohnausweis"],
 5   kreditListe: ["Corner", "kredit"],
 6   SAC: [ "Corner", "cornercard", "Cornèr", "kredit" ]
 7 };
```

Abbildung 21: Kategorien

Vergleich des Textes mit den Kategorien

Die detectCategory()-Funktion vergleicht den erkannten Text mit den vordefinierten Kategorien:



```
1 const {
2     bankenListe,
3     krankenkassenListe,
4     lohnAusweisListe,
5     kreditliste
6 } = require('../lists/liste');
7
8 exports.detectCategory = (text) => {
9     if (!text) return { category: "Uncategorized", foundTerm: "No text provided" };
10
11    const lowerText = text.toLowerCase();
12
13    // Bank match
14    const bankMatch = bankenListe.find(bank => lowerText.includes(bank.toLowerCase()));
15    if (bankMatch) return { category: "Bank", foundTerm: bankMatch };
16
17    // Health insurance match
18    const healthInsuranceMatch = krankenkassenListe.find(kk => lowerText.includes(kk.toLowerCase()));
19    if (healthInsuranceMatch) return { category: "HealthInsurance", foundTerm: healthInsuranceMatch };
20
21    // Payroll statement
22    if (lohnAusweisListe.some(lohn => lowerText.includes(lohn.toLowerCase()))) {
23        return { category: "Payroll", foundTerm: "Payroll statement detected" };
24    }
25
26    // Credit
27    if (kreditliste.some(kredit => lowerText.includes(kredit.toLowerCase()))) {
28        return { category: "Credit", foundTerm: "Credit related terms found" };
29    }
30
31    return { category: "Uncategorized", foundTerm: "No category detected" };
32
33 };
34
```

Abbildung 22: Vergleich des Textes mit den Kategorien

Einbindung der Kategorisierung in die Datei-Verarbeitung

Die Kategorisierung wird in fileService.js integriert, indem die detectCategory()-Funktion nach der OCR-Erkennung aufgerufen wird:



```
1 const ocrService = require('./ocrService');
2 const categoryService = require('./categoryService');
3 const { performance } = require('perf_hooks');
4 const mime = require('mime-types'); // Import für MIME-Typ-Erkennung
5
6 async function processFile(file) {
7     const startTime = performance.now(); // Startzeit erfassen
8
9     // MIME-Typ bestimmen, falls nicht vorhanden
10    const mimeType = mime.lookup(file.path) || 'application/octet-stream';
11
12    // OCR-Texterkennung ausführen
13    const ocrResult = await ocrService.processFile(file.path, mimeType);
14    const endTime = performance.now(); // Endzeit erfassen
15
16    // Kategorisierung des erkannten Textes
17    const category = ocrResult.detectedText
18        ? categoryService.detectCategory(ocrResult.detectedText)
19        : { category: "Uncategorized", foundTerm: "No text detected" };
20
21    return {
22        fileName: file.originalname, // Originalname beibehalten
23        originalFileName: file.originalname, // Originalname für die Ausgabe beibehalten
24        detectedText: ocrResult.detectedText,
25        ocrMethod: 'Google Cloud Vision API', // OCR Methode explizit setzen
26        category,
27        processingTime: ((endTime - startTime) / 1000).toFixed(2) // Zeit in Sekunden
28    };
29 }
30
31 module.exports = { processFile };

```

Abbildung 23: Implementierung der Dateiverarbeitung

7.4 Datei Umbenennen/Sortierung

Die Funktion zum Umbenennen und Sortieren der Dateien ermöglicht es, Dateien sowohl automatisch als auch manuell umzubenennen.

Automatische Umbenennung

Die automatische Umbenennung erfolgt direkt nach der Texterkennung und Kategorisierung der Dokumente. Dateien, deren Namen den Begriff „decrypted“ enthalten, werden automatisch umbenannt. Hierbei wird der Dateiname entsprechend der erkannten Kategorie angepasst und fortlaufend nummeriert. Die Implementierung erfolgt in der Datei fileService.js:



```
1  async function renameFileAutomatically(file) {
2      try {
3          const folderPath = path.dirname(file.path);
4          const fileExtension = path.extname(file.path);
5
6          if (!file.category || file.category.category === "Uncategorized") {
7              return path.basename(file.path);
8          }
9
10         const categoryKey = file.category.category.toLowerCase();
11         if (!counters[categoryKey]) {
12             counters[categoryKey] = 1;
13         }
14
15         const baseName = `${file.category.category}${counters[categoryKey]}`;
16         const newFileName = await getUniqueFileName(folderPath, baseName, fileExtension);
17
18         await fs.rename(file.path, path.join(folderPath, newFileName));
19
20         counters[categoryKey]++;
21
22         return newFileName;
23     } catch (error) {
24         console.error(`Fehler beim automatischen Umbenennen: ${error.message}`);
25         return path.basename(file.path);
26     }
27 }
```

Abbildung 24: Automatische Umbenennung

Manuelle Umbenennung

Benutzer können Dateien auch manuell über die Benutzeroberfläche umbenennen. Diese Funktionalität ist besonders nützlich, falls die automatische Benennung nicht passend ist. Die manuelle Umbenennung wird über eine Schaltfläche "Rename" in der Ergebnistabelle ausgelöst.

Die API-Route für die manuelle Umbenennung ist in fileRoutes.js definiert:

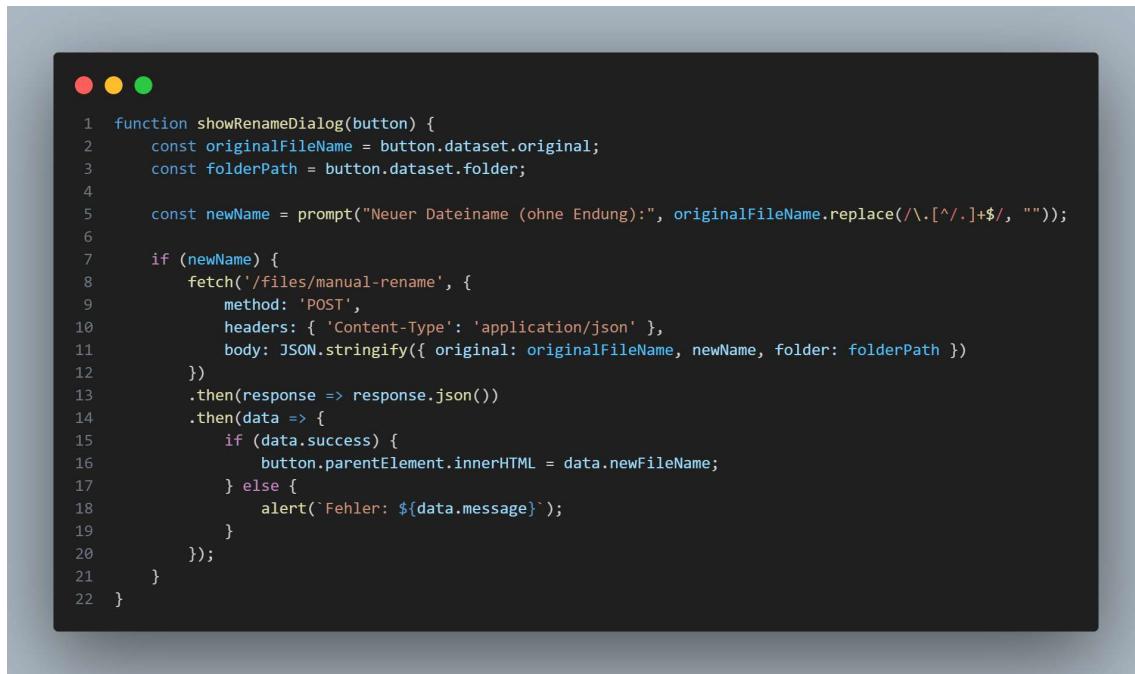


```
1 router.post('/manual-rename', async (req, res) => {
2     const { original, newName, folder } = req.body;
3
4     if (!original || !newName || !folder) {
5         return res.status(400).json({ success: false, message: "Ungültige Parameter" });
6     }
7
8     const absoluteFolderPath = path.resolve(folder);
9
10    const result = await fileService.renameFile(absoluteFolderPath, original, newName);
11
12    if (result.success) {
13        return res.json(result);
14    } else {
15        return res.status(400).json(result);
16    }
17});
```

Abbildung 25: Manuelle Umbenennung

Interaktive Umbenennung (Frontend)

Diese Funktion aktualisiert den neuen Dateinamen in der bestehenden Tabelle, ohne die Seite komplett neu laden zu müssen.



```
1 function showRenameDialog(button) {
2     const originalFileName = button.dataset.original;
3     const folderPath = button.dataset.folder;
4
5     const newName = prompt("Neuer Dateiname (ohne Endung):", originalFileName.replace(/^.+[.]+$/, ""));
6
7     if (newName) {
8         fetch('/files/manual-rename', {
9             method: 'POST',
10             headers: { 'Content-Type': 'application/json' },
11             body: JSON.stringify({ original: originalFileName, newName, folder: folderPath })
12         })
13         .then(response => response.json())
14         .then(data => {
15             if (data.success) {
16                 button.parentElement.innerHTML = data.newFileName;
17             } else {
18                 alert(`Fehler: ${data.message}`);
19             }
20         });
21     }
22 }
```

Abbildung 26: Interaktive Umbenennung



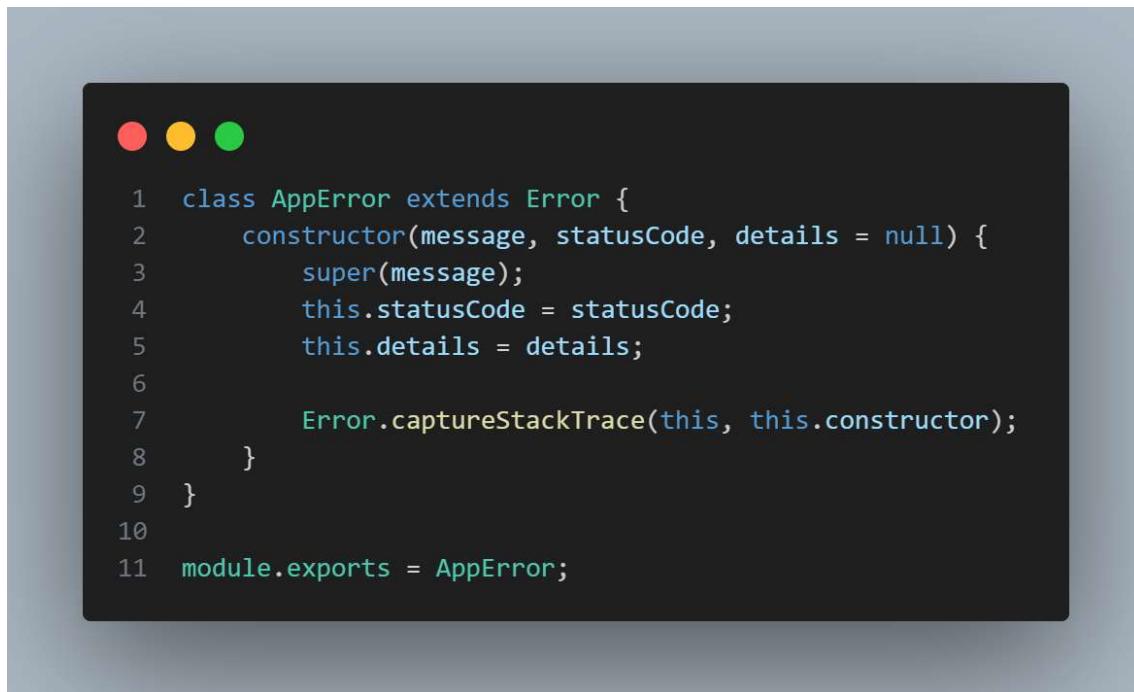
7.5 Fehlerbehandlung

Damit das OCR-System zuverlässig und benutzerfreundlich funktioniert, ist eine Fehlerbehandlung wichtig. Fehler können in verschiedenen Bereichen auftreten, z. B. bei der Texterkennung, Kategorisierung oder Dateiverarbeitung. Um Probleme schnell zu erkennen und zu beheben, wird ein Fehlerhandling eingesetzt.

Dazu wird eine spezifische Fehlerklasse, die „AppError“-Klasse verwendet, welche die Standard-Fehlerklasse von JavaScript erweitert. Diese Klasse erlaubt es, anwendungsbezogene Fehler anzugeben und zu behandeln und den Benutzern hilfreiche Fehlermeldungen bereitzustellen.

Die Implementierung der „AppError“-Klasse erfolgt mit folgenden Merkmalen:

- **Fehlermeldung (message):** Klare und verständliche Fehlermeldung für den Benutzer.
- **HTTP-Statuscode (statusCode):** Zuordnung eines spezifischen HTTP-Statuscodes zur eindeutigen Klassifizierung des Fehlertyps.
- **Details (optional):** Weitere Informationen zum Fehler, um eine präzisere Analyse und Behebung zu ermöglichen.



```
 1  class AppError extends Error {
 2      constructor(message, statusCode, details = null) {
 3          super(message);
 4          this.statusCode = statusCode;
 5          this.details = details;
 6
 7          Error.captureStackTrace(this, this.constructor);
 8      }
 9  }
10
11 module.exports = AppError;
```

Abbildung 27:AppError-Klasse

Fehlerbehandlung im Workflow des OCR-Systems:

1. **Dateiformat- und Texterkennungsfehler:**



- Falls eine Datei ein nicht unterstütztes Format besitzt (z.B. PDF oder DOCX) oder keinen lesbaren Text enthält, wird eine spezifische Fehlermeldung generiert und dem Benutzer in der GUI angezeigt.

2. Kommunikationsfehler mit Google Cloud Vision API:

- Sollte die API nicht erreichbar sein oder einen Fehler zurückgeben, wird eine AppError-Instanz mit einem entsprechenden Statuscode erzeugt, der Benutzer wird entsprechend informiert, und der Fehler im Protokoll erfasst.

3. Fehler beim Speichern und Umbenennen von Dateien:

- Treten Fehler bei der automatischen Umbenennung oder Ablage der Dateien auf, informiert das System den Benutzer, und die Datei wird zur manuellen Prüfung in einen separaten Ordner gelegt.



8 Phase 4: Abschlussphase

In der Abschlussphase meines Projekts liegt der Fokus auf der abschliessenden Validierung und Qualitätssicherung der entwickelten Anwendung. Ziel dieser Phase ist es, sicherzustellen, dass das System alle definierten Anforderungen erfüllt und verlässlich im produktiven Einsatz funktioniert. Dabei werden sämtliche Komponenten des Systems nochmals kritisch geprüft und getestet. Zusätzlich erfolgt in dieser Phase die abschliessende Dokumentation des Projekts, einschliesslich der Auswertung der Testergebnisse sowie der Vorbereitung auf die endgültige Projektabgabe.

8.1 Units Test

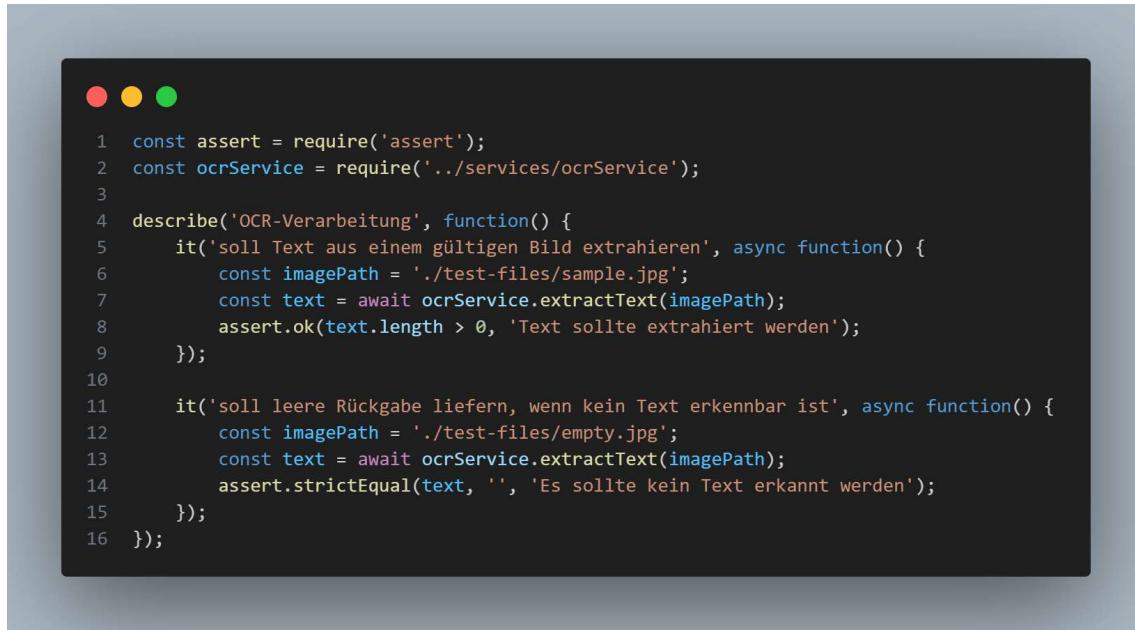
Unit-Tests helfen dabei, die Qualität der Software sicherzustellen. Sie testen einzelne Funktionen und Bausteine des Programms getrennt, um zu prüfen, ob sie richtig arbeiten. In meinem Projekt habe ich Unit-Tests verwendet, um die OCR-Texterkennung, die Kategorisierung von Dokumenten und die Umbenennung von Dateien zu überprüfen.

- Die Texterkennung für verschiedene Bildformate korrekt funktioniert.
- Die erkannte Textkategorisierung die richtigen Kategorien zuweist.
- Dateien nach der Verarbeitung korrekt umbenannt und gespeichert werden.
- Fehlerfälle wie ungültige Dateiformate oder fehlende Berechtigungen korrekt behandelt werden.

Test-ID	Testfall	Beschreibung	Erwartetes Ergebnis	Testergebnis
UT-01	OCR-Textverarbeitung	Testet, ob ein klares Bild korrekt erkannt wird	Der Text wird vollständig extrahiert	<input checked="" type="checkbox"/> Erfolgreich
UT-02	Unleserliches Bild	Testet, ob verschwommene Bilder Fehler verursachen	System meldet „Kein Text erkannt“	<input checked="" type="checkbox"/> Erfolgreich
UT-03	Kategorisierung mit Schlüsselwörtern	Prüft, ob Text korrekt in Kategorien eingeordnet wird	Datei wird richtig klassifiziert	<input checked="" type="checkbox"/> Erfolgreich
UT-04	Datei-Speicherung	Überprüft, ob Dateien nach der Verarbeitung gespeichert werden	Datei ist im korrekten Ordner zu finden	<input checked="" type="checkbox"/> Erfolgreich
UT-05	Fehlerhafte Datei	Test mit nicht unterstützten Dateiformaten (PDF, DOCX)	System lehnt Datei ab und gibt eine Fehlermeldung aus	<input checked="" type="checkbox"/> Erfolgreich
UT-06	Manuelle Umbenennung	Testet, ob eine Datei über das UI korrekt umbenannt wird	Neuer Dateiname wird in der Tabelle und im System aktualisiert	<input checked="" type="checkbox"/> Erfolgreich
UT-07	API	Testet was passiert, wenn die API nicht funktioniert	HTTP-Status 400	<input checked="" type="checkbox"/> Erfolgreich



Code für einen Unit-Test



```
1 const assert = require('assert');
2 const ocrService = require('../services/ocrService');
3
4 describe('OCR-Verarbeitung', function() {
5     it('soll Text aus einem gültigen Bild extrahieren', async function() {
6         const imagePath = './test-files/sample.jpg';
7         const text = await ocrService.extractText(imagePath);
8         assert.ok(text.length > 0, 'Text sollte extrahiert werden');
9     });
10    it('soll leere Rückgabe liefern, wenn kein Text erkennbar ist', async function() {
11        const imagePath = './test-files/empty.jpg';
12        const text = await ocrService.extractText(imagePath);
13        assert.strictEqual(text, '', 'Es sollte kein Text erkannt werden');
14    });
15 });
16 
```

Abbildung 28:Beispiel Unit-Test Code

Test 1: Falls ein lesbare Bild übergeben wird, soll der extrahierte Text eine Länge > 0 haben.

Test 2: Falls ein leeres Bild übergeben wird, sollte der erkannte Text leer sein.



8.2 Integrationstests

Während **Unit-Tests** einzelne Funktionen oder Module testen, prüfen **Integrationstests**, ob mehrere Module korrekt zusammenarbeiten.

Test-ID	Testfall	Erwartetes Ergebnis	Tatsächliches Ergebnis	Status
IT-01	OCR-Analyse & Kategorisierung	Bilder werden korrekt analysiert, erkannter Text wird angezeigt	OCR hat Text aus Rechnungen extrahiert und korrekt kategorisiert	<input checked="" type="checkbox"/> Erfolgreich
IT-02	Hochladen & Verarbeitung mehrerer Dateien	Unterstützte Dateien werden verarbeitet, fehlerhafte Dateien abgelehnt	PDF wurde abgelehnt, Bilder wurden korrekt verarbeitet	<input checked="" type="checkbox"/> Erfolgreich
IT-03	Dateispeicherung nach Verarbeitung	Dateien werden umbenannt und im richtigen Ordner gespeichert	Dateien wurden in den richtigen Ordner gespeichert	<input checked="" type="checkbox"/> Erfolgreich
IT-04	Manuelle Umbenennung & Speicherung	Dateiname ändert sich in UI und im Ordner	Datei wurde sowohl in der UI als auch im Ordner umbenannt	<input checked="" type="checkbox"/> Erfolgreich
IT-05	Verarbeitung grosser Dateien (5MB+)	Datei wird verarbeitet, ohne das System zu verlangsamen	System hat grosse Bilder erkannt und verarbeitet	<input checked="" type="checkbox"/> Erfolgreich
IT-06	Verarbeitung eines leeren Ordners	System zeigt eine Meldung an, dass keine Dateien vorhanden sind	Meldung „Keine Dateien gefunden“ wurde korrekt angezeigt	<input checked="" type="checkbox"/> Erfolgreich



8.3 Systemtest & Fehleranalyse

Der Systemtest dient dazu, dass entwickelte OCR-System in seiner Gesamtheit unter realistischen Bedingungen zu prüfen. Dabei wird überprüft, ob die Anwendung die definierten Anforderungen erfüllt und in der Ajooda Umgebung arbeitet.

Test ID	Testfall	Beschreibung	Erwartetes Ergebnis	Testergebnis
ST-01	Verarbeitung von 50 Dateien	Belastungstest durch Hochladen von 50 Dateien	System verarbeitet Dateien ohne Absturz	<input checked="" type="checkbox"/> Erfolgreich
ST-02	Test mit unscharfen Bildern	Hochladen von Dokumenten mit schlechter Bildqualität	Text wird erkannt	<input checked="" type="checkbox"/> Erfolgreich
ST-03	Fehlerszenario Dateiformate	Hochladen von nicht unterstützten Formaten (z. B. PDF, TxT)	Dateien werden abgelehnt + Fehlermeldung	<input checked="" type="checkbox"/> Erfolgreich
ST-04	Geschwindigkeitstest	Messung der Verarbeitungsgeschwindigkeit von 100 Dateien	System verarbeitet alle Dateien in angemessener Zeit	<input checked="" type="checkbox"/> Erfolgreich, dazu den VF nach seiner Meinung gefragt und er fand es Gut.

8.4 Fehlerkorrektur und Code Optimierung

Während der Entwicklung des Projekts wurden kleinere Code-Optimierungen vorgenommen, um die Lesbarkeit und Effizienz zu verbessern. Eine der Änderungen ist die Überprüfung, ob eine Datei existiert.

Problem:

Die ursprüngliche Funktion fileExists nutzte try-catch, um zu überprüfen, ob eine Datei existiert. Dies war funktional, aber unnötig lang.

Vorher:



```
1  async function fileExists(filePath) {
2      try {
3          await fs.access(filePath);
4          return true;
5      } catch {
6          return false;
7      }
8  }
9
```

Abbildung 29:Vorherige Implementierung der Datei-Existenzprüfung

Lösung:

Die Funktion wurde in eine kompaktere Arrow-Funktion umgewandelt, die mit .then() und .catch() arbeitet.

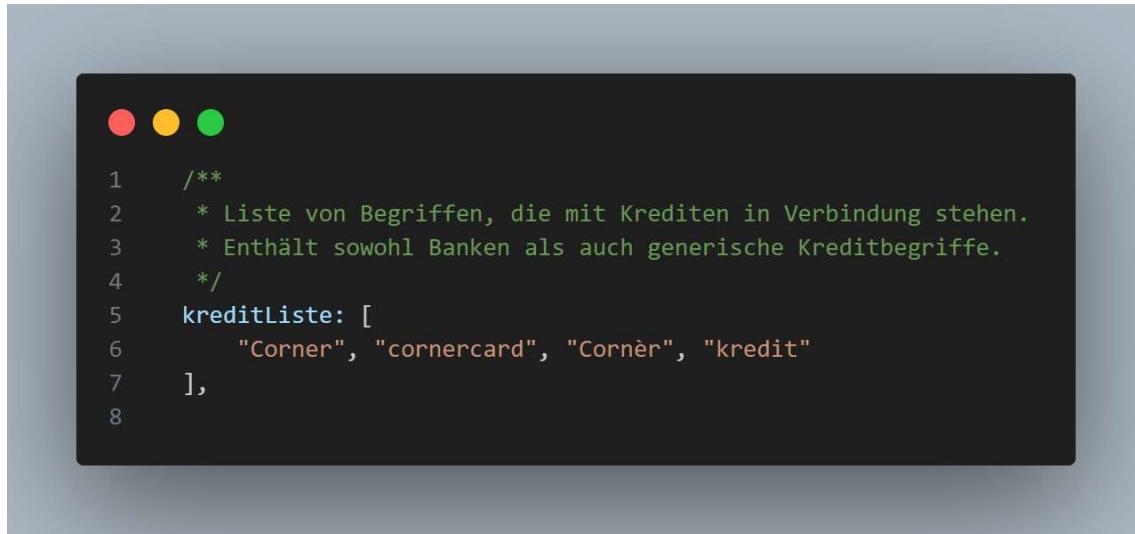
Nachher:

```
1  const fileExists = async (filePath) =>
2      fs.access(filePath).then(() => true).catch(() => false);
3
```

Abbildung 30:Optimierte Datei-Existenzprüfung mit Arrow-Funktion

8.5 Code Review

Im Rahmen des Code Reviews wurde der gesamte Quellcode nochmals überprüft, um ihn übersichtlicher und verständlicher zu gestalten. Dabei wurden unnötige Kommentare entfernt, Variablennamen verbessert und kleinere Anpassungen zur Erhöhung der Lesbarkeit vorgenommen.



```
1  /**
2   * Liste von Begriffen, die mit Krediten in Verbindung stehen.
3   * Enthält sowohl Banken als auch generische Kreditbegriffe.
4   */
5  kreditListe: [
6    "Corner", "cornercard", "Cornèr", "kredit"
7  ],
8
```

Abbildung 31: Unnötiger Kommentar

Den Kommentar einfach entfernt da man aus dem Array einfach lesen kann, was es ist.



8.6 Reflexion

Ich bin froh, dass mein Projekt nun abgeschlossen ist und ich mit meiner Leistung zufrieden sein kann. Es war mein zweiter Anlauf für die IPA, und ich merke, dass ich mich im Vergleich zum ersten Mal deutlich verbessert habe. Besonders bei der Codierung und Strukturierung meines Projekts habe ich aus meiner letzten IPA viel mitgenommen und konnte meine Arbeit dadurch besser gestalten.

Eine der grössten Herausforderungen war es, meinen Zeitplan konsequent einzuhalten. Mir war von Anfang an bewusst, dass die Aufgaben umfangreich sind und gut koordiniert werden müssen, um nicht unter Zeitdruck zu geraten. Deshalb habe ich mir jeden Tag klare Ziele gesetzt und regelmässig überprüft, ob ich im Zeitplan bin. So konnte ich Abweichungen frühzeitig erkennen und anpassen, sodass ich am Ende alles rechtzeitig fertigstellen konnte.

Besonders zufrieden bin ich mit der **OCR-Integration**. Das System läuft stabil und könnte theoretisch direkt bei Ajooda eingesetzt werden. Die Texterkennung funktioniert zuverlässig, die Kategorisierung läuft automatisiert und Dokumente werden verarbeitet. Das war einer der wichtigsten Bestandteile des Projekts, und es ist cool zu sehen, dass es genau so funktioniert, wie ich es geplant habe.

Während der Entwicklung habe ich verschiedene Ansätze analysiert, vor allem bei der **Kategorisierung und Fehlerbehandlung**. Ich habe mich bewusst für eine regelbasierte Kategorisierung entschieden, weil sie für den Anwendungsfall am besten passt. Die Implementierung einer robusten Fehlerbehandlung war ebenfalls wichtig, damit das System stabil läuft und Fehler klar angezeigt werden.

Wenn ich das Projekt weiterentwickeln würde, wären **eine PDF-Unterstützung** oder **eine KI-gestützte Kategorisierung** sinnvolle Erweiterungen. Insgesamt habe ich durch dieses Projekt viel gelernt – nicht nur technisch, sondern auch in der Art und Weise, wie ich strukturiert arbeite und meine Zeit einteile. Die Erfahrungen aus dieser IPA werde ich auf jeden Fall in zukünftigen Projekten nutzen.

8.7 Fazit

Das Projekt war eine wertvolle Erfahrung für mich, sowohl technisch als auch organisatorisch. Ich konnte mein Wissen in der Softwareentwicklung erweitern und habe gelernt, effizienter zu planen und meine Zeit besser einzuteilen. Besonders die Arbeit mit der **Google Cloud Vision API** und die Umsetzung der **OCR-Integration** haben mir gezeigt, wie man externe Services sinnvoll in eine eigene Anwendung integriert.

"Every mistake is a lesson." – Wukong



9 Anhang

9.1 Glossar

Batch-Verarbeitung

Automatisierte Verarbeitung mehrerer Dateien oder Daten in einem einzigen Durchgang.

OCR (Optical Character Recognition)

Automatisierte Erkennung und Extraktion von Texten aus Bildern oder eingescannten Dokumenten.

Google Cloud Vision API

Cloud-basierter Dienst von Google zur Analyse von Bildern und zur Texterkennung mittels OCR.

AppError-Klasse

Benutzerdefinierte Fehlerklasse zur strukturierten Behandlung und Protokollierung von Fehlern innerhalb der Anwendung.

Multer

Bibliothek zur Handhabung und Verarbeitung von Datei-Upsloads in Node.js-Anwendungen.

.env-Datei

Datei zur sicheren Speicherung von Umgebungsvariablen, insbesondere sensiblen Daten wie API-Schlüsseln.



9.2 Quellenverzeichnis

Thema	Link
Maga	https://www.magaya.com/6-ways-freight-forwarders-can-benefit-from-ocr-technology/#:~:text=,compliance%20and%20billing%2C%20and%20more
OCR with Google Cloud Vision	Link bugged beim einfügen (Hyperlink)
Node.js	https://nodejs.org/en
OCR Video Google Cloud Vision	https://www.youtube.com/watch?v=lyLNuLnH6Xg
AppDiagramm	https://app.diagrams.net/
Figma	https://www.figma.com
Units Test	https://q-centric.com/blogs/unit-tests-vs-integrationstests-beide-testmethoden-erklärt
Projektumgebung	https://www.kirenz.com/tutorials/data-science-toolkit/erste-schritte/projektumgebung
Text Extraction	https://stackoverflow.com/questions/51045843/google-cloud-functions-text-extraction-from-image-using-vision-api
Google Cloud Bild Erkennung	https://cloud.google.com/vision/docs/ocr?hl=de
Google Cloud Vision Api Node js	https://cloud.google.com/nodejs/docs/reference/vision/latest
Handling multiple files	https://codezup.com/node-js-file-uploads-expressjs/
Multer	https://blog.logrocket.com/multer-nodejs-express-upload-file/
Multer	https://github.com/meetayush2016/multer-use
Uploading files using Multer	https://www.slingacademy.com/article/node-js-express-file-upload-multer/



Projektmanagement Methode

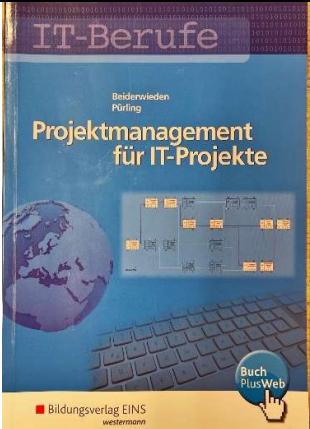


Abbildung 32: Projektmanagement für IT-Projekte

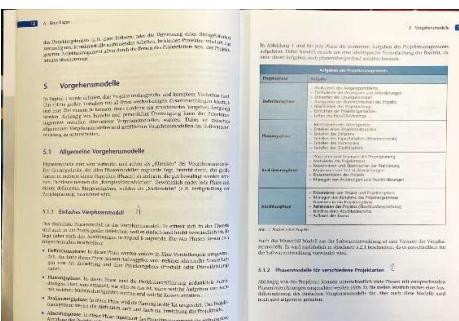


Abbildung 33: Vier Phasen Modell

Diagramme

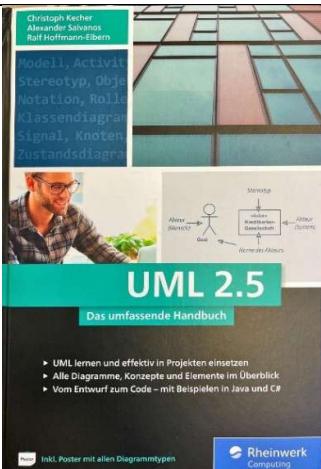


Abbildung 34: UML Buch

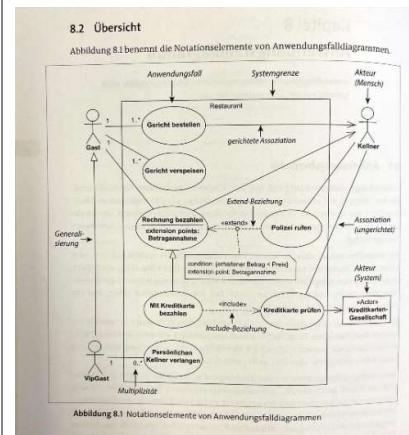


Abbildung 35: Anwendungsfalldiagramm

Kommentare

https://en.wikipedia.org/wiki/Comment_%28computer_programming%29

https://en.wikipedia.org/wiki/GNU_coding_standards



	https://eckcellent-it.de/2019/10/22/clean-code-part-2/
Unit Testen	https://de.parasoft.com/blog/how-to-write-test-cases-for-software-examples-tutorial/ https://www.it-agile.de/agiles-wissen/agile-entwicklung/unit-tests/ https://brightsec.com/blog/unit-testing/
Integrationstest	https://www.youtube.com/watch?v=IM80ablYhPE
Systemtest	https://www.youtube.com/watch?v=CugnGw0zsMs https://www.reddit.com/r/embedded/comments/14lhaq8/integration_test_vs_system_test/?tl=de&rdt=59281 https://www.studysmarter.de/ausbildung/ausbildung-in-it/fachinformatiker-anwendungsentwicklung/integrationstests/
Reflexion	https://www.scribbr.ch/studium-ch/reflexion-schreiben/



9.3 Abbildungsverzeichnis

Abbildung 1:Use-Case-Diagramm der OCR-Dokumentenverarbeitung	12
Abbildung 2: Prozessablauf	15
Abbildung 3: Mockup Startseite	16
Abbildung 4: Mockup Pop Up.....	17
Abbildung 5: Mockup Eingabe Pfad	18
Abbildung 6: Mockup Tabellenansicht.....	19
Abbildung 7: Mockup mehr Text anzeigen	20
Abbildung 8: Backup Laufwerk	27
Abbildung 9:Code Versionierung	27
Abbildung 10: Startseite in Code.....	30
Abbildung 11: Startseite UI.....	31
Abbildung 12:Methode zur Ordnerauswahl in der Benutzeroberfläche.....	31
Abbildung 13: Standard-Serverkonfiguration	32
Abbildung 14: ResultTemplate Tabelle	32
Abbildung 15:Dynamische Generierung einer Ergebnistabelle	33
Abbildung 16: Google Cloud Implementation	34
Abbildung 17:Texterkennung in Dateien.....	34
Abbildung 18:API-Route zur Verarbeitung von Datei-Upsloads.....	35
Abbildung 19:API-Route zur Verarbeitung von Ordnerpfaden	35
Abbildung 20:Funktion zur Verarbeitung eines gesamten Ordners	36
Abbildung 21: Kategorien	37
Abbildung 22: Vergleich des Textes mit den Kategorien	38
Abbildung 23:Implementierung der Dateiverarbeitung	39
Abbildung 24:Automatische Umbenennung	40
Abbildung 25: Manuelle Umbenennung.....	41
Abbildung 26:Interaktive Umbenennung	41
Abbildung 27:AppError-Klasse	42
Abbildung 28:Beispiel Unit-Test Code	45
Abbildung 29:Vorherige Implementierung der Datei-Existenzprüfung	48
Abbildung 30:Optimierte Datei-Existenzprüfung mit Arrow-Funktion	48
Abbildung 31:Unnötiger Kommentar	49
Abbildung 32:Projektmanagement für IT-Projekte	53
Abbildung 33:Vier Phasen Modell	53
Abbildung 34:UML Buch.....	53
Abbildung 35:Anwendungsfalldiagramm	53
Abbildung 36: Zwischenstand OCR Integration	59
Abbildung 37:Zwischenstand Kategorisierung	62
Abbildung 38:Units Test Beispiel Konsole	65



9.4 Arbeitsjournal

Legende: Erfüllt, Fortlaufend (Geht am nächsten Tag weiter), Pausiert

9.4.1 Tag 1, Dienstag, 04.03.2025

Tagesaktivität	Herausforderung	Erfüllt?
Soll/Ist 09:00-10:00	Projekt-Kickoff: Mein erster PA-Tag, und ich habe morgens im PkOrg die Aufgabenstellung durchgelesen und mir dazu Notizen gemacht..	
Soll/Ist 10:00-11:30	Grober Zeitplan: Direkt danach habe ich mich an den groben Zeitplan gemacht. Ich habe im Internet recherchiert, wie so etwas aussehen soll, aber nichts Passendes gefunden – also habe ich etwas Eigenes erstellt.	
Soll/Ist 11:30-12:00 13:00-13:30	Anforderungsanalyse begonnen: Erste funktionale und nicht-funktionale Anforderungen definiert..	
Soll/Ist 13:30 – 15:30	Nach dem Mittag weiter an der Wissensbeschaffung: Recherche zur OCR-Technologie: Vergleich zwischen Google Cloud Vision API und Tesseract OCR sowie Integration von Google Cloud Vision API in Node.js	
Soll/Ist 15:30 – 16:00	Technologieentscheidung: Ich konnte mich recht schnell entscheiden, weil es sehr einseitig war.	
Soll/Ist 16:00 - 18:00	Dokumentation des Tagesfortschritts: Anforderungsanalyse, Technologieentscheidung und erste Erkenntnisse festgehalten.	 Aber Knapp
Immer noch in Phase 1		
Problem	-	
Hilfmittel	Chatgpt, Youtube,	
Reflexion	Am Anfang war ich ein bisschen gestresst, aber mit der Zeit ging es, und ich kam in einen Flow. Ich habe keine Zeit mit Design verschwendet, sondern mich auf das Wesentliche konzentriert. Ein Spaziergang über den Mittag hat mir dabei echt gutgetan. Tag 1 fand im Homeoffice statt, da wegen der Fasnacht in der Stadt, wo das Geschäft liegt, mit viel Lärm zu rechnen war.	Anfang Stimmung 8/10 Danach 8/10



9.4.2 Tag 2, Mittwoch, 05.03.2025

Tagesaktivität	Herausforderung	Erfüllt?
Soll/Ist 09:00-11:00	<p>Use Case: Etwas müde wieder in den Tag gestartet, habe ich mich direkt ans Use-Case-Diagramm gemacht. Habe von der Schule ein Buch bekommen und daraus viel verwenden können.</p> <p>Detaillierten Zeitplan: Die restliche Zeit bis zum Mittag habe ich für den detaillierten Zeitplan genutzt. Ich habe mir nebenbei auch aufgeschrieben, wie die nächsten Tage aussehen sollen.</p>	<input checked="" type="checkbox"/>
Soll/Ist 11:00-12:00		<input checked="" type="checkbox"/>
Soll/Ist 13:00-15:00	<p>Wir sind jetzt in der Planungsphase Erster Meilenstein erreicht</p> <p>Systemarchitektur: Für die Systemarchitektur habe ich verschiedene Projekte im Internet analysiert und geprüft, welche Ansätze am besten passen. Dabei habe ich viele Notizen gemacht und Best Practices angewandt.</p>	<input checked="" type="checkbox"/>
Soll 15:00-16:00 Ist 15:00-16:30	<p>Mockup: Danach habe ich ein Mockup erstellt. Ich hatte ein paar Schwierigkeiten, eine passende Website zu finden, also habe ich einfach meinen VF gefragt, welche eine gute Seite dafür wäre. Er meinte Figma – und es stellte sich heraus, dass es die perfekte Seite für solche Sachen ist. Dafür habe ich heute 30 Minuten länger gebraucht, also weniger Doku.</p>	<input checked="" type="checkbox"/>
Soll 16:00 - 18:00 Ist 16:30 – 18:00	<p>Doku: Weiter an meiner Doku gearbeitet, den Text verbessert, Fehler korrigiert und ihn leserlicher gestaltet.</p>	
Problem	-	
Hilfsmittel	Draw.io, Figma,	
Reflexion	<p>Wieder im Geschäft konnte ich mich besser konzentrieren und war schnell im Flow, da auch meine Mitmenschen voll fokussiert waren.</p> <p>Ich persönlich finde, dass ich heute viel erreicht habe, und wenn ich so weitermache, kann ich es zufrieden abgeben.</p> <p>Unter anderem habe ich meinen ersten Meilenstein erfolgreich erreicht.</p>	<p>Anfang Stimmung 8/10</p> <p>Danach 9/10</p>

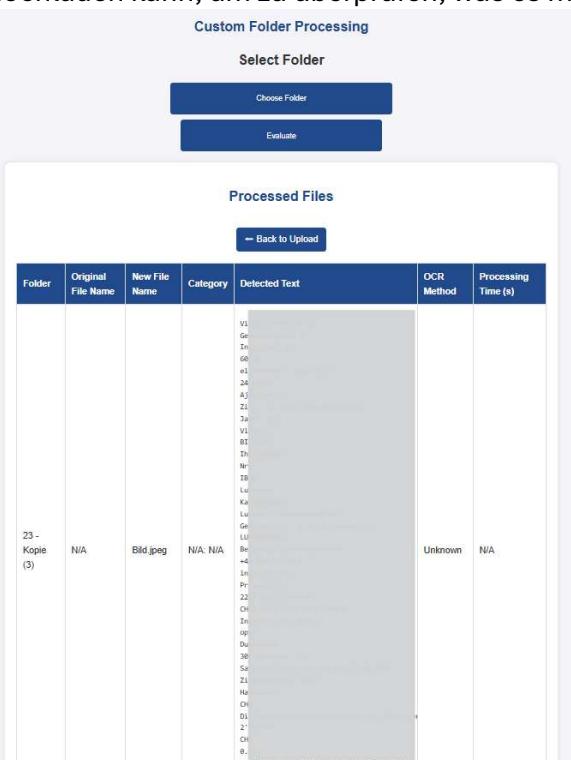


9.4.3 Tag 3, Donnerstag, 06.03.2025

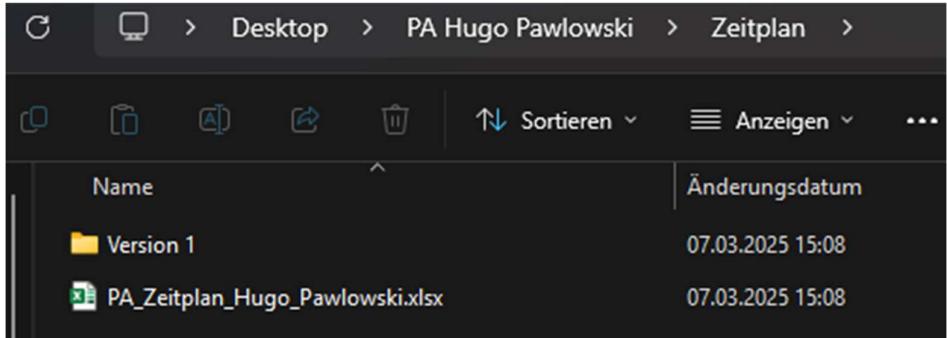
Tagesaktivität	Herausforderung	Erfüllt?
Soll/Ist 09:00-10:30	<p>Testkonzept:</p> <p>Am Morgen habe ich mich mit dem Testkonzept beschäftigt. Ich habe ein bisschen unterschätzt, wie viel es ist, bin aber dennoch fertig geworden.</p> <p>Für das Testkonzept habe ich mir etwas mehr Zeit eingeplant, da ich wusste, dass es eine höhere Priorität hat.</p>	<input checked="" type="checkbox"/>
Soll/Ist 10:30-12:00	<p>Einrichtung Projektumgebung:</p> <p>Nach dem Testkonzept habe ich mich mit der Einrichtung der Projektumgebung befasst. Dort habe ich ungefähr die Ordnerstruktur und die Installation beschrieben.</p>	<input checked="" type="checkbox"/>
Soll/Ist 13:00-16:00	<p>Wir sind jetzt in der Realisierungsphase Meilenstein erreicht</p> <p>Implementierung der Benutzeroberfläche:</p> <p>Ich habe mich für ein schlichtes und benutzerfreundliches Design entschieden. Dabei lege ich Wert auf eine einfache und übersichtliche Gestaltung.</p> <p>Ausserdem habe ich schon angefangen die Order zu erstellen wie (Controller, Route, Services etc)</p>	<input type="checkbox"/>
Soll/Ist 16:00 - 17:00	<p>Experten besuch</p> <p>Am Nachmittag hat mich der Prüfungsexperte besucht. Wir sind seine Checkliste durchgegangen und haben offene Fragen gegenseitig beantwortet. Während des Gesprächs habe ich mir auch Notizen gemacht, damit ich diese später anwenden kann.</p>	
Soll/Ist 18:00 – 19:00	<p>Doku:</p> <p>Direkt danach habe ich weiter an der Doku gearbeitet und ein paar Punkte bearbeitet, wie das Vorwort, sowie einige Sätze angepasst.</p>	<input checked="" type="checkbox"/>
Problem	-	
Hilfsmittel	YouTube, ChatGPT, Hilfs Webseiten für die Programmierung	
Reflexion	Ein bisschen aufgereggt über den Tag, da der Prüfungsexperte kam, aber dennoch sehr Konfident. Es lief gut ab.	Anfang Stimmung 7/10 Danach 8/10



9.4.4 Tag 4, Freitag, 07.03.2025

Tagesaktivität	Herausforderung	Erfüllt?
Soll: 08:00-10:00 Ist 08:00-09:00	Heute Um 8 Uhr angefangen da das Geschäft um 17 Uhr schliesst und ich nicht länger bleiben kann. Implementierung der Benutzeroberfläche: Ich bin heute etwas früher mit der Implementierung der Benutzeroberfläche fertig geworden. Die Zeit konnte ich dann für die Dokumentation nutzen.	<input checked="" type="checkbox"/>
Soll: 10:00-12:00 Ist 09:00-12:00	OCR-Integration: Bei der OCR-Integration habe ich die Grundstrukturen programmiert, damit das System zunächst funktioniert und ich einfache Bilder hochladen kann, um zu überprüfen, was es mir ausgibt.	<input checked="" type="checkbox"/>
Soll/Ist 13:00-15:00	 <p>Abbildung 36: Zwischenstand OCR Integration</p>	<input checked="" type="checkbox"/>
Soll/Ist 15:00 – 17:00	Doku Bei der Dokumentation habe ich im Nachhinein einige Dinge angepasst, wie das Soll/Ist bei der Tagesaktivität sowie redundante oder sich wiederholende Passagen. Stattdessen habe ich sie einfach referenziert.	<input checked="" type="checkbox"/>
Problem	Ich hatte ein Problem mit dem Speicher von meinem Zeitplan also habe ich die Datei einfach nochmal extra gespeichert und die alte Version in	



	einen Ordner getan (Version1) 	
Hilfsmittel	ChatGPT, Hilfs Webseiten für die Programmierung	
Reflexion	Beim Programmieren ist mir nichts schwergefallen, da Google Cloud Vision sehr gut dokumentiert ist und ich mir vieles ableiten konnte. Ich hatte an einer Stelle 1–2 Fehler, aber diese entstanden nur durch Tippfehler.	Anfang Stimmung 8/10 Danach 8/10

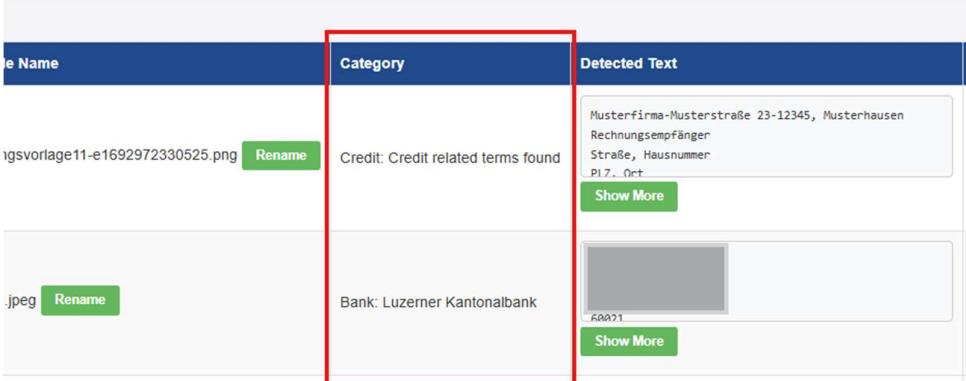


9.4.5 Tag 5, Montag, 10.03.2025

Tagesaktivität	Herausforderung	Erfüllt?
Soll/Ist 09:00-12:00 13:00-16:00	<p>OCR-Integration: Pünktlich um 9 Uhr konnte ich mich wieder mit dem Programmieren beschäftigen. Es lief die ganze Zeit gut, ich hatte kaum Fehlermeldungen. Die Funktionen, wie der Folder (der den Folderpath anzeigt) oder Processing Time (s), haben jetzt eine Ausgabe.</p> <p>An dem heutigen Tag war es auch mehr Teten und ausprobieren als Dokumentationen lesen.</p> <p>Dennoch bin ich der Meinung das ich mit der OCR Integration Dienstag fertig werde und ich zu dem nächsten Schritt gehe kann.</p> <p>Nebenbei Notizen für die Doku aufgeschrieben.</p>	
Soll/Ist 16:00-18:00	Am Nachmittag wie üblich weiter an meiner Doku geschrieben. Noch Sätze ergänzt, quellen hinzugefügt und Verzeichnisse aktualisiert.	
Problem	-	
Hilfsmittel	Hilfs Webseiten für die Programmierung, Google Cloud Vision Doc, JavaScript Doc	
Reflexion	Heute war ein guter Tag, ich fand die Entscheidung von mir gut den ganzen Tag dafür einzuplanen, dafür war ich in einen Flow und konnte mich ganz mit meinem Code befassen.	Anfang Stimmung 8/10 Danach 9/10



9.4.6 Tag 6, Dienstag, 11.03.2025

Tagesaktivität	Herausforderung	Erfüllt?
Soll 09:00-12:00 13:00-14:00	OCR-Integration: Bis zur Mittagspause habe ich weiterhin an der OCR-Integration gearbeitet. Bisher war es nämlich so, dass Unterordner nicht ausgewertet wurden. Also musste ich etwas im folderController anpassen.	<input checked="" type="checkbox"/>
Ist 9:00-12:00	Dazu habe ich eine Stunde früher aufgehört, da ich schneller fertig geworden bin. Diese Zeit nutzte ich für die Kategorisierung.	
Soll 14:00-16:00	Kategorisierung: Nach dem Mittag habe ich an der Kategorisierung gearbeitet. In der detaillierten Aufgabenstellung stand nicht genau, welche Kategorien ich verwenden muss. Dort hiess es lediglich: „ <i>Anschliessend erfolgt eine Klassifizierung anhand vordefinierter Kriterien wie 'Bank'; 'Krankenkasse'; 'Lohnausweis'; 'Kredit' etc.</i> “ Da diese Angabe zu ungenau war, fragte ich meinen VF. Er meinte, dass ich folgende fünf Kategorien mit Code einfügen soll: Bank, Krankenkasse, Lohnausweis, Kredit und Steuerdokument .	
Ist 13:00-16:00		
 <p>Abbildung 37: Zwischenstand Kategorisierung</p>		
Soll/Ist 16:00-18:00	Doku: Ich habe in der Doku den Abschnitt zur Kategorisierung überarbeitet und Screenshots der neuen UI hinzugefügt. Zudem habe ich den Abschnitt über die Speicherung der Ergebnisse ergänzt. Richtung Abschlussphase	<input checked="" type="checkbox"/>
Problem	-	
Hilfsmittel	Hilfs Webseiten für die Programmierung, Stack Overflow, Reddit	
Reflexion	Heute ging der Tag sehr schnell vorbei, ich war so fokussiert das ich glatt die Zeit vergessen habe	Anfang Stimmung 8/10 Danach 8/10



9.4.7 Tag 7, Mittwoch, 12.03.2025

Tagesaktivität	Herausforderung	Erfüllt?
Soll/Ist 09:00-11:00	Kategorisierung: Heute Morgen konnte ich die Kategorisierung abschliessen. Ich habe noch 1-2 Testdaten ausgewertet und dabei gute Ergebnisse erhalten.	<input checked="" type="checkbox"/>
Soll/Ist 11:00-12:00 13:00-16:00	Datei Umbenennen/Sortierung: Heute hatte ich etwas weniger Zeit für die Doku, da das Programmieren im Vordergrund stand. Die Bereiche „ Datei Umbenennen/Sortierung “, „ Fehlerbehandlung “ und „ Unit Test “ wurden ergänzt und auf den aktuellen Stand gebracht.	<input type="checkbox"/>
Soll/Ist 16:00-18:00	Doku: Den Abschnitt zur Datei-Umbenennung und -Sortierung habe ich heute ebenfalls abgeschlossen und in die Dokumentation eingefügt.	<input checked="" type="checkbox"/>
Problem	-	
Hilfsmittel	ChatGPT	
Reflexion	Heute musste ich mich wirklich konzentrieren, da ich ein paar Fehler im Code hatte und kaum etwas im Web finden konnte. Also habe ich GPT gefragt. Bin für heute fertig – ziemlich schlapp.	Anfang Stimmung 8/10 Danach 7/10



9.4.8 Tag 8, Donnerstag, 13.03.2025

Tagesaktivität	Herausforderung	Erfüllt?
Soll/Ist 09:00-10:00	Datei Umbenennen/Sortierung: Ich habe abschliessende Tests zur Funktion für das Umbenennen und Sortieren von Dateien durchgeführt, um sicherzustellen, dass alle Dateien korrekt und zuverlässig umbenannt und abgelegt werden.	<input checked="" type="checkbox"/>
Soll/Ist 10:00-12:00 13:00-16:00	Fehlerbehandlung: In dieser Zeit habe ich die „AppError“-Klasse vollständig implementiert und ausführlich getestet, um eine konsistente Fehlerbehandlung in meinem System zu gewährleisten. Dabei habe ich Fehlerfälle simuliert, insbesondere bei ungültigen Dateien und Kommunikationsproblemen mit der Google Cloud Vision API, und sichergestellt, dass sinnvolle Fehlermeldungen angezeigt werden. Ich habe abschliessende Tests zur Funktion für das Umbenennen und Sortieren von Dateien durchgeführt, um sicherzustellen, dass alle Dateien korrekt und zuverlässig umbenannt und abgelegt werden.	<input checked="" type="checkbox"/>
Soll/Ist 16:00-17:00	Phase 4 Abschlussphase Meilenstein erreicht Unit Test: Ich habe mit der Vorbereitung der Unit-Tests begonnen. Dafür habe ich die Testfälle definiert und erste grundlegende Unit-Tests implementiert. Dies wird morgen weitergeführt und vervollständigt.	<input type="checkbox"/>
Soll/Ist 17:00-18:00	Doku: Heute hatte ich etwas weniger Zeit für die Doku, da das Programmieren im Vordergrund stand. Die Bereiche „ Datei Umbenennen/Sortierung “, „ Fehlerbehandlung “ und „ Unit Test “ wurden ergänzt und auf den aktuellen Stand gebracht.	<input checked="" type="checkbox"/>
Problem	-	
Hilfsmittel	ChatGPT, YouTube	
Reflexion	Heute war ein produktiver Tag. Besonders zufrieden bin ich mit der erfolgreichen Implementierung der Fehlerbehandlung, die nun robust und benutzerfreundlich gestaltet ist. Ich bin happy, dass ich mit der Realisierungsphase fertig bin und nun in der Abschlussphase angekommen bin.	Anfang Stimmung 8/10 Danach 9/10



9.4.9 Tag 9, Freitag, 14.03.2025

Tagesaktivität	Herausforderung	Erfüllt?
Soll/Ist 08:00-9:00	<p>Unit Test:</p> <p>Heute standen die Implementierung und Durchführung der Unit-Tests auf dem Plan. Da ich noch nicht viel Erfahrung mit Unit-Tests hatte, habe ich mir Unterstützung geholt.</p> <p>Mit der Hilfe von ChatGPT konnte ich meine Testfälle besser verstehen und richtig aufbauen. Ich habe gelernt, wie man sie klar dokumentiert und warum das Testen wichtig ist.</p> <pre>PS C:\Users\hpawl\Desktop\PA Hugo Pawłowski\Code\Version 1> node test/ocrService.test.js >> Test: processFile sollte Text aus einem Bild extrahieren Test: processFile sollte Fehler werfen bei nicht unterstütztem Dateitype (node:16620) [DEP0040] DeprecationWarning: The `punycode` module is deprecated. Please use a userland alternative instead. (Use `node --trace-deprecation ...` to show where the warning was created) ✓ processFile funktioniert korrekt!</pre> <p><i>Abbildung 38: Units Test Beispiel Konsole</i></p>	<input checked="" type="checkbox"/>
Soll/Ist: 9:00-11:00	<p>Integrationstest:</p> <p>Für das Testen der Integration habe ich nicht allzu lange gebraucht, da ich bereits das Verständnis für Unit-Tests hatte. Die Integrationstests überprüfen zudem die einzelnen Module.</p>	<input checked="" type="checkbox"/>
Soll/Ist: 11:00-12:00 13:00:15:00	<p>Systemtest & Fehleranalyse:</p> <p>Beim Systemtest habe ich das komplette System mit verschiedenen Anforderungen getestet, z. B. mit einer großen Anzahl an Dateien, unscharfen Bildern oder sehr grossen Dateien.</p> <p>Dabei habe ich im Hintergrund ein paar Videos zu diesen Themen angehört, um noch wichtige Informationen herauszuhören.</p> <p>Dazu hat der VF die Zeit für die Auswertung der Dateien überprüft und meinte, dass die Zeit angesichts der grossen Menge an Dateien gut sei.</p>	<input checked="" type="checkbox"/>
Soll/Ist 15:00-17:00	<p>Doku:</p> <p>Direkt danach habe ich mich wieder mit meiner Doku befasst. Da ich zuvor viel programmiert und getestet habe, wollte ich noch ein paar Dinge anpassen, wie Titel, Sätze etc.</p>	
Problem	-	
Hilfsmittel	ChatGPT, Youtube, Reddit, Stack Overflow	
Reflexion	Heute habe ich Wichtige aufgaben erledigt und bin damit zufrieden, es geht langsam Endspurt von meiner PA und ich bin sehr zufrieden mit meinem Projekt und meiner Doku.	Anfang Stimmung 8/10 Danach 8,5/10



9.4.10 Tag 10, Montag, 17.03.2025

Tagesaktivität	Herausforderung	Erfüllt?
Soll/Ist 9:00-10:00	Da heute schon der letzte Tag meiner PA ist habe ich doppelt so viel Energie und kraft in alles reingesteckt, direkt angefangen mit der Systemtest & Fehleranalyse: Dort habe ich am Morgen weitergearbeitet wo ich letzte Woche aufgehört habe Letzte Tests mit verschiedenen Szenarien durchgeführt, finale Anpassungen vorgenommen. Dabei wurde geprüft, ob das System auch unter hoher Last stabil bleibt.	<input checked="" type="checkbox"/>
Soll/Ist 10:00-11:00	Fehlerkorrektur und Code Optimierung: Code nochmals überprüft, kleinere Fehler entfernt, fileExists-Funktion optimiert. Außerdem wurden einige Redundanzen aus dem Code entfernt.	<input checked="" type="checkbox"/>
Soll/Ist 11:00-12:00	Code Review (Saubерkeit): Unnötige Kommentare gelöscht, Variablennamen verbessert für bessere Lesbarkeit. Dies hilft nicht nur mir, sondern erleichtert auch zukünftigen Entwicklern die Arbeit mit meinem Code.	<input checked="" type="checkbox"/>
Ist 13:00-15:00	Doku: An der Doku gearbeitet auf Formatierungen geschaut und nochmal alles durchgelesen.	<input checked="" type="checkbox"/>
Soll/Ist 15:00-16:00	Finale Doku: Dokumentation komplett durchgegangen, Tippfehler korrigiert, überflüssige Wiederholungen entfernt. Zusätzlich habe ich noch einige Abschnitte klarer formuliert, um die Verständlichkeit zu verbessern.	<input checked="" type="checkbox"/>
Soll/Ist 16:00-16:30	Projektabschluss (Checkliste): Letzte Punkte der Checkliste abgehakt, sichergestellt, dass alles vollständig ist. Dabei wurde auch überprüft, ob alle Tests sauber dokumentiert sind.	<input checked="" type="checkbox"/>
Soll/Ist 16:30-17:00	Reflexion & Fazit: Letzte Gedanken zu meinem Projekt aufgeschrieben und meine Entwicklung reflektiert. Ich bin stolz darauf, wie viel ich in dieser Zeit gelernt und umgesetzt habe.	<input checked="" type="checkbox"/>
Problem	-	
Hilfsmittel	ChatGPT, Youtube	



Reflexion	Heute war mein letzter Tag und ich habe nochmal richtig Gas gegeben. Ich wollte sicherstellen, dass mein Projekt in einem sauberen und abgeschlossenen Zustand ist. Besonders bei der Doku habe ich nochmal genau hingeschaut und alles durchgelesen, damit keine Tippfehler oder unnötige Wiederholungen drin sind. Jetzt ist alles fertig und ich kann mein Projekt mit einem guten Gefühl abgeben.	Anfang Stimmung 9/10 Danach 10/10 🎉
-----------	--	--

9.5 Projektjournal

Datum	Teilnehmer	Thema	Besprochene Punkte	Entscheidungen & Abmachungen	Ergebnis
05.03	Ich (Einzelentscheidung)	Wahl des Vierphasenmodells	Analyse verschiedener Projektmethoden (IPERKA, Wasserfall, V-Modell)	Vierphasenmodell als beste Methode gewählt, weil es flexibel und einfacher ist	Projektstruktur basierend auf Vierphasenmodell definiert
04.03	Ich (Einzelentscheidung)	Entscheidung für Google Cloud Vision API	Vergleich zwischen Google Cloud Vision und Tesseract OCR	Google Cloud Vision API gewählt wegen besserer Genauigkeit und Geschwindigkeit	Implementierung mit Google Cloud API gestartet
11.03	Ich, VF	Kategorisierung	Welche Kategorien es geben soll	Entscheidung auf 5 Kategorien	Bank, Krankenkasse, Lohnausweis, Kredit, Steuerdokument mit Code
12.03	Ich, VF	Speicherung der daten	Welche daten Umbenannt/gespeichert werden müssen	Alle Dateien die decrypted heissen	Dateien die decrypted werden nur Automatisch umbenannt
14.03	Ich, VF	Verarbeitung von vielen Dateien	Ob es die Zeit für eine Verarbeitung von 50+ dateien in der Zeit oke sei	Für so viele Dateien ist es oke	Die Zeit passt



9.6 Quellcode

```
const fs = require('fs').promises;
const path = require('path');
const fileService = require('../services/fileService');
const folderResultsTemplate = require('../templates/folderResultsTemplate');
const AppError = require('../utils/AppError');

/**
 * Verarbeitet einen Ordner und führt OCR-Erkennung sowie Kategorisierung für alle
 * enthaltenen Dateien durch.
 * @param {string} folderPath - Der absolute Pfad zu dem verarbeitenden Ordners.
 * @returns {Promise<Object[]>} - Eine Liste von Objekten mit den Ergebnissen der
 * Datei-Analyse.
 */
async function processFolder(folderPath) {
    const results = [];

    /**
     * Rekursive Funktion zur Verarbeitung eines Ordners.
     * Liest alle Dateien und Unterordner und verarbeitet diese entsprechend.
     * @param {string} currentPath - Der aktuelle Verzeichnispfad.
     */
    async function processDirectory(currentPath) {
        let entries;
        try {
            entries = await fs.readdir(currentPath, { withFileTypes: true });
        } catch (dirError) {
            console.error(`Fehler beim Lesen des Verzeichnisses ${currentPath}:`, dirError);
            throw new AppError(`Fehler beim Lesen des Verzeichnisses: ${currentPath}` , 500, dirError.message);
        }

        // Array zum Sammeln von Datei-Promises
        let filePromises = [];

        for (const entry of entries) {
            const entryPath = path.join(currentPath, entry.name);

            if (entry.isDirectory()) {
                // Rekursiv die Unterordner verarbeiten
                await processDirectory(entryPath);
            } else if (entry.isFile()) {
                // Erstelle einen Promise für die Dateiverarbeitung inkl. Fehlerbehandlung
                filePromises.push(
                    fileService.processFile({
                        filePath: entryPath
                    })
                );
            }
        }

        return filePromises;
    }

    const results = await processDirectory(folderPath);
}
```



```
        path: entryPath,
        originalname: entry.name
    })
    .then(result => ({
        ...result,
        folder: path.dirname(entryPath)
    }))
    .catch(fileError => {
        console.error(` Fehler bei Datei ${entryPath}: `, fileError);
        return {
            fileName: entry.name,
            originalFileName: entry.name,
            detectedText: '',
            ocrMethod: 'Error',
            category: { category: 'Error', foundTerm: 'Processing error' },
            processingTime: '0',
            error: fileError.message,
            folder: path.dirname(entryPath)
        };
    })
);

// Sobald 5 Promises gesammelt wurden, diese parallel abarbeiten
if (filePromises.length === 5) {
    const chunkResults = await Promise.all(filePromises);
    results.push(...chunkResults);
    filePromises = [];
}
}

// Verbleibende Datei-Promises abarbeiten
if (filePromises.length > 0) {
    const chunkResults = await Promise.all(filePromises);
    results.push(...chunkResults);
}

// Starte die rekursive Verzeichnisverarbeitung
await processDirectory(folderPath);

// Formatiere und gebe die Ergebnisse zurück
return folderResultsTemplate(results);
}

module.exports = { processFolder };
```



```
module.exports = {

  bankenListe: [
    "UBS", "Credit Suisse", "Raiffeisen", "Zürcher Kantonalbank", "PostFinance",
    "Vontobel", "Luzerner Kantonalbank", "Mercantil Bank", "Rothschild & Co Bank",
    "Swiss Bank Corporation", "Alpinum Investment Management",
    "Hypothekarbank Lenzburg AG", "Luzerner U Kantonalbank", "SANTANDER",
    "Hypothekarbank"
  ],

  krankenkassenListe: [
    "CSS", "Helsana", "Sanitas", "SWICA", "Concordia", "Visana", "Mutuel",
    "KPT", "Sympany", "Progrès", "Atupri", "Avenir", "Arcosana",
    "Krankenkasse EGK", "Krankenkasse Flaachtal", "Sanagate Versicherungen",
    "Global Sana", "Krankenkasse Philos", "Agrisano Krankenkasse",
    "Sumiswalder Gesundheitskasse", "ÖKK Krankenkasse"
  ],

  lohnAusweisListe: [
    "Lohnausweis",
    "Lohnabrechnung",
    "Gehaltsabrechnung",
    "Lohnzettel",
    "Lohnbescheinigung"
  ],

  kreditListe: [
    "Corner", "cornercard", "Cornèr", "kredit"
  ],

  /**
   * Liste für Steuerdokumente mit Access Code (SAC).
   */
  SAC: [
    "Corner", "cornercard", "Cornèr", "kredit"
  ],
};
```



```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>OCR Folder Processing</title>
  <link rel="stylesheet" href="styles.css">
  <script src="script.js" defer></script>
</head>

<body>
<main>
  <section class="folder-section">
    <h1>Custom Folder Processing</h1>
    <form id="folderForm">
      <div class="form-group">
        <h2>Select Folders</h2>
        <input type="hidden" id="FolderPath" name="FolderPath">
        <button type="button" onclick="selectFolder()">Choose Folder</button>
      </div>
      <div class="action-group">
        <button type="submit" class="primary">Evaluate</button>
      </div>
      <div id="resultContainer"></div>
    </form>
  </section>
</main>
</body>
</html>
```



```
document.addEventListener("DOMContentLoaded", function () {
    // Listener für das Absenden des Formulars zur Verarbeitung des Ordnerpfads
    const folderForm = document.getElementById('folderForm');
    if (folderForm) {
        folderForm.addEventListener('submit', async function (e) {
            e.preventDefault();
            const folderPath = document.getElementById('FolderPath').value;

            if (!FolderPath) {
                alert('Bitte einen Ordnerpfad eingeben.');
                return;
            }

            try {
                // Anfrage an den Server senden, um den Ordner zu verarbeiten
                const response = await fetch('/folders/process-folder', {
                    method: 'POST',
                    headers: { 'Content-Type': 'application/json' },
                    body: JSON.stringify({ folderPath })
                });

                // Antwort als HTML übernehmen und in den Container einfügen
                const html = await response.text();
                document.getElementById('resultContainer').innerHTML = html;

                // Sicherstellen, dass die Toggle-Listener nach dem Laden neu gesetzt werden
                addToggleTextListeners();
            } catch (error) {
                console.error("Fehler beim Verarbeiten des Ordners:", error);
                alert("Fehler beim Verarbeiten des Ordners");
            }
        });
    }

    // Setzt Event-Listener für die "Mehr anzeigen"-Buttons
    addToggleTextListeners();
});

/***
 * Funktion zum Umschalten der Textanzeige zwischen erweitert und reduziert
 * @param {number} index - Der Index des betroffenen Containers
 */
window.toggleText = function(index) {
    const container = document.getElementById(`text-container-${index}`);
    if (!container) return;

    const pre = container.querySelector("pre");
    const button = container.querySelector(".toggle-text-btn");
```



```
// Wechsel zwischen erweiterter und reduzierter Ansicht
pre.classList.toggle("expanded");

// Aktualisiert den Button-Text entsprechend der aktuellen Ansicht
button.textContent = pre.classList.contains("expanded") ? "Show Less" : "Show
More";
};

/***
 * Öffnet einen Dialog zur manuellen Eingabe eines Ordnerpfads und trägt ihn ins
Formular ein.
*/
window.selectFolder = function () {
try{
    const folderPath = prompt("Geben Sie den Ordnerpfad ein:");
    if (FolderPath) {
        const folderInput = document.getElementById("FolderPath");
        if (folderInput) folderInput.value = folderPath;
    }
} catch (err) {
    console.error("Fehler bei der Ordnerauswahl:", err);
    alert("Fehler beim Auswählen des Ordners");
}
};

/***
 * Fügt Event-Listener für alle "Mehr anzeigen"-Buttons hinzu
*/
function addToggleTextListeners() {
    document.querySelectorAll(".toggle-text-btn").forEach(button => {
        button.removeEventListener("click", handleToggleClick); // Entfernt evtl. alte
Listener
        button.addEventListener("click", handleToggleClick);
    });
}

/***
 * Event-Handler für das Umschalten von Textabschnitten
 * @param {Event} event - Das Event-Objekt des Klicks
*/
function handleToggleClick(event) {
    event.preventDefault();
    if (!event.target.dataset.id) return;
    toggleText(event.target.dataset.id);
}

/***
```



```
* Öffnet einen Dialog zum Umbenennen einer Datei und sendet die neue Bezeichnung an den Server.
* @param {HTMLElement} button - Der Button, der die Funktion auslöst (enthält den Dateinamen als Dataset)
*/
function showRenameDialog(button) {
    // Den aktuellen Dateinamen aus data-current verwenden
    const currentFileName = button.dataset.current;
    const folderPath = button.dataset.folder;

    function getFileNameWithoutExtension(filename) {
        return filename.substring(0, filename.lastIndexOf('.')) || filename;
    }

    const newName = prompt("Neuer Dateiname (ohne Endung):",
        getFileNameWithoutExtension(currentFileName));

    if (newName) {
        fetch('/files/manual-rename', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({
                original: currentFileName, // Aktueller Dateiname wird übermittelt
                newName: newName,
                folder: folderPath
            })
        })
        .then(response => response.json())
        .then(data => {
            if (data.success) {
                const cell = button.parentElement;
                const filenameSpan = cell.querySelector('.filename');
                if (filenameSpan) {
                    filenameSpan.textContent = data.newFileName;
                }
                // Aktualisiere data-current für zukünftige Umbenennungen
                button.dataset.current = data.newFileName;
            } else {
                alert(` Fehler: ${data.message} `);
            }
        })
        .catch(error => {
            console.error("Fehler beim Umbenennen der Datei:", error);
            alert("Fehler beim Umbenennen der Datei");
        });
    }
}
```



```
/* Allgemeines Styling */
body {
    font-family: Arial, sans-serif;
    line-height: 1.6;
    margin: 0;
    padding: 0;
    background-color: #f4f4f9;
    color: #333;
    display: flex;
    justify-content: center;
    min-height: 100vh;
}

/* Haupt-Container */
.container {
    width: 60%;
    background: white;
    padding: 20px;
    box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);
    border-radius: 10px;
    text-align: center;
}

/* Überschrift */
h1 {
    color: #074b8c;
    text-align: center;
    margin-bottom: 20px;
}

h2 {
    text-align: center;
    margin-bottom: 20px;
}

/* Upload-Bereich */
.upload-section {
    margin-bottom: 15px;
    text-align: center;
}

/* Label für "Select Folder" */
.upload-section label {
    display: block;
    font-weight: bold;
    margin-bottom: 10px;
}
```



```
/* Buttons */
button {
    background-color: #074b8c;
    color: white;
    border: none;
    padding: 30px 130px;
    cursor: pointer;
    border-radius: 5px;
    display: block;
    margin: 10px auto;
    font-size: 20px;
    font-weight: bold;
    transition: background-color 0.2s ease, transform 0.1s ease;
}

button:active {
    transform: scale(0.98);
}

/* Tabellen-Design */
table {
    width: 100%;
    border-collapse: collapse;
    margin-top: 20px;
    background: white;
}

th, td {
    padding: 10px;
    border: 1px solid #ddd;
    text-align: left;
}

th {
    background-color: #074b8c;
    color: white;
}

tr:nth-child(even) {
    background-color: #f9f9f9;
}

/* Textcontainer */
.text-container {
    max-width: 400px;
    word-wrap: break-word;
    overflow: hidden;
}
```



```
/* Anfangszustand: Zeigt nur 3 Zeilen */
.short-text {
  display: -webkit-box;
  -webkit-line-clamp: 3;
  -webkit-box-orient: vertical;
  overflow: hidden;
  text-overflow: ellipsis;
}

/* Button für Mehr/Weniger anzeigen */
.toggle-text-btn {
  background-color: #5cb85c;
  color: white;
  border: none;
  padding: 8px 15px;
  font-size: 14px;
  border-radius: 3px;
  cursor: pointer;
  display: inline-block;
  margin-top: 5px;
  transition: background-color 0.2s ease;
}

.toggle-text-btn:hover {
  background-color: #4cae4c;
}

/* Back-Button */
.back-button {
  display: inline-block;
  margin-top: 15px;
  padding: 10px 20px;
  background-color: #074b8c;
  color: white;
  text-decoration: none;
  border-radius: 5px;
  font-size: 14px;
  font-weight: bold;
}

pre {
  background: #f8f9fa;
  padding: 10px;
  border: 1px solid #ccc;
  border-radius: 5px;
  overflow-x: auto;
  white-space: pre-wrap;
```



```
margin: 0;
max-height: 5em; /* Standardhöhe für versteckten Text */
overflow: hidden;
transition: max-height 0.3s ease-out;
}

pre.expanded {
  max-height: none !important;
  overflow: auto !important;
}

.manual-rename-btn {
background-color: #5cb85c;
color: white;
border: none;
padding: 8px 15px;
font-size: 14px;
border-radius: 3px;
cursor: pointer;
display: inline-block;
margin-top: 5px;
transition: background-color 0.2s ease;
}

.manual-rename-btn:hover {
background-color: #4cae4c;
}

tr[data-decrypted="true"]{
  background-color: #ffcccc !important;
}
```

```
const {
  bankenListe,
  krankenkassenListe,
  lohnAusweisListe,
  kreditListe
} = require('../lists/liste');
const AppError = require('../utils/AppError');

/**
 * Hilfsfunktion zur Prüfung von Matches in einer vordefinierten Liste.
 *
 * @param {string} text - Der zu durchsuchende Text.
 * @param {Array<string>} list - Die Liste mit Begriffen, die geprüft werden sollen.
 * @returns {string|undefined} - Gibt den gefundenen Begriff zurück oder undefined,
falls keine Übereinstimmung.
```



```
/*
const findMatch = (text, list) => list.find(item => text.includes(item.toLowerCase()));

/**
 * Kategorisiert einen gegebenen Text anhand vordefinierter Listen (Banken,
Krankenkassen, etc.).
 *
 * @param {string} text - Der zu analysierende Text.
 * @returns {Object} - Ein Objekt mit der erkannten Kategorie und dem gefundenen
Begriff.
 *           Falls keine Kategorie erkannt wird, wird "Uncategorized" zurückgegeben.
*/
exports.detectCategory = (text) => {
  try {
    // Validierung: Prüft, ob der Text ein gültiger String ist
    if (!text || typeof text !== 'string') {
      throw new AppError("Kein gültiger Text für die Kategorisierung angegeben.", 400);
    }

    const lowerText = text.toLowerCase();

    // Prüfe, ob der Text einer Bank entspricht
    const bankMatch = findMatch(lowerText, bankenListe);
    if (bankMatch) return { category: "Bank", foundTerm: bankMatch };

    // Prüfe, ob eine Krankenkasse genannt wurde
    const healthInsuranceMatch = findMatch(lowerText, krankenkassenListe);
    if (healthInsuranceMatch) return { category: "HealthInsurance", foundTerm:
healthInsuranceMatch };

    // Prüfe, ob ein Lohnausweis erwähnt wird
    const payrollMatch = findMatch(lowerText, lohnAusweisListe);
    if (payrollMatch) return { category: "Payroll", foundTerm: payrollMatch };

    // Prüfe, ob Kreditbegriffe vorkommen
    const creditMatch = findMatch(lowerText, kreditListe);
    if (creditMatch) return { category: "Credit", foundTerm: creditMatch };

    // Keine Übereinstimmung gefunden → Standard-Kategorie zurückgeben
    return { category: "Uncategorized", foundTerm: "No category detected" };
  } catch (error) {
    console.error("Fehler bei der Kategorisierung:", error);
  }

  // Fehlerbehandlung: Gibt eine Fehlerkategorie zurück
  return { category: "Error", foundTerm: error.message };
};

};
```



```
const fs = require('fs').promises;
const path = require('path');
const ocrService = require('../services/ocrService');
const categoryService = require('../services/categoryService');
const { performance } = require('perf_hooks');
const mime = require('mime-types');
const AppError = require('../utils/AppError');

const counters = {} // Zähler für automatisch umbenannte Dateien basierend auf Kategorien

/**
 * Überprüft, ob eine Datei existiert.
 * @param {string} filePath - Der absolute Pfad der Datei.
 * @returns {Promise<boolean>} - `true` , wenn die Datei existiert, sonst `false` .
 */
const fileExists = async (filePath) =>
  fs.access(filePath).then(() => true).catch(() => false);

/**
 * Erstellt einen einzigartigen Dateinamen, falls die gewünschte Datei bereits existiert.
 * @param {string} folderPath - Der Ordner, in dem sich die Datei befindet.
 * @param {string} baseName - Der gewünschte Basis-Dateiname (ohne Erweiterung).
 * @param {string} extension - Die Dateiendung (z. B. ".pdf").
 * @returns {Promise<string>} - Ein eindeutiger Dateiname mit Zähler, falls notwendig.
 */
async function getUniqueFileName(folderPath, baseName, extension) {
  let counter = 1;
  let newFileName = `${baseName}${extension}`;
  let newPath = path.join(folderPath, newFileName);

  while (await fileExists(newPath)) {
    newFileName = `${baseName}_${counter}${extension}`;
    newPath = path.join(folderPath, newFileName);
    counter++;
  }
  return newFileName;
}

/**
 * Manuelles Umbenennen einer Datei.
 * @param {string} folder - Der Ordner, in dem sich die Datei befindet.
 * @param {string} original - Der ursprüngliche Dateiname (inkl. Endung).
 * @param {string} newName - Der neue gewünschte Dateiname (ohne Endung).
 */
```



```
* @returns {Promise<Object>} - Ein Objekt mit Erfolgsmeldung und neuem Dateinamen.  
* @throws {AppError} - Fehler, falls Datei nicht existiert oder bereits eine Datei mit dem neuen Namen existiert.  
*/  
  
async function renameFile(folder, original, newName) {  
    try {  
        const absoluteFolderPath = path.resolve(folder);  
        const oldPath = path.join(absoluteFolderPath, original);  
        const newPath = path.join(absoluteFolderPath, newName + path.extname(original));  
  
        if (!(await fileExists(oldPath))) {  
            throw new AppError("Datei nicht gefunden.", 404);  
        }  
  
        if (await fileExists(newPath)) {  
            throw new AppError("Datei mit neuem Namen existiert bereits.", 409);  
        }  
  
        await fs.rename(oldPath, newPath);  
        return { success: true, newFileName: newName + path.extname(original) };  
  
    } catch (error) {  
        console.error("Fehler beim Umbenennen:", error);  
        throw error;  
    }  
}  
  
/**  
 * Automatisches Umbenennen einer Datei basierend auf ihrer erkannten Kategorie.  
 * @param {Object} file - Das Dateiobjekt mit `path` und `category`.  
 * @returns {Promise<string>} - Der neue Dateiname oder der ursprüngliche Name, falls keine Kategorie erkannt wurde.  
 * @throws {AppError} - Fehler, falls Datei nicht existiert.  
 */  
  
async function renameFileAutomatically(file) {  
    try {  
        const folderPath = path.dirname(file.path);  
        const fileExtension = path.extname(file.path);  
  
        if (!(await fileExists(file.path))) {  
            throw new AppError("Datei existiert nicht.", 404);  
        }  
  
        if (!file.category || file.category.category === "Uncategorized") {  
            return path.basename(file.path); // Keine Kategorie → Originalname behalten  
        }  
    }  
}
```



```
const categoryKey = file.category.category.toLowerCase();

// Initialisiere einen eigenen Zähler für den Ordner, falls noch nicht vorhanden
if (!counters[folderPath]) {
    counters[folderPath] = {};
}
if (!counters[folderPath][categoryKey]) {
    counters[folderPath][categoryKey] = 1;
}

const baseName =
`${file.category.category}${counters[folderPath][categoryKey]}`;
const newFileName = await getUniqueFileName(folderPath, baseName,
fileExtension);

await fs.rename(file.path, path.join(folderPath, newFileName));
counters[folderPath][categoryKey]++;

return newFileName;
} catch (error) {
    console.error("Fehler beim automatischen Umbenennen:", error);
    throw error;
}
}

/**
 * Verarbeitung einer Datei mit OCR-Erkennung und Kategorisierung.
 *
 * @param {Object} file - Das Dateiobjekt mit `path` und `originalname`.
 * @returns {Promise<Object>} - Ein Objekt mit Datei-Infos, Kategorie, OCR-Text und
Verarbeitungszeit.
 * @throws {AppError} - Fehler, falls die Datei ungültig ist.
 */
async function processFile(file) {
try {
    const startTime = performance.now();
    const mimeType = mime.lookup(file.path) || 'application/octet-stream';

    if (!file.path) {
        throw new AppError("Ungültiger Datei-Pfad.", 400);
    }

    // OCR-Verarbeitung durchführen
    const ocrResult = await ocrService.processFile(file.path, mimeType);
    const endTime = performance.now();

    // Text kategorisieren
}
```



```
const category = ocrResult.detectedText
    ? categoryService.detectCategory(ocrResult.detectedText)
    : { category: "Uncategorized", foundTerm: "No text detected" };

let newFileName = path.basename(file.path);

// Falls es sich um eine entschlüsselte Datei handelt, versuche sie umzubenennen
if (file.originalname.toLowerCase().includes('decrypted')) {
    if (category.category !== "Uncategorized") {
        file.category = category;
        newFileName = await renameFileAutomatically(file);
    }
}

// Rückgabe der Datei-Verarbeitungsinformationen
return {
    fileName: newFileName,
    originalFileName: file.originalname,
    detectedText: ocrResult.detectedText,
    ocrMethod: 'Google Cloud Vision API',
    category,
    processingTime: ((endTime - startTime) / 1000).toFixed(2) // Zeit in Sekunden mit
zwei Dezimalstellen
};

} catch (error) {
    console.error("Fehler bei der Datei-Verarbeitung:", error);
    throw error;
}
}

module.exports = { processFile, renameFileAutomatically, renameFile };
```

```
require('dotenv').config();

const { Storage } = require('@google-cloud/storage');
const vision = require('@google-cloud/vision');
const AppError = require('../utils/AppError');

// Initialisierung von Google Cloud Storage mit den Anmeldedaten aus der .env-Datei
const storage = new Storage({
    keyFilename: process.env.GOOGLE_APPLICATION_CREDENTIALS,
});

// Initialisierung des Google Cloud Vision Clients für OCR-Verarbeitung
const client = new vision.ImageAnnotatorClient();

/**
```



```
* Führt Texterkennung auf einem Bild mit der Google Cloud Vision API durch.  
*  
* @param {string} filePath - Der absolute Pfad zur Bilddatei.  
* @returns {Promise<Object>} - Ein Objekt mit dem erkannten Text und der  
Verarbeitungszeit.  
* @throws {AppError} - Falls die Datei ungültig ist oder ein OCR-Fehler auftritt.  
*/  
async function processImageWithVision(filePath) {  
    const startTime = Date.now();  
  
    try{  
        if (!filePath) {  
            throw new AppError("Kein gültiger Dateipfad angegeben.", 400);  
        }  
  
        // OCR-Verarbeitung über Google Cloud Vision  
        const [result] = await client.textDetection(filePath);  
  
        return {  
            text: result.fullTextAnnotation?.text || "",  
            processingTime: ((Date.now() - startTime) / 1000).toFixed(2)  
        };  
    } catch (error){  
        console.error(`OCR-Fehler bei Datei ${filePath}:`, error);  
        throw new AppError("Fehler bei der OCR-Verarbeitung.", 500, error.message);  
    }  
}  
  
/**  
 * Erkennt Text in einer Datei anhand des MIME-Typs.  
 * Unterstützt aktuell nur Bilddateien.  
 *  
 * @param {string} filePath - Der absolute Pfad zur Datei.  
 * @param {string} [mimeType='application/octet-stream'] - Der MIME-Typ der Datei.  
 * @returns {Promise<Object>} - Ein Objekt mit dem erkannten Text, der Methode und  
der Verarbeitungszeit.  
* @throws {AppError} - Falls der Dateityp nicht unterstützt wird oder ein Fehler  
auftritt.  
*/  
exports.processFile = async (filePath, mimeType = 'application/octet-stream') => {  
    try{  
        if (!filePath) {  
            throw new AppError("Kein Dateipfad angegeben.", 400);  
        }  
  
        if (typeof mimeType !== 'string') {  
            throw new AppError("Ungültiger MIME-Typ.", 400);  
        }  
    }
```



```
const result = {
  detectedText: '',
  ocrMethod: 'none',
  processingTime: 0,
  error: null
};

// Prüft, ob die Datei ein unterstütztes Bildformat hat
if (mimeType.startsWith('image/')) {
  const visionResult = await processImageWithVision(filePath);
  result.ocrMethod = 'google-cloud-vision';
  result.processingTime = parseFloat(visionResult.processingTime);
  result.detectedText = visionResult.text || 'No text found';
} else {
  throw new AppError("Nicht unterstützter Dateityp für OCR.", 415);
}

return result;
} catch (error) {
  console.error("Fehler bei der OCR-Dateiverarbeitung:", error);
  throw error;
}
};
```

```
const path = require('path');

/**
 * Escape HTML characters for safe display in browser.
 * (Optional: könnte in eine separate Hilfsmodul-Datei ausgelagert werden)
 */
function escapeHTML(str) {
  return str.replace(/[\&<>"]"/g, (match) => {
    const escapeMap = {
      '&': '&amp;',
      '<': '&lt;',
      '>': '&gt;',
      '\"': '&quot;',
      '\'' : '&#39;';
    };
    return escapeMap[match];
  });
}

module.exports = function folderResultsTemplate(results) {
  return `<!DOCTYPE html>
<html lang="en">
```



```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Processed Files</title>
  <link rel="stylesheet" href="/styles.css">
  <script defer src="/script.js"></script>
</head>
<body>
  <h1>Processed Files</h1>

  ${results.length === 0 ?
    '<p class="no-results">No files were processed</p>' :
    `<table>
      <thead>
        <tr>
          <th>Folder</th>
          <th>Original File Name</th>
          <th>New File Name</th>
          <th>Category</th>
          <th>Detected Text</th>
          <th>OCR Method</th>
          <th>Processing Time (s)</th>
        </tr>
      </thead>
      <tbody>
        ${results
          .map((result, index) => {
            // Hilfsvariablen zur besseren Lesbarkeit
            const folderName = result.folder ? path.basename(result.folder) : 'N/A';
            const originalFileName = escapeHTML(result.originalFileName || 'N/A');
            const fileName = escapeHTML(result.fileName || 'N/A');
            const category = result.category?.category || 'N/A';
            const foundTerm = escapeHTML(result.category?.foundTerm || 'N/A');
            const detectedText = escapeHTML(result.detectedText || result.error || 'No text
detected');
            const ocrMethod = escapeHTML(result.ocrMethod || 'Unknown');
            const processingTime = result.processingTime || 'N/A';

            // Klarere Bedingung für "decrypted"
            const isDecrypted = originalFileName.toLowerCase().includes('decrypted');

            return `
              <tr data-decrypted="${isDecrypted ? 'true' : 'false'}">
                <td>${folderName}</td>
                <td>${originalFileName}</td>
                <td>
                  <span class="filename">${fileName}</span>
                  ${fileName !== 'N/A' ? `
```



```
<button type="button" class="manual-rename-btn toggle-text-btn"
    data-current="${fileName}"
    data-folder="${escapeHTML(result.folder)}"
    onclick="showRenameDialog(this)">
    Rename
</button>` : "
</td>
<td>${category}: <span class="found-term">${foundTerm}</span></td>
<td class="text-container" id="text-container-${index}">
    <pre>${detectedText}</pre>
    ${detectedText.length > 10 ? `
        <button class="toggle-text-btn" data-id="${index}">Show More</button>` :
    `}
</td>
<td>${ocrMethod}</td>
<td>${processingTime}</td>
</tr>`;
}
.join(")");
</tbody>
</table>`}
</body>
</html>`;
};
```

Test

```
const assert = require('assert');
const http = require('http');
const fs = require('fs').promises;

// **Mock für `fs.promises.readdir()` → Simuliert, dass dein Ordner existiert**
fs.readdir = async (FolderPath) => {
    console.log(`📁 Mock fs.readdir aufgerufen für: ${FolderPath}`);
    if (FolderPath === "C:/Users/hpawl/Desktop/23 - Kopie (3) - Kopie - Kopie/23 - Kopie (3)") {
        return ["file1.jpg", "file2.png"]; // Fake-Dateien simulieren
    }
    throw new Error(`ENOENT: no such file or directory, scandir '${FolderPath}'`);
};

const BASE_URL = "http://localhost:3000"; // Passe die Portnummer an, falls dein Server auf einem anderen Port läuft.

/**
 * Test für `POST /folders/process-folder`
 */
function testProcessFolder() {
```



```
console.log("Test: `POST /folders/process-folder` sollte einen Fehler zurückgeben,  
wenn `folderPath` fehlt");

const options = {  
    hostname: 'localhost',  
    port: 3000,  
    path: '/folders/process-folder',  
    method: 'POST',  
    headers: {  
        'Content-Type': 'application/json'  
    }  
};

const req = http.request(options, (res) => {  
    let data = "";  
  
    res.on('data', (chunk) => {  
        data += chunk;  
    });  
  
    res.on('end', () => {  
        console.log("🔍 Antwort:", data);  
        assert.strictEqual(res.statusCode, 400, "Fehlercode sollte 400 sein");  
        console.log("✅ `POST /folders/process-folder` Fehler-Handling funktioniert!");  
  
        // Zweiter Test: Gültiger Ordnerpfad  
        testProcessFolderValid();  
    });  
});  
  
req.write(JSON.stringify({})); // Leerer Body  
req.end();  
}  
  
/**  
 * Test für `POST /folders/process-folder` mit gültigem Ordnerpfad  
 */  
function testProcessFolderValid() {  
    console.log("Test: `POST /folders/process-folder` sollte eine HTML-Antwort  
zurückgeben");  
  
    const options = {  
        hostname: 'localhost',  
        port: 3000,  
        path: '/folders/process-folder',  
        method: 'POST',  
        headers: {
```



```
'Content-Type': 'application/json'
    }
};

const req = http.request(options, (res) => {
    let data = "";

    res.on('data', (chunk) => {
        data += chunk;
    });

    res.on('end', () => {
        console.log("🔍 Antwort erhalten:", data);

        // Sicherstellen, dass die Antwort HTML enthält
        assert.strictEqual(res.statusCode, 200, "Status sollte 200 sein");
        assert.match(data, /<html[\s\S]*</html>/, "Antwort sollte HTML enthalten");

        console.log(` POST /folders/process-folder` funktioniert mit gültigem
Ordnerpfad!");

        // Nächster Test: Datei-Umbenennung
        testManualRename();
    });
});

req.write(JSON.stringify({ folderPath: "C:/Users/hpawl/Desktop/23 - Kopie (3) - Kopie
- Kopie/23 - Kopie (3)" }));
req.end();
}

/**
 * Test für `POST /files/manual-rename`
 */
function testManualRename() {
    console.log("Test: `POST /files/manual-rename` sollte Fehler werfen, wenn
Parameter fehlen");

    const options = {
        hostname: 'localhost',
        port: 3000,
        path: '/files/manual-rename',
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        }
};
```



```
const req = http.request(options, (res) => {
    let data = "";

    res.on('data', (chunk) => {
        data += chunk;
    });

    res.on('end', () => {
        console.log(" Antwort:", data);
        assert.strictEqual(res.statusCode, 400, "Fehlercode sollte 400 sein");
        console.log(` POST /files/manual-rename` ` Fehler-Handling funktioniert!`);

        // Zweiter Test: Erfolgreiches Umbenennen
        testManualRenameValid();
    });
});

req.write(JSON.stringify({ original: "old.txt", newName: "new" }));
req.end();
}

/**
 * Test für ` POST /files/manual-rename` mit gültigen Parametern
 */
function testManualRenameValid() {
    console.log("Test: ` POST /files/manual-rename` sollte eine Datei erfolgreich umbenennen");

    const options = {
        hostname: 'localhost',
        port: 3000,
        path: '/files/manual-rename',
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        }
    };

    const testFilePath = "C:/Users/hpawl/Desktop/23 - Kopie (3) - Kopie - Kopie/23 - Kopie (3)/old.txt";

    // **Erstelle die Datei, falls sie nicht existiert**
    require('fs').writeFileSync(testFilePath, "Testinhalt");

    const req = http.request(options, (res) => {
        let data = "";

        res.on('data', (chunk) => {
```



```
    data += chunk;
  });

  res.on('end', () => {
    console.log("🔍 Antwort:", data);
    assert.strictEqual(res.statusCode, 200, "Status sollte 200 sein");

    // **Überprüfen, ob alte Datei noch existiert, bevor sie gelöscht wird**
    const fs = require('fs');
    if (fs.existsSync(testFilePath)) {
      fs.unlinkSync(testFilePath);
    }

    console.log(` ` POST /files/manual-rename` funktioniert!`);
  });
});

req.write(JSON.stringify({
  original: "old.txt",
  newName: "new",
  folder: "C:/Users/hpawl/Desktop/23 - Kopie (3) - Kopie - Kopie/23 - Kopie (3)"
}));
req.end();
}

// **Tests starten**
testProcessFolder();

const assert = require('assert');
const AppError = require('../utils/AppError');

function testAppError() {
  console.log("Test: AppError sollte eine Instanz von Error sein");

  const error = new AppError("Test Fehler", 400, "Details");

  assert(error instanceof Error, "AppError sollte von Error erben");
  assert.strictEqual(error.message, "Test Fehler", "Fehlermeldung sollte korrekt sein");
  assert.strictEqual(error.statusCode, 400, "StatusCode sollte 400 sein");
  assert.strictEqual(error.details, "Details", "Details sollten korrekt gesetzt sein");

  console.log(" Test bestanden!");
}

// Test ausführen
testAppError();
```



```
const assert = require('assert');
const categoryService = require('../services/categoryService');

function testDetectCategory() {
    console.log("Test: detectCategory sollte richtige Kategorien erkennen");

    const bankResult = categoryService.detectCategory("Meine Bank ist UBS");
    assert.strictEqual(bankResult.category, "Bank", "Sollte als Bank erkannt werden");

    const healthResult = categoryService.detectCategory("Meine Krankenkasse ist
CSS");
    assert.strictEqual(healthResult.category, "HealthInsurance", "Sollte als
Krankenkasse erkannt werden");

    const payrollResult = categoryService.detectCategory("Lohnabrechnung für Januar
2024");
    assert.strictEqual(payrollResult.category, "Payroll", "Sollte als Payroll erkannt
werden");

    const creditResult = categoryService.detectCategory("Ich habe einen Kredit bei der
Bank");
    assert.strictEqual(creditResult.category, "Credit", "Sollte als Kredit erkannt
werden");

    console.log(" Alle Kategorien wurden korrekt erkannt!");
}

function testUnknownCategory() {
    console.log("Test: detectCategory sollte unbekannte Texte als 'Uncategorized'
zurückgeben");

    const result = categoryService.detectCategory("Das ist ein Test ohne Kategorie");
    assert.strictEqual(result.category, "Uncategorized", "Sollte als Uncategorized
erkannt werden");

    console.log(" Unbekannte Texte wurden korrekt als 'Uncategorized' markiert!");
}

function testEmptyText() {
    console.log("Test: detectCategory sollte leeren Text behandeln");

    const result = categoryService.detectCategory("");
    assert.strictEqual(result.category, "Error", "Leerer Text sollte einen Fehler
zurückgeben");

    console.log(" Leerer Text wurde korrekt als Fehler erkannt!");
}
```



```
// Tests ausführen
testDetectCategory();
testUnknownCategory();
testEmptyText();

const assert = require('assert');
const fileService = require('../services/fileService');
const ocrService = require('../services/ocrService');
const categoryService = require('../services/categoryService');
const fs = require('fs').promises;

const mockFilePath = "/fake/path/testfile.pdf";

async function testProcessFile() {
    console.log("Test: processFile sollte eine Datei analysieren und kategorisieren");

    // Mock OCR-Service
    ocrService.processFile = async () => ({
        detectedText: "UBS Bank Statement",
        ocrMethod: "google-cloud-vision",
        processingTime: "1.2"
    });

    // Mock Kategorie-Service
    categoryService.detectCategory = (text) => ({
        category: "Bank",
        foundTerm: "UBS"
    });

    const file = { path: mockFilePath, originalname: "testfile.pdf" };
    const result = await fileService.processFile(file);

    assert.strictEqual(result.fileName, "testfile.pdf");
    assert.strictEqual(result.category.category, "Bank");
    assert.strictEqual(result.detectedText, "UBS Bank Statement");
    console.log("processFile funktioniert korrekt!");
}

async function testRenameFile() {
    console.log("Test: renameFile sollte eine Datei korrekt umbenennen");

    // Mock Datei-Existenzprüfung
    fs.access = async (filePath) => {
        if (filePath.includes("newname.txt")) {
            throw new Error("Datei existiert nicht"); // Simuliere, dass die neue Datei nicht
existiert
        }
    }
}
```



```
};

// Mock fs.rename
fs.rename = async () => { };

const result = await fileService.renameFile("/fake/path", "oldname.txt", "newname");

assert.strictEqual(result.success, true);
assert.strictEqual(result.newFileName, "newname.txt");
console.log(" renameFile funktioniert korrekt!");

}

async function testRenameFileAutomatically() {
  console.log("Test: renameFileAutomatically sollte Datei basierend auf Kategorie umbenennen");

  // Mock Datei-Existenzprüfung
  fs.access = async () => { };

  // Mock fs.rename
  fs.rename = async () => { };

  const file = {
    path: mockFilePath,
    category: { category: "Bank" }
  };

  const newFileName = await fileService.renameFileAutomatically(file);

  assert.ok(newFileName.startsWith("Bank"), "Dateiname sollte mit Kategorie beginnen");
  console.log(" renameFileAutomatically funktioniert korrekt!");
}

// Tests ausführen
testProcessFile()
  .then(testRenameFile)
  .then(testRenameFileAutomatically)
  .catch(error => console.error(" Fehler während der Tests:", error));


```



```
const assert = require('assert');
const folderController = require('../controllers/folderController');
const fileService = require('../services/fileService');
const fs = require('fs').promises;
const path = require('path');

// **Mocking von `fileService.processFile()`**
```



```
fileService.processFile = async (file) => {
    console.log(` Mock fileService.processFile aufgerufen für Datei:
${file.originalname}`);
    return {
        fileName: file.originalname,
        originalFileName: file.originalname,
        detectedText: "Mock OCR Text",
        ocrMethod: "mock-ocr",
        category: { category: "MockCategory", foundTerm: "MockTerm" },
        processingTime: "0.5",
        folder: path.dirname(file.path)
    };
};

// **Mocking von `fs.readdir()` für Verzeichnisinhalt**
fs.readdir = async (folderPath, options) => {
    console.log(` Mock fs.readdir aufgerufen für Ordner: ${FolderPath}`);
    // Verhindere Endlosschleife: Falls bereits in "subfolder", keine weiteren Unterordner
    // mehr hinzufügen!
    if (FolderPath.includes("subfolder")){
        return [
            { name: "file3.jpg", isFile: () => true, isDirectory: () => false }
        ];
    }

    return [
        { name: "file1.jpg", isFile: () => true, isDirectory: () => false },
        { name: "file2.png", isFile: () => true, isDirectory: () => false },
        { name: "subfolder", isFile: () => false, isDirectory: () => true }
    ];
};

// **Test: processFolder sollte alle Dateien analysieren**
async function testProcessFolder() {
    console.log("Test: processFolder sollte alle Dateien analysieren");

    const folderPath = "/fake/path";

    try{
        const result = await folderController.processFolder(folderPath);

        console.log(` 🔍 Ergebnis von processFolder: ${JSON.stringify(result, null, 2)} `);

        assert.ok(result.includes("file1.jpg"), "file1.jpg sollte verarbeitet werden");
        assert.ok(result.includes("file2.png"), "file2.png sollte verarbeitet werden");
        assert.ok(result.includes("MockCategory"), "Kategorie sollte vorhanden sein");
    }
}
```



```
        console.log(" processFolder funktioniert korrekt!");
    } catch (error) {
        console.error(" Fehler in processFolder:", error);
    }
}

// **Test ausführen**
testProcessFolder();
```



```
const assert = require('assert');
const ocrService = require('../services/ocrService');

async function testProcessImage() {
    console.log("Test: processFile sollte Text aus einem Bild extrahieren");

    // Mock Google Vision API
    ocrService.processFile = async () => ({
        detectedText: "Dies ist ein Testtext",
        ocrMethod: "google-cloud-vision",
        processingTime: "1.5"
    });

    const result = await ocrService.processFile("/fake/path/image.jpg", "image/jpeg");

    assert.strictEqual(result.detectedText, "Dies ist ein Testtext");
    assert.strictEqual(result.ocrMethod, "google-cloud-vision");
    console.log(" processFile funktioniert korrekt!");
}

async function testProcessUnsupportedFile() {
    console.log("Test: processFile sollte Fehler werfen bei nicht unterstütztem Dateityp");

    try {
        // Verwende die echte Funktion, um sicherzustellen, dass der Fehler tatsächlich auftritt
        await ocrService.processFile("/fake/path/file.txt", "text/plain");
        console.error(" Fehler erwartet, aber nicht erhalten!");
    } catch (error) {
        assert.strictEqual(error.message, "Nicht unterstützter Dateityp für OCR.");
        assert.strictEqual(error.statusCode, 415);
        console.log(" Fehler wurde korrekt erkannt!");
    }
}
```



```
// Tests ausführen
```



```
testProcessImage();
testProcessUnsupportedFile();
```

```
/** 
 * Erweiterte Fehlerklasse für eine einheitliche Fehlerbehandlung in der Anwendung.
 * Diese Klasse ermöglicht die Definition eigener Fehler mit einem Statuscode und
optionalen Details.
 */
class AppError extends Error {
    /**
     * Erstellt eine neue Instanz eines anwendungsbezogenen Fehlers.
     *
     * @param {string} message - Die Fehlermeldung.
     * @param {number} statusCode - Der HTTP-Statuscode, der mit diesem Fehler
verbunden ist.
     * @param {string|null} [details=null] - Zusätzliche Details zum Fehler (optional).
     */
    constructor(message, statusCode, details = null) {
        super(message); // Ruft den Konstruktor der Error-Klasse auf
        this.statusCode = statusCode; // Speichert den HTTP-Statuscode
        this.details = details; // Zusätzliche Fehlerdetails (optional)

        // Erstellt eine saubere Stack-Trace für Debugging-Zwecke
        Error.captureStackTrace(this, this.constructor);
    }
}

module.exports = AppError;
```

```
const express = require('express');
const path = require('path');
const cors = require('cors');
const AppError = require('./utils/AppError');

const folderRoutes = require('./routes/folderRoutes');
const fileRoutes = require('./routes/fileRoutes');

const app = express();
const PORT = 3000;

// Middleware-Konfiguration
app.use(cors()); // Aktiviert CORS, um Anfragen von anderen Domains zu ermöglichen
app.use(express.json()); // Ermöglicht das Parsen von JSON-Daten im Request-Body
```



```
app.use(express.urlencoded({ extended: true })); // Unterstützt Form-Data (x-www-form-urlencoded)
app.use(express.static('public')); // Stellt statische Dateien aus dem 'public'-Verzeichnis bereit

// Registriert API-Routen
app.use('/folders', folderRoutes); // Routen für die Ordnerverwaltung
app.use('/files', fileRoutes); // Routen für die Dateioperationen

/**
 * Middleware für nicht gefundene Routen.
 * Falls eine Route nicht existiert, wird eine AppError-Instanz mit Status 404 erzeugt.
 */
app.all('*', (req, res, next) => {
  next(new AppError("Route not found", 404));
});

/**
 * Zentrale Fehlerbehandlung.
 *
 * - Fängt alle Fehler auf, die während der Verarbeitung auftreten.
 * - Loggt den Fehler in der Konsole.
 * - Gibt eine strukturierte JSON-Antwort mit Fehlerdetails zurück.
 */
app.use((err, req, res, next) => {
  console.error("Server error:", err.stack);
  res.status(err.statusCode || 500).json({
    success: false,
    message: err.message,
    details: err.details || null
  });
});

// Startet den Server und gibt die URL in der Konsole aus
app.listen(PORT, () => {
  console.log(`Server läuft unter http://localhost:${PORT}`);
});
```