



HTL | MÖSSINGERSTRASSE

**HÖHERE TECHNISCHE BUNDESLEHRANSTALT**  
KLAGENFURT, MÖSSINGERSTRASSE

ABENDSCHULE INFORMATIK

---

# DIPLOMARBEIT

## KLV goes AI

Einsatz von Künstlicher Intelligenz für die automatisierte  
Betrugserkennung und Schadensbewertung.

## KLV goes AI

Using Artificial Intelligence for Automated  
Fraud Detection and Claims Assessment

JAHRGANG 8AAIF

eingereicht von      Alexander Di Bernardo  
                                 Michael Griesser  
                                 Gordon Telsnig

Projektbetreuer      Peter Gigler, MSc. BSc.

Diese Diplomarbeit entspricht den Standards gemäß dem Leitfaden zur  
Umsetzung der Reife- und Diplomprüfung des BMBWF in der  
letztgültigen Fassung.

Klagenfurt, am 31.03.2025

## EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

Verfasser / Verfasserin

---

Alexander Di Bernardo

---

Michael Griesser

---

Gordon Telsnig

Klagenfurt, am 31.03.2025

## Kurzbeschreibung

Im Projekt „KLV goes AI“ wird untersucht, wie Künstliche Intelligenz (KI) die Kärntner Landesversicherung (KLV) dabei unterstützen kann, Schadensfälle effizienter und schneller zu bearbeiten. Ziel ist es, Sachbearbeiterinnen und Sachbearbeitern durch KI-gestützte Prognosen, automatisierte Deckungsprüfungen und Betrugserkennung wertvolle Hilfestellung zu geben. Dabei kommen moderne Technologien wie React-Frontend, vektorisierte Datenbanken und automatisierte Bild- und Exifdatenanalysen zum Einsatz.

## Aufgabenstellung

Die Versicherungsbranche steht vor der Herausforderung, manuelle und zeitintensive Prozesse zu digitalisieren und zu automatisieren. Ziel dieses Projekts ist es, eine KI-basierte Webanwendung zu entwickeln, die:

- Schadensfälle automatisiert bewertet und deckungsrechtlich prüft,
- Betrugsversuche erkennt,
- eine performante und sichere Dateninfrastruktur auf Basis von Microsoft Azure und Pinecone bereitstellt und
- eine benutzerfreundliche Oberfläche mittels React und TypeScript implementiert.

## Realisierung

Für die Umsetzung wurde eine cloudbasierte Infrastruktur in Microsoft Azure aufgebaut, welche eine MySQL-Datenbank, einen Pinecone-Vektorindex und eine eigene Azure OpenAI-Instanz umfasst. Die Datenintegration und Echtzeitverarbeitung erfolgen über standardisierte API-Schnittstellen. Mit KI-Modellen, unter anderem GPT-4o, werden Schadensmeldungen automatisch auf Versicherungsdeckung geprüft und Schadensbilder ausgewertet. Das React-Frontend bietet eine intuitive, interaktive Benutzeroberfläche zur schnellen und transparenten Bearbeitung.

## Ergebnisse

Es entstand ein voll funktionsfähiger Prototyp, der Versicherungsmitarbeiterinnen und Versicherungsmitarbeitern ermöglicht, Schadensfälle wesentlich effizienter und genauer zu bearbeiten. Die automatische Deckungsprüfung reduziert die Bearbeitungsdauer von Stunden auf Sekunden, wobei die Qualität der Entscheidungen dank KI-Unterstützung gesteigert wird. Darüber hinaus ermöglicht das System durch semantische Suchen und KI-basierte Bildanalysen eine zuverlässige Erkennung von Betrugsversuchen. Die erfolgreiche Implementierung demonstriert eindrucksvoll das Potenzial künstlicher Intelligenz.

**Kurztitel:** KLV goes AI

**Schlüsselwörter:** Künstliche Intelligenz, React-Frontend, Vektorisierte Datenbank, Exifdaten, Schadenanalyse

## Abstract

The project "KLV goes AI" investigates how artificial intelligence (AI) can support Kärntner Landesversicherung (KLV) in processing insurance claims more efficiently and quickly. The aim is to assist claims handlers through AI-driven forecasts, automated coverage checks, and fraud detection. Modern technologies such as React frontend, vectorised databases, and automated image and EXIF data analysis are employed.

## Assignment of Tasks

The insurance industry faces the challenge of digitalizing and automating manual and time-consuming processes. The objective of this project is to develop an AI-based web application capable of:

- Automatically assessing claims and performing coverage checks,
- Detecting fraudulent activities,
- Providing a performant and secure data infrastructure based on Microsoft Azure and Pinecone,
- Implementing a user-friendly interface using React and TypeScript.

## Implementation

A cloud-based infrastructure was set up in Microsoft Azure, including a MySQL database, a Pinecone vector index, and a dedicated Azure OpenAI instance. Data integration and real-time processing are achieved through standardized API interfaces. AI models, notably GPT-4o, automatically check claims for insurance coverage and analyse claim images. The React frontend provides an intuitive, interactive user interface for fast and transparent handling.

## Results

A fully functional prototype was developed, significantly enhancing efficiency and accuracy for insurance personnel when handling claims. The automatic coverage check reduces processing time from hours to seconds, improving decision-making quality through AI support. Additionally, semantic searches and AI-driven image analyses reliably identify fraudulent activities. The successful implementation impressively demonstrates artificial intelligence's potential for the future of the insurance industry and lays the foundation for further digital innovations.

**Short title:** KLV goes AI

**Keywords:** Artificial Intelligence, React frontend, vectorised database, EXIF data, claim analysis

## Inhaltsverzeichnis

1. Einleitung.....	9
2. Grundlagen / Methoden .....	11
2.1. Projektüberblick und eingesetzte Technologien.....	11
2.2. Auswahl der Datenbank: Gründe, Vor und Nachteile (Azure-Integration).....	12
2.2.1. Vorteile von Pinecone .....	12
2.2.2. Nachteile und Herausforderungen bei Pinecone .....	13
2.2.3. Entscheidung für Pinecone .....	13
2.3. Auswahl des KI-Anbieters: Vergleiche und Benchmarks .....	13
2.3.1. Anbieter und Modelle .....	15
2.3.2. Benchmarks.....	15
2.3.3. Kontextlänge .....	17
2.3.4. Geschwindigkeit.....	18
2.3.5. Preis.....	19
2.3.6. Vor und Nachteile .....	20
2.3.7. Konklusion.....	21
2.4. Auswahl der Web-Frontends: Vergleich von Frameworks .....	21
2.4.1. Framework-basierte Entwicklung vs. JavaScript.....	21
2.4.1.1. Framework-basierte Entwicklung .....	21
2.4.1.2. Vorteile von Frameworks .....	21
2.4.1.3. Nachteile von Frameworks .....	22
2.4.2. Vergleich verschiedener Frameworks .....	23
2.4.2.1. React [22] .....	23
2.4.2.2. Angular [23].....	24
2.4.2.3. Vue.js [24].....	25
2.4.2.4. Svelte [29].....	26
2.4.2.5. Ember [30].....	27
2.4.2.6. Konklusion .....	28
2.4.2.7. Exchangeable Image File Format (EXIF) Daten .....	29
2.4.2.8. Nutzbarkeit von EXIF-Daten .....	29
2.4.2.9. Fiktives Fallbeispiel.....	29
2.4.2.10. Funktion eines ExifReaders .....	30

2.4.2.11. Auswahl eines EXIF-Readers .....	30
2.4.2.12. Konklusion .....	30
2.5. Datenschutzrechtliche Überlegungen (DSGVO) in der Versicherungsbranche .....	31
3. Implementierung einer vektorisierten Datenbank für existierende Schadensfälle sowie einer API-Schnittstelle in einer cloudbasierten Infrastruktur (Alexander Di Bernardo) .....	33
3.1. Aufgabenstellung .....	33
3.2. Grundlagen / Methoden .....	34
3.2.1. Pinecone – Vektordatenbank .....	34
3.2.2. Migration der Datenbankbestände .....	35
3.2.3. Aufbau der zentralen Regeldatenbank .....	35
3.2.4. Datenfluss in der Deckungsprüfung .....	35
3.3. Realisierung / Implementierung .....	36
3.3.1. Architektur der Deckungsprüfung .....	36
3.3.2. Datenbank <i>viva_regelwerke</i> (Versicherungsregelwerke) .....	38
3.3.3. Batch-Prozesse zur Pinecone-Aktualisierung .....	40
3.3.4. Ablauf der Deckungsprüfung (API-Endpunkte) .....	42
3.3.4.1. Endpunkt „/claimcoverage“ .....	42
3.3.4.2. Endpunkt „/claimimages“ .....	46
3.3.5. Dokumentenverarbeitung und AI-gestützte Zusammenfassung .....	47
3.4. Ergebnisse / Konklusion .....	48
3.4.1. Erreichte Funktionalitäten .....	48
3.4.2. Leistungsbewertung .....	49
3.4.3. Mögliche Erweiterungen für die Zukunft .....	50
3.4.4. Zusammenfassung .....	51
4. Konfiguration einer cloudbasierten Projektinfrastruktur, sowie Implementierung eines KI-Systems mit zugehöriger API-Schnittstelle. (Michael Griesser) .....	52
4.1. Aufgabenstellung .....	52
4.2. Realisierung / Implementierung .....	52
4.2.1. Azure Cloud .....	52
4.2.2. Server .....	57
4.2.3. Bereitstellung des FastAPI Backends .....	57
4.2.4. Bereitstellung des React Frontends .....	58

4.2.5. Bereitstellung der Regel Eingabeseite .....	59
4.3. Implementierung.....	59
4.3.1. API .....	59
4.3.2. KI gestützte Bildanalyse .....	61
4.3.3. Beispiel.....	68
4.3.4. Verbesserungen .....	69
5. Implementierung eines Frontend für Sachbearbeiter/innen zur Darstellung und Analyse der Ergebnisse. (Gordon Telsnig) .....	71
5.1. Aufgabenstellung .....	71
5.2. Evaluierung der darzustellenden Daten und Anwendungsfälle für KI-Systeme .....	71
5.3. Bewertung der Sachbearbeiterinnen und Sachbearbeiter Anforderungen .....	71
5.4. Zusammenfassung der Anforderungen.....	72
5.5. Überblick und Architektur .....	72
5.5.1. Welcomescreen .....	74
5.5.2. Section 1 – Übersicht .....	74
5.5.3. Section 2 – Detaillierte EXIFAnalyse .....	75
5.5.4. Auslesen von Exifdaten:.....	76
5.5.5. Section 3 – KIReport und Dokumentenanzeige .....	77
5.5.6. Section 4 – Deckungsprüfung .....	78
5.6. Routing und Zugriffsschutz.....	79
5.7. Globale Zustände und Context.....	81
5.8. React Hooks [58] .....	82
5.8.1. useEffect() Hook.....	82
5.8.2. useState() Hook [58] .....	83
5.8.3. useReducer() Hook [59] .....	84
5.8.4. React.Memo [59] .....	86
5.9. TypeScript in React-Anwendungen [60].....	87
5.10. Erreichte Funktionalitäten .....	88
5.10.1. Section 1 – Übersicht und Datendarstellung .....	88
5.10.2. Section 2– Detaillierte EXIF-Datenanalyse.....	88
5.10.3. Section 3 – KI-Analyse und Dokumentenmanagement .....	89
5.10.4. Section 4 – Dokumentenmanagement und integrierter Chat.....	89
5.11. Leistungsbewertung .....	89

5.12. Mögliche Erweiterungen für die Zukunft .....	90
5.13. Konklusion .....	90
6. Projektmanagement.....	91
6.1. Aufgabenstellung des Gesamtprojekts .....	91
6.2. Herausforderungen und Lösungsansätze.....	91
6.2.1. Verzögerungen durch Datenzugang .....	91
6.2.2. Einschränkungen durch externe Vorgaben .....	91
6.2.3. Effektive Kommunikation .....	91
6.2.4. Fazit und Lessons Learned .....	92
6.3. Scrum-Projektplan.....	92
6.3.1. Michael Griesser .....	93
6.3.2. Alexander Di Bernardo.....	94
6.3.3. Gordon Telsnig.....	97
6.4. Inbetriebnahme.....	100
6.4.1. Features .....	100
6.4.2. Installation & Setup .....	100
6.4.3. Konfiguration .....	101
6.4.4. Verzeichnisstruktur .....	101
6.4.5. Nutzung der API .....	101
6.4.6. Tests .....	102
6.4.7. Weiterentwicklung & Contributing.....	102
6.5. Kostenaufstellung.....	103
6.6. Besprechungsprotokolle .....	104
6.7. Arbeitszeitnachweis .....	112
6.7.1. Alexander Di Bernardo.....	112
6.7.2. Michael Griesser .....	115
6.7.3. Gordon Telsnig.....	117
7. Literaturverzeichnis.....	120
8. Abbildungsverzeichnis.....	126
9. Tabellenverzeichnis .....	127
10. Listings.....	127

## 1. Einleitung

Die Versicherungsbranche sieht sich aktuell mit erheblichen Herausforderungen im Bereich der Digitalisierung konfrontiert. Besonders deutlich wird dies bei der Abwicklung von Schadensfällen, wo traditionelle, manuelle Arbeitsprozesse oftmals zeitintensiv, fehleranfällig und ineffizient sind. Diese Situation belastet nicht nur die internen Ressourcen der Versicherungsunternehmen, sondern beeinträchtigt auch maßgeblich die Zufriedenheit ihrer Kundinnen und Kunden, die schnelle und transparente Entscheidungen erwarten.

Vor diesem Hintergrund hat die Kärntner Landesversicherung (KLV) gemeinsam mit Schülern der HTL Mössingerstraße das Projekt „**KLV goes AI**“ ins Leben gerufen. Ziel dieses Projekts ist es, mittels Einsatz moderner KI-Technologien die Bearbeitung von Schadensfällen effizienter, schneller und genauer zu gestalten.

Die konkrete Problemstellung sowie der Impuls zu diesem Projekt stammen vom Auftraggeber, der Kärntner Landesversicherung. Ansprechpartner auf Seiten der KLV ist Herr Ing. Harald Klary, Bakk. MSc., Leiter der IT-Abteilung. Die Projektumsetzung erfolgte im Rahmen einer Diplomarbeit, die von Alexander Di Bernardo – Mitarbeiter und Produktmanager der KLV – gemeinsam mit seinen Kollegen Michael Griesser und Gordon Telsnig ausgearbeitet wurde. Unterstützend stand ihnen dabei seitens der HTL Mössingerstraße Herr Peter Gigler, MSc. BSc., als betreuender Lehrer zur Seite.

Die Zielsetzung des Gesamtprojekts lässt sich wie folgt zusammenfassen:

- Entwicklung einer KI-gestützten Anwendung zur automatisierten Deckungsprüfung, um eingehende Schadensmeldungen in Echtzeit auf Versicherungsdeckung zu analysieren.
- Integration eines Systems zur intelligenten Betrugserkennung durch semantische Analysen von Schadensbeschreibungen sowie automatisierte Auswertungen von Bild- und Metadaten (Exif-Daten).
- Aufbau einer sicheren und leistungsfähigen Cloud-Infrastruktur auf Basis von Microsoft Azure, in Kombination mit einer Pinecone-Vektordatenbank und der Nutzung einer Azure OpenAI-Instanz.
- Implementierung einer intuitiven Benutzeroberfläche auf Grundlage moderner Technologien (React und TypeScript), um Sachbearbeiterinnen und Sachbearbeitern eine benutzerfreundliche und schnelle Bearbeitung der Schadenfälle zu ermöglichen.

Die technische und organisatorische Struktur des Systems ist in Abbildung 1.1 übersichtlich dargestellt. Dabei werden auch die Zuständigkeitsbereiche der einzelnen Projektmitglieder hervorgehoben: Alexander Di Bernardo verantwortete die Implementierung der Vektordatenbank und die API zur automatisierten Deckungsprüfung, Michael Griesser war zuständig für

die cloudbasierte Infrastruktur sowie die Integration der KI-Services, und Gordon Telsnig setzte die Frontend-Anwendung auf Basis von React und TypeScript um.

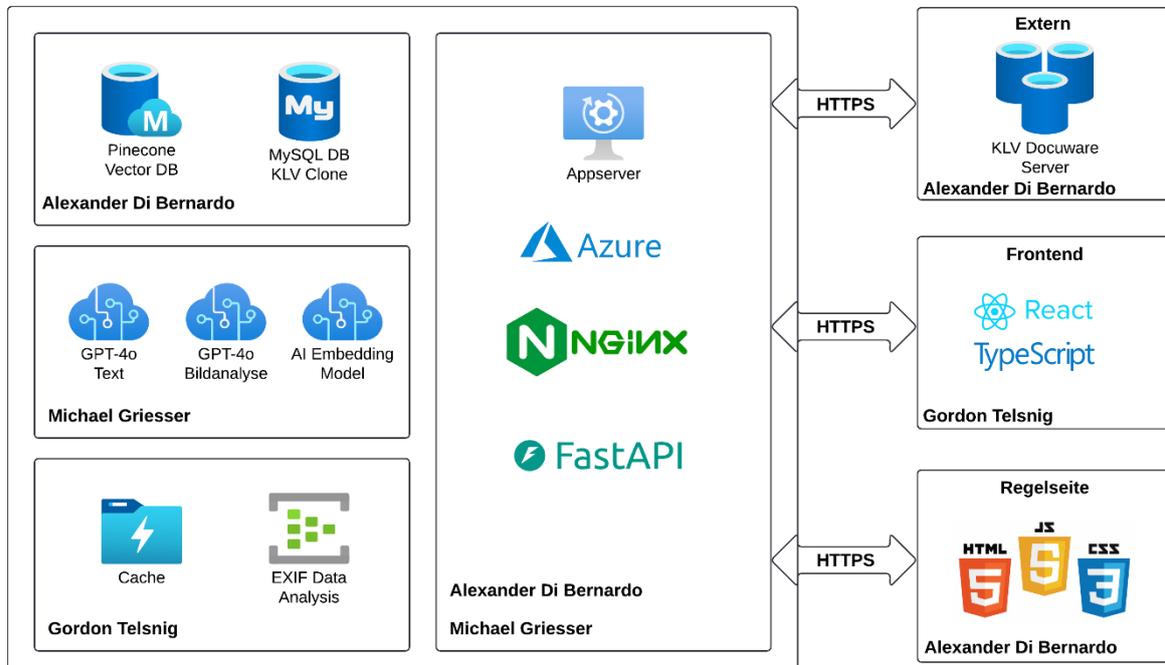


Abbildung 1.1 Systemstruktur

Die vorliegende Diplomarbeit gliedert sich in mehrere Abschnitte: Zunächst werden im Kapitel „Grundlagen und Methoden“ (Kapitel 2) die technologischen Entscheidungen nachvollziehbar dargestellt. Daraufhin folgen die detaillierte Beschreibung der Implementierung der Vektordatenbank und API-Schnittstellen (Kapitel 3), die Konfiguration der Azure-basierten Infrastruktur und Implementierung der KI-Komponenten (Kapitel 4) sowie die Evaluierung und Entwicklung des React-Frontends zur Exif-Datenanalyse und Dokumentenverwaltung (Kapitel 5).

Durch dieses Projekt wird deutlich, wie KI-Technologien die Versicherungsbranche nachhaltig verändern können. „KLV goes AI“ zeigt beispielhaft, wie Digitalisierung und Automatisierung nicht nur interne Prozesse beschleunigen, sondern auch entscheidend dazu beitragen, die Wettbewerbsfähigkeit und Kundenzufriedenheit langfristig zu sichern.

## 2. Grundlagen / Methoden

Dieses Kapitel vermittelt die notwendigen **Grundlagen** und stellt die verwendeten **Methoden** des Diplomarbeitprojekts vor. Zunächst wird ein **Überblick** über die eingesetzten Technologien und deren Zusammenspiel gegeben. Darauf folgt eine detaillierte Darstellung und Begründung der getroffenen **technologischen Entscheidungen**. Die erläuterten Konzepte und Technologien bilden die Basis für eine effiziente, nachhaltige und DSGVO-konforme Umsetzung des Projekts.

### 2.1. Projektüberblick und eingesetzte Technologien

Die Umsetzung erfolgt durch eine Kombination innovativer Technologien, die gezielt ausgewählt wurden, um den spezifischen Anforderungen der Versicherungsbranche gerecht zu werden. Zentrales Element des Projekts ist eine **Cloud-basierte Infrastruktur**, welche auf Basis von **Microsoft Azure** umgesetzt wurde. Diese Infrastruktur bietet die notwendige Skalierbarkeit, Sicherheit und Stabilität, um die Anforderungen eines Versicherungsunternehmens wie der Kärntner Landesversicherung vollumfänglich zu erfüllen.

Eine zentrale Komponente ist dabei die Vektor-Datenbank **Pinecone**, die speziell für die semantische Suche und die Speicherung hochdimensionaler Vektoren entwickelt wurde. Durch den Einsatz von **Pinecone** können große Mengen unstrukturierter Daten, insbesondere Schadensmeldungen und Versicherungsbedingungen, schnell und semantisch analysiert werden. Dies ermöglicht eine automatisierte **Deckungsprüfung**, bei der KI-gestützte Modelle Schadensmeldungen mit Vertragsbedingungen abgleichen und in Echtzeit Entscheidungsvorschläge generieren.

Zur Umsetzung der KI-Analysen und automatisierten Prozesse werden leistungsfähige KI-Modelle eingesetzt, insbesondere das von Microsoft angebotene Modell **GPT-4o** über die **Azure OpenAI-Instanz**. Diese KI-Komponenten ermöglichen eine effiziente **Betrugserkennung**, indem Schadensbeschreibungen, Dokumente sowie Bild- und Metadaten (wie beispielsweise **EXIF-Daten**) analysiert und auf mögliche Auffälligkeiten geprüft werden.

Die Benutzeroberfläche der entwickelten Anwendung basiert auf einem modernen Web-Frontend, umgesetzt mit den Technologien **React** und **TypeScript**. Diese Wahl garantiert sowohl eine hohe Benutzerfreundlichkeit als auch eine optimale Wartbarkeit und Skalierbarkeit. Das Frontend erlaubt Sachbearbeiterinnen und Sachbearbeitern, die automatisiert gewonnenen Erkenntnisse intuitiv nachzuvollziehen und in ihre Entscheidungsprozesse einzubeziehen. Schadensmeldungen, Analyseergebnisse und Empfehlungen werden übersichtlich dargestellt, was die Effizienz und Genauigkeit der Bearbeitung erhöht.

Zusätzlich wurde eine speziell entwickelte **REST-API** implementiert, die als Schnittstelle zu bestehenden externen Systemen der KLV, beispielsweise dem Dokumentenmanagementsystem **DocuWare**, dient. Diese API sorgt für eine nahtlose Integration der entwickelten KI-Lösung in

die bestehende IT-Landschaft der Kärntner Landesversicherung und ermöglicht eine umfassende Automatisierung bestehender Workflows.

In den folgenden Abschnitten werden zunächst die Hintergründe zur Auswahl der **verwendeten Technologien** und deren Vor- und Nachteile detailliert erläutert. Anschließend werden wichtige datenschutzrechtliche Überlegungen gemäß der **DSGVO** vorgestellt, die insbesondere bei der Verarbeitung personenbezogener Versicherungsdaten zu berücksichtigen sind.

## 2.2. Auswahl der Datenbank: Gründe, Vor und Nachteile (Azure-Integration)

Für die semantische Suche und KI-gestützte Analyse von Dokumenten und Regeln fiel die Wahl auf **Pinecone** als Vektordatenbank. Pinecone ist ein spezialisiertes Cloud-Datenbanksystem zur **Speicherung und schnellen Abfrage hochdimensionaler Vektoren**, typischerweise von Text- oder Dokumentembeddings. Damit lassen sich unstrukturierte Daten (z.B. Textbausteine, Dokumenteninhalte oder Geschäftsregeln) so ablegen, dass ein KI-Modell relevante Informationen bei Bedarf schnell auffinden kann. Dies ist insbesondere wichtig, um **große Textmengen oder Regelwerke effizient zu durchsuchen** und passenden Kontext für KI-Abfragen bereitzustellen – ein Ansatz, der sogenannte *Retrieval-Augmented Generation (RAG)*, hilft z.B. Halluzinationen von Sprachmodellen zu reduzieren, indem stets die relevantesten Wissensdaten geliefert werden. [1]

### 2.2.1. Vorteile von Pinecone

Im Projektkontext sind vor allem in Performance und einfacher Integration zu sehen. Pinecone wurde als vollständig verwalteter Dienst gewählt, der seit 2023 auch direkt auf Microsoft Azure verfügbar ist. Durch die **Bereitstellung in Azure-Rechenzentren** kann Pinecone nahe an den restlichen Anwendungen und Daten betrieben werden, was **geringe Latenzen** für Vektor-Abfragen ermöglicht. Zudem entfällt komplexe eigene Infrastrukturverwaltung, da Pinecone automatisch skaliert und als *Serverless*-Dienst nach Verbrauch abgerechnet wird. [2] [1] Dies führt zu einer **flexiblen und potenziell kosteneffizienten Nutzung** im Vergleich zu manchen Azure-eigenen Lösungen. Während Azure-Angebote wie Cosmos DB sich nahtlos in das Microsoft-Ökosystem einfügen und hohe Skalierbarkeit bieten, überzeugt Pinecone durch **entwicklerfreundliche API und budgetfreundliche Skalierung**. [3] In der Praxis erwies sich Pinecone auch als **sehr einfach in die bestehende Anwendung zu integrieren**, da es bereits Bibliotheken und Beispiele (u.a. für Python und Node.js) gibt und mit gängigen KI-Frameworks (z.B. LangChain) kompatibel ist.

Nicht zuletzt hat Pinecone im Bereich **Versicherungen** bereits erfolgreiche Anwendungsfälle, was die Entscheidung untermauerte. So wird Pinecone etwa von InsurTech-Unternehmen eingesetzt, um **umfangreiche Versicherungsdokumente und historische Schadendaten semantisch zu durchsuchen** und relevante Informationen für Underwriting-Entscheidungen bereitzustellen. [4] Diese Fähigkeit, **sowohl strukturierte als auch unstrukturierte Daten schnell nach inhaltlicher Ähnlichkeit zu durchsuchen**, ist ein entscheidender Vorteil bei der Deckungsprüfung, wo textuelle Vertragsbedingungen mit Schadensschilderungen abgeglichen werden müssen.

### 2.2.2. Nachteile und Herausforderungen bei Pinecone

Diese lagen hauptsächlich in der anfänglichen Azure-Integration und Datenhaltung. Zum Projektstart war Pinecone auf Azure noch neu (im Early Access verfügbar [1]), sodass zusätzliche Konfiguration nötig war, um es mit der Azure-Umgebung zu verbinden. Obwohl Pinecone inzwischen im Azure Marketplace verfügbar ist, handelt es sich um einen **Drittanbieter-Service**, was bedeutet, dass z.B. für die Nutzung ein separater API-Schlüssel verwaltet und der Dienst außerhalb des eigenen Azure-Abonnements abgerechnet wird. Dies kann den Verwaltungsaufwand erhöhen und erfordert vertrauenswürdige Verbindungen (z.B. absichern der API-Keys, Konfiguration von Netzwerkroutern). Ferner konzentriert sich Pinecone rein auf Vektor-Index und -Suche; relationale Abfragen oder Transaktionen auf Geschäftsobjekte sind damit nicht möglich. Im Projekt wurde dies dadurch gelöst, dass **Pinecone nur als ergänzender Suchindex** verwendet wird, während **persistente Geschäftsdaten weiterhin in der MySQL-Datenbank** liegen. Ein alternatives Azure-eigenes Produkt wäre z.B. **Azure Cognitive Search** mit neuer Vektor-Suche-Funktionalität gewesen. Dieses lässt sich noch enger in Azure AD und übrige Dienste integrieren und bietet zusätzlich klassische Volltextsuche-Funktionen (z.B. Autocomplete, Synonym-Maps). [5] Allerdings war zum Projektzeitpunkt die Vektor-Suche in Azure Cognitive Search noch weniger erprobt, und Pinecone versprach eine schnellere Implementierung mit bestehenden KI-Workflows. Zudem hat Pinecone durch spezielle Algorithmen (Approximate Nearest Neighbour – ANN) eine sehr hohe Abfragegeschwindigkeit auch bei wachsendem Datenvolumen, wie praktische Tests zeigten.

### 2.2.3. Entscheidung für Pinecone

Zusammenfassend fiel die Entscheidung auf Pinecone, **um die semantische Suche nach Ähnlichkeiten in Textdaten optimal umzusetzen**, ohne sich durch die Komplexität eines selbst aufzubauenden Vektorindexes aufzuhalten. Die Vorteile – **hohe Performance, schnelle Entwicklungsintegration und seit 2024 auch Azure-native Verfügbarkeit** – überwogen die potenziellen Nachteile. Für künftige produktive Umsetzungen sollte jedoch geprüft werden, ob Pinecone langfristig die **Compliance-Anforderungen** (z.B. bzgl. EU-Datenspeicherung, Identity-Management) erfüllt oder ob eine **Azure-native Lösung** vorzuziehen ist, falls diese funktional aufholt.

## 2.3. Auswahl des KI-Anbieters: Vergleiche und Benchmarks

Die rasante Entwicklung der Künstlichen Intelligenz (KI) in den letzten Jahren hat zu einer Vielzahl von Modellen geführt, die in unterschiedlichsten Anwendungsbereichen der Wirtschaft eingesetzt werden. Von der Automatisierung einfacher Aufgaben bis hin zur Lösung komplexer Probleme – KI-Modelle spielen eine immer wichtigere Rolle in unserer modernen Gesellschaft. Insbesondere Sprachmodelle, die in der Lage sind, menschliche Sprache zu verstehen und zu generieren, haben erhebliche Fortschritte gemacht und bieten vielfältige Einsatzmöglichkeiten in zahlreichen Branchen, von der Kundenbetreuung über die Marktforschung bis hin zur Prozessoptimierung.

Ein besonders beeindruckender Bereich ist die Medizin, wo KI-Modelle mittlerweile grundlegende Fortschritte ermöglichen. So hat die Anwendung von Deep-Learning-Algorithmen in der

Vorhersage von Proteinfaltungen – ein entscheidender Prozess in der biomedizinischen Forschung – zu Durchbrüchen in der Medikamentenentwicklung und der personalisierten Medizin geführt. [6]

Darüber hinaus revolutioniert KI auch den Kundenservice. Moderne Chatbots, die auf fortschrittlichen Sprachmodellen basieren, ermöglichen es Unternehmen, rund um die Uhr auf Kundenanfragen zu reagieren. Diese Chatbots bieten nicht nur schnelle und präzise Antworten, sondern lernen auch kontinuierlich dazu, um den Service weiter zu optimieren und die Zufriedenheit der Kundinnen und Kunden nachhaltig zu steigern. Neben diesen Bereichen findet KI zudem Anwendung in der Finanzanalyse, im industriellen Produktionsmanagement und in der Optimierung von Logistik- und Lieferketten, wodurch Prozesse effizienter gestaltet und strategische Entscheidungen fundiert getroffen werden können. [7]

In den letzten Jahren haben Fortschritte im Bereich der künstlichen Intelligenz, insbesondere durch den Einsatz tiefer neuronaler Netze, auch die Bildanalyse revolutioniert. Moderne Algorithmen ermöglichen es, visuelle Informationen mit hoher Genauigkeit und Geschwindigkeit zu verarbeiten. Diese Fortschritte führen zu einer signifikanten Verbesserung der Erkennung und Klassifizierung von Schadensmustern, was insbesondere in der Versicherungsbranche von großem Nutzen sein kann.

Die automatische Bildanalyse bietet Versicherungen zahlreiche Vorteile bei der Bearbeitung von Schadensfällen. Durch die Integration von KI-basierten Lösungen können eingereichte Schadensbilder in Echtzeit analysiert werden, wodurch die Schadensbewertung objektiver und konsistenter erfolgt. Dies reduziert nicht nur den manuellen Aufwand und die Bearbeitungszeit, sondern minimiert auch menschliche Fehler und ermöglicht eine schnellere Entscheidungsfindung. Darüber hinaus tragen präzise Analysen zur besseren Erkennung von Betrugsfällen bei, da Anomalien und Unregelmäßigkeiten in den Bildern frühzeitig identifiziert werden können. Insgesamt ermöglicht die automatisierte Bildanalyse eine effizientere und kostengünstigere Abwicklung von Schadensfällen, was letztlich zu einer verbesserten Kundenzufriedenheit und einer optimierten internen Prozessstruktur führt.

Dieser Teil der Arbeit befasst sich mit der detaillierten Analyse und dem Benchmarking aktueller KI-Modelle, um deren Leistungsfähigkeit und Anwendbarkeit in der Wirtschaft umfassend zu bewerten. Im Fokus stehen dabei Modelle von renommierten Anbietern wie OpenAI und Anthropic. Des Weiteren wird auch das Open-Source-Modell Llama 3.1 untersucht, um ein breites Spektrum der verfügbaren Technologien abzudecken und deren Potenziale und Einschränkungen sichtbar zu machen.

Das Ziel ist es, die verschiedenen Modelle systematisch zu vergleichen, ihre Stärken und Schwächen zu identifizieren und herauszuarbeiten, welche Modelle sich besonders für spezifische Anwendungen in der Wirtschaft eignen. Dabei werden sowohl technische Leistungsparameter als auch praktische Anwendungsszenarien berücksichtigt, um eine fundierte Bewertung vorzunehmen.

### 2.3.1. Anbieter und Modelle

In diesem Abschnitt werden wir die zwei führenden KI-Anbieter und eine Open-Source Variante genauer betrachten. Anhand von Benchmarks wird die Leistungsfähigkeit der Modelle festgestellt.

#### OpenAI – GPT-4o

OpenAI ist ein führendes Forschungsunternehmen im Bereich der Künstlichen Intelligenz und der Entwicklung von fortschrittlicher Sprachmodellen. Eines der neuesten Modelle von OpenAI ist GPT-4o. GPT-4o, oder Generative Pre-trained Transformer 4o, ist ein leistungsfähiges Sprachmodell, das darauf trainiert ist, menschenähnlichen Text zu verstehen und zu generieren. Es nutzt eine tiefe neuronale Netzarchitektur, die es ihm ermöglicht, komplexe Zusammenhänge in großen Textmengen zu erkennen und darauf basierend kohärente und relevante Antworten zu generieren. [8]

#### Anthropic – Claude 3.5 Sonnet

Anthropic ist ein aufstrebendes Forschungsunternehmen im Bereich der Künstlichen Intelligenz, das sich auf die Entwicklung sicherer und verständlicher KI-Systeme konzentriert. Eines ihrer fortschrittlichen Modelle ist Claude 3.5 Sonnet, das speziell entwickelt wurde, um sicherere und ethisch verantwortungsvollere KI-Anwendungen zu ermöglichen. Es wurde mit einem starken Fokus auf Sicherheit, Transparenz und ethische Nutzung entwickelt. [9] [10]

#### (META) Open Source – Llama3.1 70B

Neben den proprietären Modellen von Unternehmen wie OpenAI und Anthropic spielen Open-Source-Modelle eine bedeutende Rolle in der KI-Landschaft. Sie bieten eine kosteneffiziente und flexible Alternative, die von der globalen Entwicklergemeinschaft genutzt und weiterentwickelt werden kann. Diese laufen, im Vergleich zu den proprietären Modellen, lokal und gewährleisten somit die Hoheit über die Daten. Llama3.1 wurde von Meta (Facebook) entwickelt und verwendet eine erweiterte Transformer-Architektur, die eine effiziente Verarbeitung großer Textmengen ermöglicht. Es unterstützt sowohl die Verarbeitung von Text als auch die Generierung von menschenähnlichem Text und kann für verschiedene NLP-Aufgaben angepasst werden. Open Source Modelle können mit bestimmter Software, wie z.B. Ollama, lokal betrieben werden. [11]

### 2.3.2. Benchmarks<sup>1</sup>

Um KI-Modelle miteinander vergleichen zu können, werden domänenspezifische Tests und standardisierte Benchmarks erstellt, die eine objektive Evaluierung ihrer Leistungsfähigkeit ermöglichen. Benchmarks spielen eine zentrale Rolle in der KI-Forschung, da sie als einheitlicher Maßstab dienen, um die Genauigkeit, Robustheit und Effizienz verschiedener Modelle zu mes-

---

<sup>1</sup> ChatGPT (GPT-4o), Datum der Abfrage: 05.03.2025, Prompt: „Verbessere die Formulierungen und Satzstellungen“

sen. Hierbei werden spezifische Aufgaben definiert, die repräsentativ für die jeweilige Domäne sind, und es werden klare Metriken festgelegt, an denen die Modelle bewertet werden können. Dieser strukturierte Vergleich hilft nicht nur dabei, die Stärken und Schwächen einzelner Ansätze zu identifizieren, sondern fördert auch den technologischen Fortschritt, indem er als Grundlage für Weiterentwicklungen und Optimierungen dient. Darüber hinaus werden Benchmarks häufig von unabhängigen Institutionen oder Forschungskonsortien entwickelt und regelmäßig aktualisiert, um den aktuellen Stand der Technik sowie veränderte Anforderungen in der Praxis abzubilden.

Als Beispiel hierfür wird der MMLU-Pro Benchmark genannt. Dieser Test ist sehr breit gefächert und beinhaltet akademische Fragen aus 14 verschiedenen Bereichen, darunter Geschichte, Mathematik und Informatik. Zu jeder Frage gibt es 10 Antwortmöglichkeiten, aus denen die KI die richtige Option auswählen muss. Der MMLU-Pro Benchmark ermöglicht somit eine umfassende Bewertung der Wissensbasis und Problemlösungsfähigkeiten von KI-Modellen, indem er deren Fähigkeit testet, in unterschiedlichen akademischen Disziplinen präzise und konsistente Antworten zu liefern. [12]

Weitere Benchmarks zur Index Berechnung:

- GPQA Diamond
- Humanity's Last Exam
- LiveCodeBench
- SciCode
- AIME
- MATH-500

Der Artificial Analysis Intelligence Index ist eine umfassende Bewertungsmethode für die Fähigkeiten von Sprachmodellen, die im Februar 2025 in Version 2.0 veröffentlicht wurde. Der Index kombiniert acht verschiedene Evaluierungsdatensätze, um die Leistung in den Bereichen Reasoning, Wissen, Mathematik und Programmierung zu bewerten. Die Methodik legt Wert auf Fairness und praktische Anwendbarkeit, mit einem geschätzten 95%-Konfidenzintervall von weniger als  $\pm 1\%$ . Der Index besteht aus verschiedenen Komponenten, darunter der Math Index und der Coding Index, die jeweils spezifische Fähigkeiten messen. Die Evaluierung erfolgt nach vier Kernprinzipien: standardisiert, unvoreingenommen, Zero-Shot-Instruktion und transparent. Die Bewertung wird mit einer Gewichtung von 50% für allgemeines Reasoning und Wissen, 25% für mathematisches Reasoning und 25% für Code-Generierung berechnet. Die Tests werden unter kontrollierten Bedingungen durchgeführt, einschließlich spezifischer Temperatureinstellungen und Fehlerbehandlungsprotokolle. [13]

### Artificial Analysis Intelligence Index

Intelligence Index incorporates 7 evaluations spanning reasoning, knowledge, math & coding

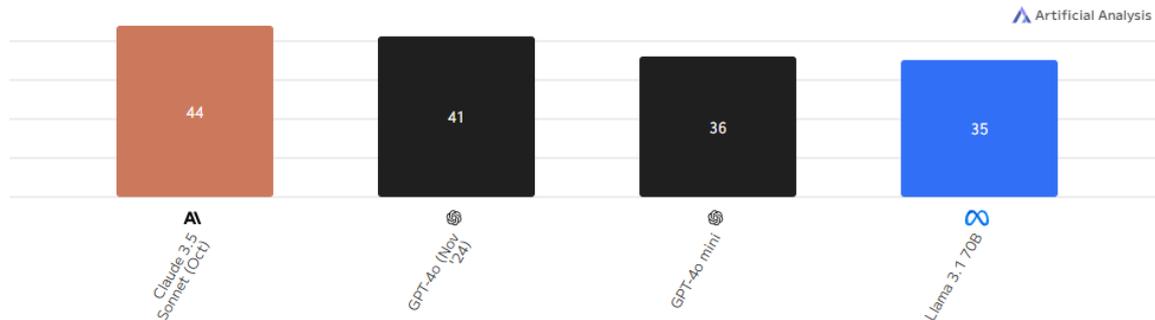


Abbildung 2.1 Intelligenz Benchmark [14]

Wie in Abbildung 4.1 zu sehen ist schneidet Anthropic bei diesem Benchmark am besten ab. Die Open Source Modelle haben in den letzten Monaten ordentlich an Leistung zugelegt und sind mittlerweile auf einem Niveau, um einfachere Aufgaben übernehmen zu können.

### 2.3.3. Kontextlänge

Die Kontextlänge von LLMs definiert, wie viele Token – also Wörter, Satzzeichen und Teile von Wörtern – das Modell in einem Durchlauf berücksichtigen kann. Im Schnitt enthält ein Token, je nach Sprache, 4 ASCII Zeichen. [15] Diese Grenze ist entscheidend dafür, wie viel zusammenhängende Information verarbeitet und in eine sinnvolle Antwort eingebettet werden kann. Überschreitet die Eingabe diesen Rahmen, müssen ältere Teile des Textes abgeschnitten oder weniger gewichtet werden, was die Ausgabe beeinträchtigen kann. Eine ausreichende Kontextlänge ermöglicht es dem Modell, längere Dokumente oder komplexe Dialoge zu analysieren, ohne wichtige Zusammenhänge zu verlieren. Somit ist die Kontextlänge ein zentraler Faktor für die Leistungsfähigkeit und Anwendungsbreite von LLMs.

Sollte die Kontextlänge nicht ausreichend sein, kann man, sofern möglich, die bisherigen Daten von der KI zusammenfassen lassen und als neuen Prompt übergeben. Auf diese Weise kann die Kontextlänge künstlich gestreckt werden. Die Präzision der Antworten leidet dadurch aber.

## Context Window

Context Window: Tokens Limit; Higher is better

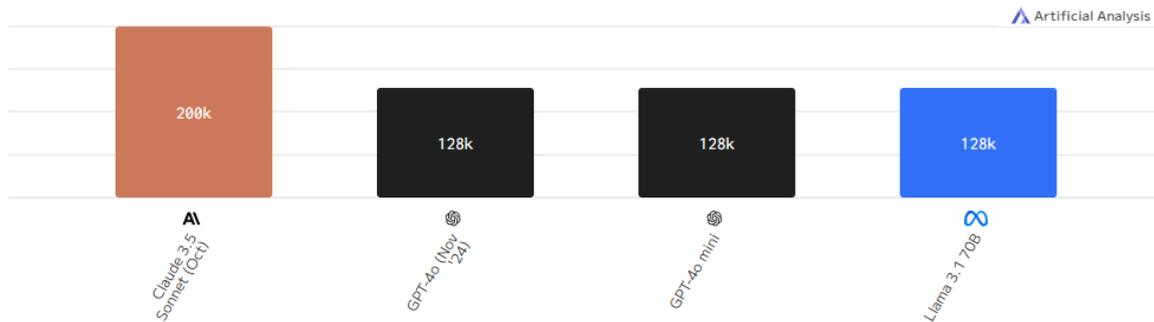


Abbildung 2.2 Kontextlänge [14]

Hier bietet Anthropic mit Claude das größte Kontextfenster, welches 200.000 Tokens umfasst. (Abbildung 4.2) OpenAI und Llama bieten beide gleich viel. Die Kontextlänge von GPT-4o mit 128.000 Tokens entspricht dem äquivalent von ca. 300 Seiten Text.

### 2.3.4. Geschwindigkeit

Ein wichtiger Begriff in der KI-Domäne ist „Inferenz“ [16] und bezieht sich auf den Prozess, in dem ein trainiertes KI-Modell auf Basis von Eingabedaten Vorhersagen trifft oder Entscheidungen generiert. In dieser Phase wird das während des Trainings erworbene Wissen angewendet, um beispielsweise Texte zu vervollständigen oder Antworten in Echtzeitanwendungen zu liefern. Die Geschwindigkeit der Inferenz ist daher ein wesentlicher Faktor, der die praktische Nutzbarkeit und Reaktionszeit des Modells maßgeblich bestimmt.

Sie wird häufig in Tokens pro Sekunde (Tokens/sec) gemessen und ist entscheidend für performante Anwendungen. Die Leistung hängt stark von der Modellgröße, der verwendeten Hardware und der Optimierung der Inferenzpipeline ab, wobei spezialisierte Beschleuniger, wie Grafikkarten oder spezielle Inferenzchips, die Verarbeitung erheblich verbessern können.

Mit diesem Benchmark wird die Geschwindigkeit gemessen, in der das Modell seine Antwort generiert. In der Regel gilt, je intelligenter ein Modell, desto langsamer wird es. Die Anbieter versuchen mit mehr Hardwareressourcen dieses Manko auszugleichen. Den schnellsten KI Inferenz Cluster besitzt momentan Elon Musks mit seinem Unternehmen „xAI“. [17]

### Output Speed

Output Tokens per Second; Higher is better

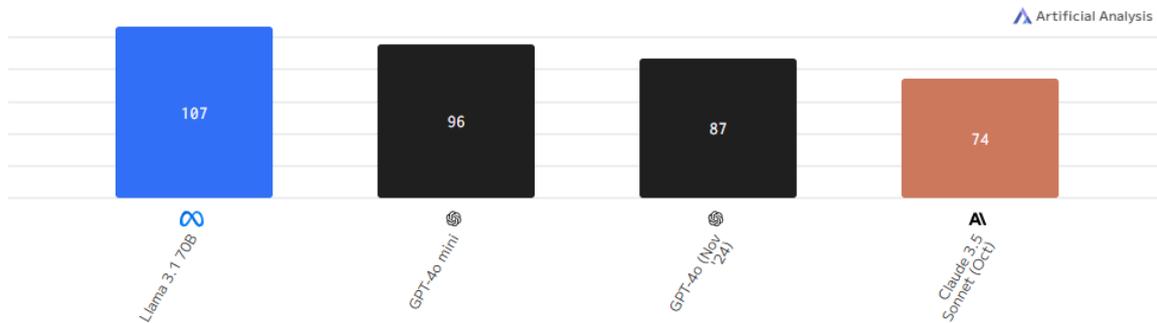


Abbildung 2.3 Ausgabegeschwindigkeit [14]

Hier können die Open-Source Modelle überzeugen, was aber hauptsächlich der geringeren Intelligenz geschuldet ist wie in Abbildung 4.3 zu sehen ist.

### 2.3.5. Preis

Basierend auf dem „Artificial Analyses Intelligence Index“ und den Preisen für verwendete Tokens kann anschließend ein „Intelligenz vs. Preis“ Vergleich (Abbildung 4.4) gezogen werden.

#### Intelligence vs. Price

Artificial Analysis Intelligence Index (Version 2, released Feb '25); Price: USD per 1M Tokens

Most attractive quadrant

■ GPT-4o (Nov '24) ■ GPT-4o mini ■ Llama 3.1 70B ■ Claude 3.5 Sonnet (Oct)

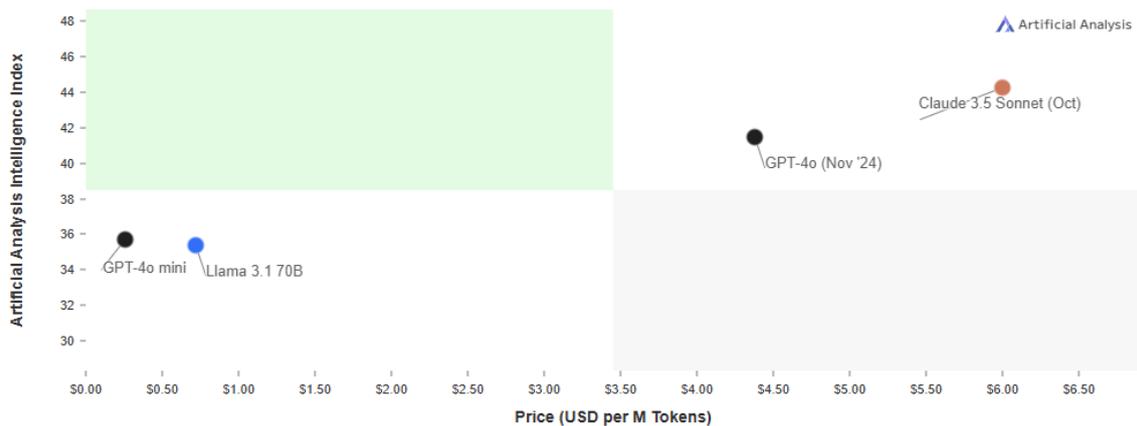


Abbildung 2.4 Intelligenz vs. Preis [14]

### 2.3.6. Vor und Nachteile

Es werden kurz die Vor- und Nachteile der 3 Modelle präsentiert.

#### **OpenAI – GPT-4o**

Ein wesentlicher Vorteil von OpenAI liegt darin, dass Unternehmen die Möglichkeit haben, eine eigene Instanz in Azure bereitzustellen. Dadurch behalten sie die volle Datenhoheit und können sicherstellen, dass alle Anwendungen DSGVO-konform betrieben werden, da sensible Informationen intern verwaltet werden. [18] Zusätzlich überzeugt die benutzerfreundliche Integration über das bereitgestellte SDK, das einen schnellen und unkomplizierten Zugriff auf eine breite Auswahl an Funktionen und Modellen ermöglicht, was die Entwicklung maßgeschneiderter Lösungen erleichtert. Als Nachteil muss jedoch erwähnt werden, dass das aktuell von OpenAI angebotene Modell in puncto Intelligenz und Leistungsfähigkeit nicht als das fortschrittlichste auf dem Markt angesehen wird.

#### **Claude 3.5 Sonnet**

Das SDK von Anthropic ist so gestaltet, dass es eine benutzerfreundliche und intuitive Integration in Anwendungen ermöglicht, wodurch Entwicklerinnen und Entwickler schneller mit der Nutzung beginnen können. Allerdings bietet es im Vergleich zu anderen Anbietern einen eingeschränkteren Funktionsumfang, was möglicherweise die Flexibilität in bestimmten Anwendungsszenarien reduziert. Trotz dieser Limitation gilt das Modell von Anthropic aktuell als eines der leistungsstärksten am Markt, was besonders bei rechenintensiven und komplexen Aufgaben von Vorteil ist. Ein wesentlicher Nachteil besteht darin, dass die Modelle von Anthropic nicht in der eigenen Cloud bereitgestellt werden können, was in manchen Fällen zu Herausforderungen hinsichtlich des Datenschutzes und der Datensouveränität führt. Zudem liegt der Preis für die Nutzung der Anthropic-Modelle etwas über dem von OpenAI, was insbesondere in kostenbewussten Projekten eine Rolle spielen kann.

#### **Llama 3.1 70B**

Das Llama Modell, von Facebook entwickelt, gehört zur Familie der Open-Source-Modelle, die insbesondere durch ihre Datenhoheit überzeugen. Der entscheidende Vorteil dieser Modelle liegt darin, dass sämtliche verarbeiteten Daten in der eigenen Hand bleiben – ein Aspekt, der besonders bei der Arbeit mit sensiblen Informationen von großer Bedeutung ist. Abgesehen von den erforderlichen Hardwarekosten ist die Nutzung dieser Modelle nahezu kostenfrei, was sie zu einer attraktiven Option für Unternehmen macht, die in KI-basierte Lösungen investieren möchten. Zudem bietet die Llama-Reihe verschiedene Modellgrößen an, sodass Anwender das für ihre spezifischen Hardwareanforderungen und Leistungsbedürfnisse optimale Modell auswählen können. Es ist jedoch zu beachten, dass die derzeit verfügbaren Llama-Modelle ausschließlich für textbasierte Anwendungen entwickelt wurden und aktuell keine Bildanalysen durchführen können.

### 2.3.7. Konklusion

Aus Gründen des Datenschutzes und des Funktionsumfangs ist die Wahl auf OpenAI gefallen. Aufgrund der soliden Leistung und der erweiterten Funktionalitäten ist es später leichter neue Features zu implementieren. Die Leistung der Modelle GPT-4o und Claude 3.5 Sonnet unterscheidet sich nur minimal und ist somit kein Ausschlusskriterium. Das Open Source Modell Llama 3.1 kann mit den führenden Anbietern leider nicht mithalten, kann aber für einfache kognitive Aufgaben, wie z.B. Texte zusammenfassen, benutzt werden, um die Kosten zu optimieren. Allerdings wird für eine lokale Ausführung eine leistungsstarke Grafikkarte und CPU benötigt.

## 2.4. Auswahl der Web-Frontends: Vergleich von Frameworks

Im Rahmen der Grundlagen und Methoden wurden zunächst verschiedene Frameworks sowie diverse EXIF-Reader verglichen. Dabei wurden deren Leistungsfähigkeit, Erweiterbarkeit und Kompatibilität mit den Anforderungen geprüft. Auf Basis der gewonnenen Erkenntnisse und unter Berücksichtigung der spezifischen Projektvorgaben wurde die am besten geeignete Lösung ausgewählt.

### 2.4.1. Framework-basierte Entwicklung vs. JavaScript

Bei der Entwicklung moderner Webanwendungen stellt sich die Frage, ob der Einsatz eines Frameworks notwendig ist. Grundsätzlich kann man mit nativem JavaScript arbeiten, das ohne zusätzliche Bibliotheken auskommt. Für kleinere und weniger komplexe Anwendungen kann Vanilla JavaScript eine geeignete Wahl sein, insbesondere wenn keine Skalierung oder Erweiterung der Anwendung in Aussicht steht. Allerdings erfordert die aktuelle Aufgabenstellung eine hohe Skalierbarkeit, weshalb Vanilla JavaScript in diesem Fall nicht in Frage kommt. Stattdessen wird ein Framework-basiertes Vorgehen bevorzugt, das die spätere Erweiterbarkeit und Wartbarkeit der Anwendung sicherstellt.

#### 2.4.1.1. Framework-basierte Entwicklung

Ein Framework bildet das Fundament für die Software-Entwicklung. Es erlaubt Entwicklerinnen und Entwicklern, nicht bei null anzufangen, sondern auf bewährte Programmstrukturen für Standardfunktionen zurückzugreifen. In der objektorientierten Programmierung werden dabei Klassen als Baupläne für Software-Objekte genutzt. Ein Framework stellt eine Vielzahl solcher Klassen bereit und definiert das grundlegende Design der Software. Wird es für Web-Anwendungen verwendet, spricht man von einem Webframework. [19] [20] [21]

#### 2.4.1.2. Vorteile von Frameworks

Moderne Frontend-Frameworks bieten zahlreiche Vorteile, die die Entwicklung effizienter und skalierbarer Anwendungen ermöglichen. Im Folgenden werden zentrale Stärken wie verbesserte Entwicklerproduktivität, optimierte Wartbarkeit, standardisierte Architekturansätze sowie Performance- und Erweiterbarkeitspotenziale näher betrachtet.

### **Strukturierte Entwicklung**

Frameworks wie React [22], Angular [23] oder Vue.js [24] bieten eine klar definierte, komponentenbasierte Architektur, die besonders bei großen Projekten eine bessere Code-Organisation und eine erhöhte Wartbarkeit ermöglicht. Sie unterstützen durch integrierte Tools wie State Management und Routing, was die effiziente Handhabung komplexer Anwendungen vereinfacht.

### **Performance-Optimierungen**

Frameworks implementieren Techniken wie den Virtual DOM (z. B. in React und Vue.js), der nur die Teile einer Webseite aktualisiert, die sich tatsächlich ändern. Dies reduziert den Rechenaufwand und verbessert die Performance erheblich im Vergleich zu einem Vanilla-JavaScript-Ansatz, bei dem der gesamte DOM neu gerendert werden müsste.

### **Modularität und Wiederverwendbarkeit**

Frameworks fördern die Wiederverwendung von Komponenten und ermöglichen dadurch eine modulare Entwicklung. Dies erleichtert nicht nur die Skalierbarkeit und beschleunigt die Entwicklungszeit bei großen Projekten, sondern sorgt auch dafür, dass der Quellcode lesbarer und einfacher wartbar bleibt.

### **Community und Support:**

Die Community und Unterstützung großer Frameworks zeigen sich nicht nur in der Qualität der Dokumentation, sondern auch in der Anzahl von GitHub-Stars und Forks, die in der Tabelle 2.1 Frameworks dargestellt sind. [21]

#### **2.4.1.3. Nachteile von Frameworks**

Trotz ihrer zahlreichen Vorteile bringen moderne Frontend-Frameworks auch spezifische Herausforderungen mit sich. Im Folgenden werden zentrale Schwächen wie erhöhte Komplexität, zusätzlicher Overhead sowie mögliche Abhängigkeiten von der Community und deren Auswirkungen auf Wartbarkeit und Performance dargestellt.

#### **Lernkurve**

Komplexere Frameworks erfordern das Erlernen zusätzlicher Konzepte und Techniken, was den Einstieg erschwert und mehr Zeit für die Einarbeitung erfordert. Die Syntax ist oft schwer zu erlernen, was eine zusätzliche Hürde für eine schnelle und effiziente Entwicklung darstellen kann.

#### **Abhängigkeiten und Overhead**

Umfangreiche Frameworks bringen oft zusätzliche Abhängigkeiten mit, was zu größeren Dateigrößen und somit zu langsameren Ladezeiten der Anwendung führt.

## Codeballast und Community

Frameworks bieten oft viele universelle Funktionen. Entwicklerinnen und Entwickler nutzen oft nur einen Bruchteil davon. Das führt zu überflüssigem Code und Ballast. Bleibt die Weiterentwicklung aus, sei es durch die Entwicklerinnen und Entwickler oder die Community, fehlen künftige Updates und Fehlerbehebungen. Langfristig entstehen Wartungsprobleme.

### 2.4.2. Vergleich verschiedener Frameworks

Viele Unternehmen wie Facebook (React) und Google (Angular) haben ihre Frameworks als Open Source veröffentlicht, um von der Entwicklergemeinschaft zu profitieren. Durch Open Source kann die Software schneller weiterentwickelt werden, da Entwicklerinnen und Entwickler weltweit Fehler finden, Sicherheitslücken schließen und neue Features beitragen. Gleichzeitig erhöhen Unternehmen so die Verbreitung ihrer Technologien, setzen Standards und binden Entwicklerinnen und Entwickler langfristig an ihre Plattformen und Ökosysteme. Dafür werden mehrere Quellen recherchiert und miteinander verglichen. [25] [20] [26]

#### 2.4.2.1. React [22]

React ist keine vollständige Framework-Lösung, sondern eine UI-Bibliothek, die sich auf die Entwicklung von Benutzeroberflächen konzentriert. Es nutzt den virtuellen DOM, um die Performance zu steigern, indem es nur Teile der Webseite neu rendert, die sich geändert haben. React ist sehr flexibel, da es Entwicklerinnen und Entwicklern erlaubt, andere Bibliotheken für Funktionen wie Routing und State-Management hinzuzufügen. Diese Modularität kann jedoch, als Nachteil betrachtet werden, da für komplexere Projekte zusätzliche Tools integriert werden müssen.

#### Sicherheitsaspekte

React bietet keinen integrierten Schutz vor Sicherheitsrisiken wie Cross-Site Scripting (XSS), nutzt jedoch automatisches Escaping von Zeichen in JSX, um XSS-Angriffe zu verhindern. Zusätzliche Sicherheitsbibliotheken zum Schutz vor unsicheren Inhalten müssen jedoch implementiert werden.

#### Benutzerfreundlichkeit

React ist sehr flexibel und ermöglicht die Erstellung benutzerfreundlicher Interfaces. Es bietet jedoch keine nativen UI-Komponenten für Barrierefreiheit, weshalb externe Bibliotheken genutzt werden müssen.

#### Leistungsoptimierung und Skalierbarkeit

Durch den Virtual DOM und Techniken wie Lazy Loading und Code-Splitting ist React sehr performant und lässt sich für große Anwendungen gut skalieren.

#### Wartbarkeit und Zukunftssicherheit

Die komponentenbasierte Struktur von React fördert die Wartbarkeit. Die starke Community und kontinuierliche Updates von Facebook machen es zu einer zukunftssicheren Wahl.

### **Stärken**

Eine hohe Flexibilität wird erreicht, da Komponenten wiederverwendbar sind und leicht angepasst werden können. Zusätzlich wird ein großes Ökosystem bereitgestellt. UI-Updates werden schnell ausgeführt, da nur veränderte Bereiche im virtuellen DOM verarbeitet werden. Projekte können dadurch effizient angepasst und erweitert werden.

### **Schwächen**

React selbst bietet nur grundlegende Funktionen. Deshalb werden oft zusätzliche Bibliotheken benötigt, um typische Anforderungen abzudecken. Dies erhöht die Komplexität und erschwert neuen Entwicklerinnen und Entwicklern den Einstieg. Komplexere Anwendungen bestehen meist aus vielen miteinander verbundenen Komponenten. Dadurch entstehen Abhängigkeiten, die eine sorgfältige und umfassende Architekturplanung erfordern.

### **Unternehmen**

Dropbox, Instagram, Airbnb und Facebook setzen React ein. Diese Plattformen profitieren von hoher Skalierbarkeit und Leistung. Eine universelle Einsetzbarkeit wird dadurch gefördert.

### **2.4.2.2. Angular [23]**

Angular ist ein vollständiges Framework, das sich auf die Entwicklung großer und komplexer Anwendungen konzentriert. Es bietet Zwei-Wege-Datenbindung und Dependency Injection, um den Code strukturierter zu gestalten. TypeScript wird eingesetzt, um robuste Anwendungen zu ermöglichen. Der umfangreiche Funktionsumfang kann jedoch die Lernkurve erhöhen.

### **Sicherheitsaspekte**

Angular schützt vor Cross-Site Scripting (XSS) Angriffen. Dabei wird verhindert, dass Angreifer schädlichen Code in Webseiten einschleusen. Die Template-Syntax filtert unsichere Inhalte automatisch. So gelangen keine gefährlichen Daten ins HTML. Zusätzliche Sicherheitsfunktionen sind standardmäßig enthalten.

### **Benutzerfreundlichkeit**

Viele vorgefertigte UI-Komponenten sind enthalten. Barrierefreie Interfaces können effizient erstellt werden. Dies erleichtert die Umsetzung benutzerfreundlicher Lösungen.

### **Leistungsoptimierung und Skalierbarkeit**

Ahead of Time Kompilierung übersetzt den Code schon vor der Ausführung in optimiertes JavaScript. Lazy Loading lädt einzelne Teile der Anwendung erst, wenn sie tatsächlich gebraucht werden. Dadurch verbessert sich die Performance und Angular eignet sich gut für große Projekte.

### **Wartbarkeit und Zukunftssicherheit**

Klare Strukturen fördern die Wartbarkeit. Die Unterstützung durch Google und eine aktive Community sichert langfristige Aktualisierungen. Projekte bleiben somit gut erweiterbar.

**Stärken**

Umfangreiche Funktionen erleichtern die Entwicklung komplexer Anwendungen. Starke Sicherheitsfeatures beugen Angriffen vor. Eine gute Eignung für große Projekte wird gewährleistet.

**Schwächen**

Eine steile Lernkurve erfordert intensivere Einarbeitung. Die komplexe Architektur verlangt mehr Planungsaufwand. Kleinere Projekte können dadurch überfrachtet wirken.

**Unternehmen**

Die Deutsche Bank nutzt Angular für interne Anwendungen. Gmail wird ebenfalls mit Angular realisiert. Beide profitieren von hoher Stabilität und guter Wartbarkeit.

**2.4.2.3. Vue.js [24]**

Vue.js ist eine progressive Bibliothek. Das bedeutet, man kann Vue stufenweise in ein bestehendes Projekt einbinden, ohne alles neu schreiben zu müssen. Die einfache Syntax senkt die Lernkurve. Vor allem kleinere und mittlere Anwendungen profitieren von dieser Flexibilität. Für größere Projekte fehlen jedoch einige Funktionen, die Angular oder React mitbringen. Daher sind hier oft zusätzliche Bibliotheken nötig.

**Sicherheitsaspekte**

Vue.js schützt grundlegend vor XSS. In den Templates werden unsichere Zeichen automatisch escaped, damit kein schädlicher Code ins HTML gelangt. Dedizierte Sicherheitsfunktionen muss man selbst implementieren, um die Anwendung umfassend abzusichern.

**Benutzerfreundlichkeit**

Die API von Vue ist leicht verständlich und ermöglicht eine schnelle Entwicklung von Benutzeroberflächen. Es gibt keine integrierten Werkzeuge für Barrierefreiheit. Wer solche Funktionen benötigt, muss dafür zusätzliche Lösungen einsetzen.

**Leistungsoptimierung und Skalierbarkeit**

Vue verwendet einen virtuellen DOM, der nur veränderte Teile der Seite neu rendert. Das sorgt für schnelle und effiziente Darstellungen. Dank der modularen Struktur lässt sich die Anwendung bei Bedarf mit Bibliotheken wie Vuex erweitern, ohne dass die Performance leidet.

**Wartbarkeit und Zukunftssicherheit**

Die modulare Architektur erleichtert die Pflege des Codes. Die Community um Vue wächst fortlaufend. Deshalb kann man darauf vertrauen, dass es auch in Zukunft regelmäßige Updates und Verbesserungen geben wird.

**Stärken**

Vue ist schnell zu erlernen und bietet viel Freiheit im Aufbau. Die effizienten Rendering-Techniken sorgen für gute Performance. Dadurch lassen sich Projekte zügig umsetzen.

## Schwächen

Die Community ist kleiner als bei Angular oder React. Einige grundlegende Funktionen sind nicht im Standardumfang enthalten, was den Einsatz in sehr großen Projekten erschweren kann. Aus diesem Grund greifen manche Entwickler in solchen Fällen lieber zu anderen Frameworks.

## Unternehmen

Nintendo, Adobe und GitLab setzen Vue.js ein. Sie profitieren von der einfachen Integration und der stabilen Performance. Diese Eigenschaften unterstützen eine erfolgreiche Projektumsetzung.

### 2.4.2.4. Svelte [27]

Svelte ist ein compilerbasiertes Framework, das den Code vor der Ausführung in optimiertes JavaScript umwandelt. Anders als React oder Vue gibt es keinen virtuellen DOM. Dadurch fällt weniger Bibliothekscode an, und die Anwendung bleibt schlank. Beim Kompilierungsansatz übersetzt Svelte Komponenten direkt in effiziente JavaScript-Funktionen. Das spart Laufzeit und sorgt für kurze Ladezeiten.

## Sicherheitsaspekte

Svelte schützt grundlegend vor XSS, weil Expressions in Templates automatisch escaped werden. So kann kein schädlicher Code ins HTML gelangen. Einen nativen Schutz durch Content Security Policy (CSP) gibt es nicht. CSP legt fest, welche Quellen für Skripte, Styles und Bilder erlaubt sind. Wer CSP einsetzen will, muss die entsprechenden Header oder Meta-Tags selbst einrichten, damit zusätzliche Schutzmaßnahmen greifen.

## Benutzerfreundlichkeit

Die Syntax von Svelte ist einfach und übersichtlich. Entwicklerinnen und Entwickler lernen schnell, wie man Komponenten definiert und Daten bindet. Da viele Funktionen ohne externe Bibliotheken funktionieren, bleibt die Projektstruktur aufgeräumt. Werkzeuge für Barrierefreiheit sind nicht integriert. Benötigt man solche Funktionen, kommen zusätzliche Pakete zum Einsatz.

## Leistungsoptimierung und Skalierbarkeit

Dank des Kompilierungsansatzes entstehen nur minimale JavaScript-Dateien. Die Anwendung startet daher sehr schnell und beansprucht weniger Ressourcen. Für größere Projekte lassen sich Module in getrennten Dateien organisieren und bei Bedarf über dynamische Importe nachladen. So bleibt die Performance auch bei wachsenden Anforderungen hoch.

## Wartbarkeit und Zukunftssicherheit

Der übersichtliche, generierte Code erleichtert die Pflege und das Finden von Fehlern. Änderungen am Quelltext sind direkt im kompilierten Ergebnis sichtbar. Die Community hinter Svelte wächst konstant, und das Team veröffentlicht regelmäßig Updates. Dadurch ist die langfristige Pflege des Frameworks gesichert.

### **Stärken**

Svelte erzeugt kompakte Bundles, was schnelle Ladezeiten ermöglicht. Ohne virtuellen DOM sinkt die Laufzeitlast. Die klare Syntax ist nah an HTML, CSS und JavaScript, sodass der Einstieg leichtfällt. CSP-Header lassen sich dank des schlanken Bundles gezielt setzen, um die Sicherheit zu erhöhen.

### **Schwächen**

Svelte ist als junges Projekt noch nicht so etabliert wie React oder Vue. Die Community ist kleiner, und manche Bibliotheken fehlen noch. In sehr komplexen Anwendungen entscheiden Entwickler selbst, welche externen Pakete sie hinzufügen, weil Svelte nur Basisfunktionen bietet.

### **Unternehmen**

Capital One und GoGuardian nutzen Svelte. Beide profitieren von kurzen Ladezeiten. Die Effizienz wird durch den Kompilieransatz gesteigert.

#### **2.4.2.5. Ember [28]**

Ember.js setzt auf feste Konventionen, die den Entwicklungsablauf vorgeben. Diese Orientierung nennt man „opinionated“, weil das Framework klare Vorgaben macht, statt jede Entscheidung dem Entwickler zu überlassen. Durch diese Regeln arbeiten Teams schnell und einheitlich. Umfangreiche Dokumentation und eingebaute Funktionen wie Ember Data erleichtern zudem den Umgang mit komplexen Datenmodellen. Allerdings bringt diese Struktur mehr Komplexität mit sich als bei leichteren Frameworks.

### **Sicherheitsaspekte**

Ember schützt durch Template-Escaping vor XSS. So wird verhindert, dass schädlicher Code ins HTML gelangt. Einen nativen CSRF-Schutz gibt es nicht automatisch. Wer CSRF-Schutz benötigt, muss diesen selbst im Backend einrichten oder eigene Maßnahmen ergänzen. So bleibt die Anwendung sicher, wenn man die nötigen Anpassungen vornimmt.

### **Benutzerfreundlichkeit**

Ember liefert viele Werkzeuge und Vorgaben, die die Entwicklung beschleunigen. Apps lassen sich schnell barrierefrei gestalten. Die festen Regeln sorgen dafür, dass Entwickler nicht lange nach Best Practices suchen müssen, sondern von Anfang an einheitlich arbeiten.

### **Leistungsoptimierung und Skalierbarkeit**

Ember lädt dank Lazy Loading zunächst nur die wirklich benötigten Module. In Kombination mit der modularen Bauweise kann man auch sehr umfangreiche Projekte performant halten. Zwar führt die umfangreiche Grundstruktur manchmal zu etwas höheren Ladezeiten, doch durch gezieltes Nachladen von Code bleibt die Performance auf hohem Niveau.

### **Wartbarkeit und Zukunftssicherheit**

Die klare Konventionsstruktur und die ausführliche Dokumentation erleichtern die Pflege des Codes. Fehler lassen sich schneller finden, weil festgelegt ist, wo Dateien liegen und wie sie zusammenarbeiten. Die langjährige Community hinter Ember sorgt zudem für regelmäßige Updates und eine stabile Weiterentwicklung.

### **Stärken**

Starke Konventionen verringern den Konfigurationsaufwand und helfen, Projekte zügig zu starten. Viele eingebaute Features wie Data-Layer und CLI-Tools unterstützen besonders umfangreiche Anwendungen. Da Ember opinionated ist, legt es bestimmte Strukturen fest, so dass weniger Diskussionen über Architektur nötig sind und der Code einheitlicher bleibt.

### **Schwächen**

Die Einstiegshürde ist höher als bei leichteren Frameworks, weil man zuerst die strikten Regeln lernen muss. Bei sehr individuellen Anforderungen bietet Ember weniger Spielraum, da die Konventionen stark vorgeben, wie etwas aufgebaut sein soll.

### **Unternehmen**

LinkedIn, Intercom und Netflix setzen Ember.js ein. Sie schätzen die klare Struktur, die hohe Stabilität und die integrierten Werkzeuge. Damit können sie große Projekte effizient umsetzen und langfristig warten.

### **2.4.2.6. Konklusion**

Die oben genannten Frameworks sind in der **Fehler! Verweisquelle konnte nicht gefunden werden.** übersichtlich dargestellt. Ergänzt werden sollen dabei persönliche Erfahrungen sowie GitHub-Stars und Forks. GitHub-Stars zeigen das allgemeine Interesse an einem Projekt. Forks spiegeln die Anzahl der Repository-Kopien wider, die zur Weiterentwicklung oder individuellen Anpassung genutzt werden. Aufgrund persönlicher Erfahrungen, stabiler Weiterentwicklungsperspektiven und der hohen Eignung für performante Benutzeroberflächen fällt die Wahl auf React als bevorzugtes Framework. React steht unter der MIT-Lizenz, was es für zahlreiche Projekte attraktiv macht. Zur Verbesserung der Typsicherheit wird zusätzlich TypeScript eingesetzt, das unter der Apache-2.0-Lizenz steht.

Aspekt	React/TS	Angular	Vue	Svelte	Ember
Performance	+	~	+	+	~
Skalierbarkeit	+	+	~	~	+
Sicherheit	~	+	~	~	+
Nutzerfreundlichkeit	+	+	+	+	~
Wartbarkeit	+	+	+	~	+
Zukunftssicherheit	+	+	+	~	+
Erfahrung	+	-	-	-	-
GitHub Stars	~214k+	~89k+	~214k+	~72k+	~23k+
GitHub Forks	~44k+	~24k+	~37k+	~3.6k+	~5k+
Lizenz	MIT/Apache2.0	MIT	MIT	MIT	MIT

Tabelle 2.1 Frameworks

#### 2.4.2.7. Exchangeable Image File Format (EXIF) Daten

Digitalkameras speichern neben den Bilddaten auch sogenannte Exif-Daten ab. Diese Metadaten enthalten zahlreiche Informationen zur jeweiligen Aufnahme, wie z. B. das Kameramodell, das Aufnahmedatum, die Uhrzeit, die Blende und die Belichtungszeit. EXIF steht für "Exchangeable Image File Format" und stellt einen Standard dar, der es ermöglicht, diese zusätzlichen Aufnahmeinformationen in Bilddateien zu hinterlegen. Aktuell ist die Version 2.3 des EXIF-Standards, die im April 2010 von der JEITA (Japan Electronics [29] and Information Technology Industries Association) und der CIPA (Camera & Imaging Products Association) gemeinsam veröffentlicht wurde. [30] [31]

#### 2.4.2.8. Nutzbarkeit von EXIF-Daten

Wird ein Foto mit einer Digitalkamera aufgenommen, werden sowohl die Pixelinformationen als auch die Exif-Daten erzeugt und in einem voreingestellten Bildformat gespeichert. Einige dieser Metadaten können Aufschluss darüber geben, ob sie zu einem bestimmten Ereignis passen. Mithilfe vorgefertigter Bibliotheken lassen sich diese Daten auslesen und durch im Quellcode implementierte Vergleichslogiken automatisch analysieren und visualisieren. [32]

#### 2.4.2.9. Fiktives Fallbeispiel

Person A sendet Person B ein Foto von einem Strandurlaub. Auf dem Bild ist Person A an einem idyllischen Sandstrand in Italien zu sehen. Person B interpretiert das Bild zunächst so, dass es weder bearbeitet wurde noch gängige Smartphone-Filter verwendet wurden. Dennoch bestehen Zweifel, weshalb Person B die EXIF-Daten des Bildes mit geeigneter Software ausliest. Dabei stellt sie fest, dass das Aufnahmedatum des Bildes bereits ein Jahr zurückliegt. Auch andere Zeitstempel, wie die Digitalisierungszeit und der GPS-Zeitstempel, bestätigen dies. Es wäre lebensfremd anzunehmen, dass Person A die Zeitstempel manipuliert hat, um das Bild zurückzudatieren und es dennoch als aktuelles Urlaubsbild zu versenden. Damit ist bewiesen,

dass das Bild ein Jahr alt ist. Nicht bewiesen ist jedoch, ob Person A aktuell tatsächlich im Urlaub ist oder nicht.

#### **2.4.2.10. Funktion eines ExifReaders**

Das Programm lädt die Bilddatei und öffnet sie im binären Modus. EXIF-Daten befinden sich in spezifischen Bereichen der Datei und müssen in binärer Form gelesen werden.

##### **Suchen der EXIF-Daten**

EXIF-Daten sind in JPEG- und TIFF-Dateien in einem bestimmten Abschnitt eingebettet. Der EXIF-Reader sucht nach dem sogenannten APP1-Segment in JPEG-Dateien oder im (Image File Directory) IFD in TIFF-Dateien, in denen die Metadaten gespeichert sind. Dieses Segment wird durch einen speziellen Header (z.B. Exif\0\0) gekennzeichnet, und der Reader muss diesen Header erkennen, um die EXIF-Daten zu finden. [33]

##### **Auslesen der EXIF-Tags**

Sobald der EXIF-Header erkannt wurde, beginnt der Reader mit dem Auslesen der EXIF-Tags. Diese sind als Schlüssel-Wert-Paare (Key/Value-Pairs) strukturiert, wobei jeder Tag spezifische Metadaten wie Belichtungszeit, ISO-Wert, Datum oder GPS-Koordinaten enthält. Die Struktur der Tags ist standardisiert und umfasst die Tag-Nummer, den Datentyp, den Wert sowie die Länge. Der EXIF-Reader analysiert diese Struktur und interpretiert die enthaltenen Informationen entsprechend. [34]

##### **Interpretation der Daten**

Die rohen Daten werden dann in verständliche Formate umgewandelt. Einige Daten sind einfache numerische Werte (wie ISO), andere sind komplexer (wie Datum und Uhrzeit) und müssen entsprechend formatiert werden.

#### **2.4.2.11. Auswahl eines EXIF-Readers**

Für jede Programmiersprache steht eine Vielzahl von Bibliotheken zur Verfügung, die Metadaten aus Bildern auslesen und bearbeiten können. Abhängig davon, ob es sich um ein proprietäres Projekt handelt, spielt die Lizenz, unter der der Quellcode genutzt werden darf, eine entscheidende Rolle. [35] [36]

#### **2.4.2.12. Konklusion<sup>2</sup>**

Pillow ist eine Python-Bibliothek zur Bildbearbeitung, die sich leicht installieren und nutzen lässt. Es gibt auch andere Bibliotheken wie Pyexif oder libtiff, doch Pillow bietet von Haus aus viele Funktionen zum Lesen und Schreiben verschiedener Bildformate. Die Lizenz von Pillow erlaubt den Einsatz in proprietären Projekten ohne Einschränkungen. Da in unserem Backend Python verwendet wird, lässt sich Pillow problemlos in den bestehenden Technologie-Stack

---

<sup>2</sup> Quellmaterial: ChatGPT (GPT-4), Datum der Abfrage: 10.10.2024, Prompt: „Erkläre, warum Lizenzen bei der Auswahl eines EXIF-Readers wichtig sind. Pro und Contra für Pillow.“

integrieren. Außerdem erfordert Pillow wenig zusätzlichen Konfigurationsaufwand, sodass die Extraktion und Verarbeitung von EXIF-Daten schnell umgesetzt werden kann.

## 2.5. Datenschutzrechtliche Überlegungen (DSGVO) in der Versicherungsbranche

Die Verarbeitung von Daten in diesem Projekt unterliegt den strengen Anforderungen der EU-Datenschutz-Grundverordnung (DSGVO), zumal es sich um Versicherungsdaten handelt. Versicherungsunternehmen verarbeiten typischerweise **zahlreiche sensible personenbezogene Daten** ihrer Kundinnen und Kunden – von grundlegenden Identifikationsdaten (Name, Adresse, Geburtsdatum) bis hin zu gesundheitsbezogenen oder finanzrelevanten Informationen je nach Versicherungsart. Solche Daten dürfen gemäß DSGVO nur verarbeitet werden, wenn eine Rechtsgrundlage vorliegt, insbesondere wenn sie zur Vertragserfüllung notwendig sind (Art. 6 Abs.1b DSGVO) oder eine Einwilligung der betroffenen Person vorliegt. **Besondere Kategorien personenbezogener Daten** (wie Gesundheitsdaten bei einer Unfall- oder Lebensversicherung) erfordern in der Regel eine **ausdrückliche Einwilligung** oder müssen unter einen der spezialgesetzlichen Ausnahmetatbestände fallen. [37]

Für Versicherer bedeutet dies, dass alle im Projekt verwendeten Daten unter **Privacy-by-Design** Gesichtspunkten behandelt werden müssen. Im konkreten Fall wurde darauf geachtet, Produktivdaten zu anonymisieren, bevor sie in der Entwicklungs- und Testumgebung (Azure Cloud) genutzt werden. Dies ist heutzutage Branchenstandard, da echte Kundendaten ohne ausreichende Maskierung in nicht-produktiven Umgebungen massive Risiken bergen. Die DSGVO sowie weitere Gesetze (wie nationale Datenschutzgesetze oder in den USA HIPAA) **untersagen die uneingeschränkte Verwendung personenbezogener Daten zu Testzwecken ohne Zustimmung**. [38] Verstöße können zu schwerwiegenden Folgen führen – neben dem Risiko von Datenpannen drohen **hohe rechtliche Strafen** (bis zu 4 % des weltweiten Jahresumsatzes) sowie Imageschäden und Vertrauensverlust bei Kundinnen und Kunden. [39] Entsprechend wurde beim Migrieren der Oracle-Datenbank sämtliche persönlich identifizierbare Information (PII) durch realistische Dummy-Werte ersetzt, um ein **datenschutzkonformes Testen** zu ermöglichen.

Darüber hinaus sind bei Cloud-Lösungen in der Versicherungsbranche vertragliche und technische Maßnahmen zu berücksichtigen, um DSGVO-Compliance sicherzustellen. Mit dem Hosting der Datenbank in Microsoft Azure gelten die hohen Sicherheitsstandards von Azure; jedoch musste z.B. geklärt werden, in welcher Region die Daten liegen (bevorzugt in der EU, um internationale Datenflüsse zu minimieren). Alle **Kommunikation zwischen Frontend, API und Datenbank** erfolgt ausschließlich verschlüsselt über HTTPS. Auch die Ablage von Dokumenten in Azure Blob Storage erfolgt **verschlüsselt**; Azure speichert standardmäßig alle Blobs verschlüsselt und erlaubt zusätzlich clientseitige Verschlüsselung bei besonders sensiblen Inhalten. Versicherungsunternehmen sind verpflichtet, personenbezogene Daten durch angemessene technische und organisatorische Maßnahmen zu schützen – dazu zählen **Verschlüsselung der Datenspeicher und gesicherte Übertragungswege (TLS/HTTPS)**. Im Falle einer entdeckten Datenschutzverletzung (z.B. ein Datenleck) muss zudem binnen 72 Stunden die Aufsichtsbehörde und die Betroffenen informiert werden. [39] Diese Anforderungen wurden im Projekt durch entsprechende Cloud-Einstellungen und regelmäßige Backups adressiert.

Speziell im Kontext der **Schadensfallabwicklung mit KI** ist auch Art. 22 DSGVO relevant, der automatisierte Entscheidungen mit erheblicher Auswirkung auf Personen regelt. Eine vollständige automatisierte Ablehnung oder Genehmigung eines Versicherungsanspruchs ohne menschliche Überprüfung könnte unter diese Bestimmung fallen. Daher wurde im Projekt darauf geachtet, dass die KI-Komponenten entscheidungsunterstützend wirken, letztlich aber ein Mensch im Loop verbleibt. Beispielsweise kann die KI einen Deckungsvorschlag abgeben oder Fälle mit hoher Betrugswahrscheinlichkeit kennzeichnen, doch die endgültige Entscheidung würde eine Sachbearbeiterin oder ein Sachbearbeiter treffen. Damit wird nicht nur DSGVO-Konformität gewahrt, sondern auch praktischen Erwägungen Rechnung getragen, da **Transparenz und Nachvollziehbarkeit** bei KI-Entscheidungen in der Versicherung essenziell sind. Die verwendeten Modelle (z.B. GPT-4o) liefern Begründungen bzw. können so konfiguriert werden, dass Entscheidungsgrundlagen (welche Regel griff, welche Abweichung erkannt wurde etc.) mit ausgegeben werden – dies erleichtert eine etwaige **Erklärbarkeit** der KI-Resultate gegenüber Kundinnen und Kunden und Prüfern.

Es wurden im Projekt die **DSGVO-Compliance** durch folgende Maßnahmen sichergestellt:

**Datenminimierung und Anonymisierung:** Einsatz anonymisierter Echtdaten in der Cloud-Datenbank zu Test- und Entwicklungszwecken. [38] Keine unnötige Verarbeitung von Klardaten außerhalb produktiver Systeme.

**Sichere Infrastruktur:** Azure-Dienste mit Verschlüsselung und rollenbasierten Zugriffsrechten; Kommunikation nur über gesicherte Kanäle (HTTPS).

**Vertragliche Absicherung:** Nutzung von Azure (EU) und ggfs. Abschluss eines Auftragsverarbeitungsvertrags. Bei externen KI-Services (z.B. OpenAI) Nutzung über Azure OpenAI Service, um vertragliche Rahmenbedingungen von Microsoft zu nutzen.

**Menschliche Kontrolle:** KI-gestützte Entscheidungen werden durch Menschen validiert, um sowohl fachliche Korrektheit als auch rechtliche Zulässigkeit sicherzustellen.

Diese Überlegungen bilden den Rahmen, innerhalb dessen die Implementierung erfolgte.

### 3. Implementierung einer vektorisierten Datenbank für existierende Schadensfälle sowie einer API-Schnittstelle in einer cloudbasierten Infrastruktur (Alexander Di Bernardo)

In diesem Kapitel wird die technische Umsetzung einer automatisierten Deckungsprüfung im Projekt „KLV goes AI“ vorgestellt. Im Mittelpunkt steht dabei der Aufbau einer vektorisierten Datenbank sowie einer API-Schnittstelle in einer cloudbasierten Infrastruktur.

#### 3.1. Aufgabenstellung

Die **Deckungsprüfung** in Versicherungsunternehmen beschäftigt sich mit der Frage, ob ein gemeldeter Schadenfall im Rahmen der bestehenden Versicherungspolize abgesichert ist. Derzeit erfolgt diese Prüfung in vielen Unternehmen noch per Hand. Dabei analysieren Sachbearbeiterinnen oder Sachbearbeiter sämtliche von Kundinnen und Kunden eingereichten Unterlagen und Informationen wie Schadensberichte, Fotos oder andere Dokumente, um das versicherte Risiko und die Ursache des Schadens eindeutig festzustellen. Anschließend vergleichen sie die ermittelten Sachverhalte mit den Bedingungen der jeweiligen Polize, um eine Entscheidung über die Deckung treffen zu können. Dieses Vorgehen ist nicht nur **zeitintensiv und fehleranfällig**, sondern verlangt häufig auch das Heranziehen verschiedener zusätzlicher Informationsquellen – beispielsweise Vertragsklauseln, frühere Schadensmeldungen oder externe Datenquellen, wie etwa Wetteraufzeichnungen bei Schäden durch Naturereignisse. [40] Der manuelle Prüfprozess kann somit die Bearbeitung verzögern und belastet zusätzlich erhebliche personelle Ressourcen.

Die **Zielsetzung** des Projekts „KLV goes AI“ besteht darin, den Ablauf der Deckungsprüfung durch innovative KI-Verfahren weitgehend zu automatisieren, um die Prozesse schneller, wirtschaftlicher und zuverlässiger durchzuführen. Die **automatisierte** Deckungsprüfung soll eigenständig relevante Daten aus vorliegenden Schadensmeldungen, inklusive Freitexten und Anhängen, erkennen und mit den aktuellen Versicherungsbedingungen vergleichen. Insbesondere Technologien aus den Bereichen Natural Language Processing (NLP) und maschinelles Lernen bieten hierfür großes Potenzial, da sie in der Lage sind, einen wesentlichen Teil der bisher manuell durchgeführten Prüfung zu übernehmen. Beispielsweise analysiert eine NLP-Anwendung die eingegebenen Schadensbeschreibungen und überprüft eigenständig, ob die beschriebene **Schadenart und -ursache** mit dem versicherten Risiko übereinstimmen. Damit kann unmittelbar festgestellt werden, ob ein Schaden gedeckt ist und welche spezifische Vertrag greift, sodass der Kunde bestenfalls bereits während der Meldung eine vorläufige Deckungsentscheidung erhalten könnte. [41] Eine wesentliche **Herausforderung** hierbei ist die korrekte Interpretation der oft komplexen und juristisch formulierten Versicherungsbedingungen sowie deren Zuordnung zu den vielfältigen Schadensschilderungen der Kundinnen und Kunden. Da rein regelbasierte Ansätze hier an Grenzen stoßen, ist der Einsatz **intelligenter, lernfähiger Systeme** erforderlich, welche den Sinngehalt der Texte semantisch erfassen können.

Die **Notwendigkeit**, diesen Prüfprozess zu automatisieren, ergibt sich sowohl aus internen **betrieblichen** Erfordernissen als auch aus den **steigenden Erwartungen** der Kundinnen und Kunden. Rasche und verlässliche Entscheidungen zur Deckung sind wesentlich, um Schäden schnell regulieren zu können – etwas, das Kundinnen und Kunden heute zunehmend voraussetzen. Laut Branchenanalysen wünschen sich 21 % der Versicherungskunden, dass Schäden innerhalb weniger Stunden bearbeitet werden, während in der Praxis 43 % der Betroffenen über zwei Wochen auf eine Entscheidung warten müssen. [42] Diese Abweichung verdeutlicht den Handlungsdruck auf Versicherungen, ihre Abläufe effizienter zu gestalten, um im Wettbewerb zu bestehen. Durch den Einsatz automatisierter Prozesse kann die **Bearbeitungsdauer** deutlich reduziert werden, was sowohl die Zufriedenheit der Kundinnen und Kunden erhöht als auch die Mitarbeiterinnen und Mitarbeiter entlastet. Routinefälle lassen sich dabei automatisiert abwickeln, während Experten lediglich anspruchsvolle Fälle übernehmen müssen. KI-Systeme fungieren in diesem Kontext als eine **Art Filter**, der einfache Fälle unmittelbar bearbeitet und nur solche Vorgänge an Sachbearbeiterinnen oder Sachbearbeiter weitergibt, die eine genauere Prüfung benötigen. [41] Zusätzlich sorgt eine einheitliche KI-basierte Prüfung für **gleichbleibende Qualität** bei Entscheidungen und minimiert das Risiko menschlicher Fehler sowie unterschiedliche Interpretationen der Versicherungsbedingungen.

Zusammenfassend liegt die Aufgabe darin, eine automatisierte Deckungsprüfung zu entwickeln, welche Schadensmeldungen **eigenständig analysiert**, mit den hinterlegten Versicherungsregelwerken abgleicht und eine **klare Entscheidungsempfehlung** ausgibt (Deckung gegeben oder nicht). Diese Prüfung soll **deutlich schneller** erfolgen als bisher – idealerweise nahezu in Echtzeit oder innerhalb weniger Sekunden – und dabei **verlässlich** genug sein, um direkt in den operativen Betrieb übernommen werden zu können. Bereits vorhandene Lösungen am Markt, beispielsweise KI-Plattformen wie omni:us, demonstrieren, dass eine solche automatisierte Deckungsprüfung **technisch realisierbar** ist und schon heute erfolgreich im Hintergrund eingesetzt wird. [43] Unsere Diplomarbeit konzentriert sich darauf, diese Technologie im Rahmen des Projekts „KLV goes AI“ praxisnah umzusetzen und optimal in die bestehende Systemlandschaft der Kärntner Landesversicherung (KLV) einzubinden.

### 3.2. Grundlagen / Methoden

Um **die automatisierte Deckungsprüfung im** Projekt "KLV goes AI" umsetzen zu können, wurden zunächst wesentliche **technische Voraussetzungen** geschaffen. Die Auswahl der KI-Technologien und deren grundlegende Integration erfolgte durch Herrn Griesser. Im Rahmen meines Teiles der Diplomarbeit war ich für die Implementierung der **Vektordatenbank Pinecone**, die **Migration umfangreicher Datenbestände** sowie den Aufbau einer zentralen, über ein **eigenentwickeltes Frontend wartbaren Regeldatenbank** verantwortlich. Im Folgenden werden diese zentralen technischen Komponenten detailliert vorgestellt.

#### 3.2.1. Pinecone – Vektordatenbank

Pinecone ist ein verwalteter Cloud-Dienst, der speziell für die Speicherung und Abfrage von **hochdimensionalen Vektordaten** (Embeddings) entwickelt wurde. Im Rahmen der automatisierten Deckungsprüfung werden relevante Textstellen wie Versicherungsbedingungen, Klauseln und Gesetze mithilfe von Embeddings in Pinecone gespeichert. Diese Vektorrepräsentationen ermöglichen schnelle semantische **Ähnlichkeitssuchen**, wodurch relevante Passagen

gefunden werden können, selbst wenn sie sprachlich unterschiedlich formuliert sind. [44] Bei Eingang einer Schadensmeldung wird die Schadensbeschreibung in ein Embedding überführt und Pinecone durchsucht nach Texten, die inhaltlich am besten übereinstimmen. Pinecone übernimmt dabei sämtliche Aufgaben rund um Skalierung, Indexierung und Verwaltung der Daten, sodass ich mich im Projekt auf die Integration und Nutzung konzentrieren konnte.

### 3.2.2. Migration der Datenbankbestände

Ein zentraler Bestandteil war die vollständige Migration von **rund zwei Milliarden anonymisierten Datensätzen** aus einer bestehenden Oracle-Datenbank in eine Azure-basierte MySQL-Datenbank. Diese Migration erforderte eine vollumfängliche Datenintegration sowie eine umfassende Anonymisierung, um einen DSGVO-konformen und leistungsfähigen Betrieb sicherzustellen. Das Vorgehen zur Datenmigration umfasste die Extraktion der Oracle-Daten, deren Umwandlung und Anpassung an das Zielsystem sowie die **automatisierte Einspielung in die Azure MySQL-Datenbank**. Dieser Prozess wurde **vollständig dokumentiert und automatisiert** durchgeführt, wodurch die Konsistenz und Vollständigkeit der Daten gewährleistet wurde.

### 3.2.3. Aufbau der zentralen Regeldatenbank

Um sämtliche gültigen Regelwerke (z.B. Gesetze, Bedingungen und Klauseln) effizient verwalten zu können, haben wir eine eigene zentrale Regeldatenbank aufgebaut. Diese Datenbank ermöglicht eine flexible Speicherung und Verwaltung aller relevanten Regelwerke und bietet über ein **eigens entwickeltes Web-Frontend** eine komfortable Möglichkeit für Fachanwender, **Regelwerke zu aktualisieren, hinzuzufügen oder anzupassen**. Durch diese Lösung wird gewährleistet, dass die Versicherungsbedingungen stets aktuell sind und Änderungen unmittelbar in der operativen Umgebung berücksichtigt werden können.

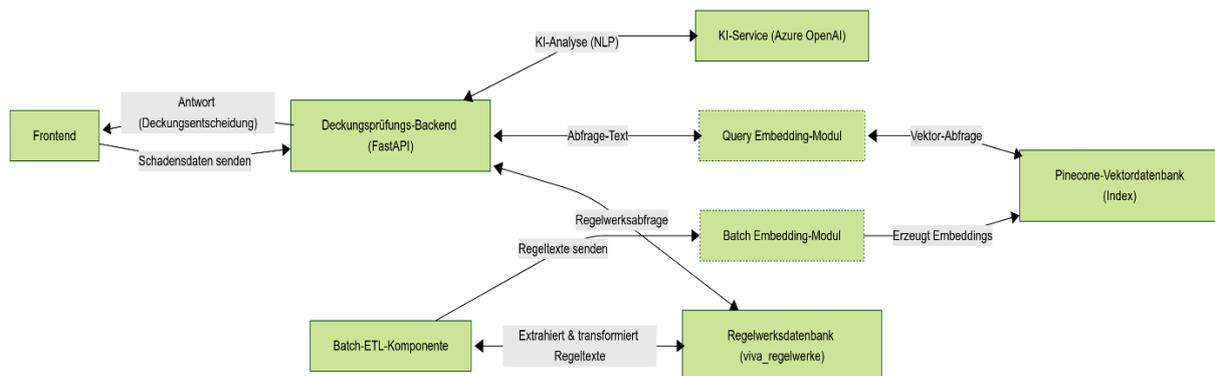


Abbildung 3.1 Datenfluss der Deckungsprüfung

### 3.2.4. Datenfluss in der Deckungsprüfung

Abbildung 3.1 stellt den schematischen Ablauf und die Integration der einzelnen Komponenten im Rahmen der **Deckungsprüfung** dar. Zu Beginn erhält das System alle notwendigen Informationen zu einem gemeldeten Schadenfall. Diese Daten werden entweder über ein digitales Schadenportal oder direkt aus einem bestehenden Bestandssystem übergeben. Die Schadendaten umfassen dabei typischerweise **sowohl strukturierte Daten** (wie Schadennummer, Vertragsnummer, Sparte) **als auch unstrukturierte Inhalte** wie Schadensschilderungen in Freitextform sowie Anhänge wie Bilder oder Dokumente.

Parallel dazu existiert eine interne **Regeldatenbank** (*viva\_regelwerke*), in der sämtliche relevanten Versicherungsbedingungen, Klauseln und Geschäftsregeln abgelegt sind. Diese Regeltexte wurden zuvor als sogenannte Embeddings (Vektorrepräsentationen) in der Pinecone-Datenbank gespeichert, um eine effiziente semantische Suche zu ermöglichen.

Wenn eine Anfrage zur Deckungsprüfung über das Frontend erfolgt, läuft der Prozess wie folgt ab:

Zunächst werden **alle benötigten Daten** zusammengeführt. Sollten lediglich Basisinformationen wie eine Schadennummer übermittelt werden, **greift das System automatisch auf Schnittstellen zurück**, um detaillierte Informationen (z.B. Kundendaten, Vertragsinformationen oder Schadenbeschreibungen) nachzuladen. Eingereichte Dokumente und Bilder werden einer Vorverarbeitung unterzogen, Details dazu sind in Abschnitt 3.3 beschrieben. Bei spezifischen Fällen, beispielsweise Blitzschäden, werden **zusätzlich externe Daten** wie Wetterinformationen genutzt, um die Plausibilität der Angaben zu prüfen.

Im nächsten Schritt erfolgt eine **semantische Analyse** der konsolidierten Schadendaten. Der beschreibende Freitext des Schadens wird dabei in einen Vektor umgewandelt und **mit der Pinecone-Datenbank abgeglichen**, um jene Regelwerke und Klauseln zu ermitteln, die semantisch am besten zur Schadenbeschreibung passen. Anschließend übermittelt das System diese identifizierten **Regeltexte zusammen mit den Schadendaten an das KI-Sprachmodell**, das auf Basis dieser Informationen **beurteilt, ob eine Deckung durch die Polizze besteht oder Ausschlüsse greifen** könnten. Diese Beurteilung erfolgt durch **logische Schlussfolgerungen** unter Berücksichtigung von Bedingungen wie Deckungssummen oder Selbstbehalten.

Abschließend liefert das System die **Entscheidungsempfehlung** (Deckung gegeben oder nicht), **ergänzt um eine nachvollziehbare Begründung** sowie die relevanten Textpassagen aus den Versicherungsbedingungen, zurück an das Frontend. Dieses Ergebnis wird der Sachbearbeiterinnen oder dem Sachbearbeiter übersichtlich dargestellt, wodurch die Transparenz und **Nachvollziehbarkeit** der KI-Entscheidung gewährleistet bleibt.

### 3.3. Realisierung / Implementierung

In diesem Abschnitt wird die konkrete **Realisierung** der Deckungsprüfung im Projekt beschrieben. Zunächst wird ein Überblick über die **Systemarchitektur** gegeben, gefolgt von Details zur verwendeten Regelwerksdatenbank (*viva\_regelwerke*) und deren Struktur. Anschließend werden die implementierten **Batch-Prozesse** zur Aktualisierung der Vektordatenbank erläutert. Der Kern dieses Kapitels ist die Beschreibung des **Ablaufs der Deckungsprüfung** aus technischer Sicht – insbesondere anhand der entwickelten API-Endpunkte, wobei der Fokus auf dem Endpunkt „/claimcoverage“ liegt. Zusätzlich wird der Endpunkt „/claimimages“ mittels eines Sequenzdiagramms dargestellt, da er für die Verarbeitung von Schadensbildern zuständig ist. Abschließend wird auf die **Dokumentenverarbeitung** und KI-gestützte **Zusammenfassung** eingegangen, die innerhalb der Deckungsprüfung eine wichtige Rolle spielt.

#### 3.3.1. Architektur der Deckungsprüfung

Die **Architektur** der Lösung ist modular aufgebaut und in Abbildung 3.1 dargestellt. Sie umfasst folgende Hauptkomponenten:

### Regelwerksdatenbank (viva\_regelwerke)

Eine relationale Datenbank, in der sämtliche relevanten Versicherungsbedingungen und Geschäftsregeln gespeichert sind. Hier liegen die Gesetztestexte, Vertragsklauseln, Deckungsausschlüsse, Bedingungen etc., die für die Deckungsprüfung herangezogen werden. Die Datenbank dient als *Single Source of Truth* für das Fachwissen, das die KI nutzt.

### Batch-ETL-Komponente

Ein Prozess (oder Dienst), der periodisch die neuesten Änderungen aus der Regelwerksdatenbank extrahiert, in Vektoren umwandelt und in die Pinecone-Datenbank lädt. Diese Komponente stellt sicher, dass die KI immer auf dem aktuellen Stand der Versicherungsbedingungen arbeitet.

### Vektordatenbank (Pinecone-Index)

Ein externer Cloud-Service (Pinecone), der die Embeddings der Regelwerkstexte speichert und Ähnlichkeitssuchen ermöglicht. Der Pinecone-Index enthält pro Regel einen oder mehrere Vektoreinträge sowie zugehörige Metadaten (z.B. eine ID oder Regelwerktags). Er kann sehr schnell die *Top-N* ähnlichsten Einträge zu einer Anfrage finden.

### Deckungsprüfungs-Backend (FastAPI)

Der zentrale Anwendungsserver, der die Logik der Deckungsprüfung implementiert und nach außen hin als API zur Verfügung stellt. Dieses Backend orchestriert die Abläufe: Es empfängt Anfragen (etwa eine neue Schadennummer zur Prüfung), holt benötigte Daten aus Datenbank und Pinecone, ruft den KI-Dienst an und bereitet die Antwort auf. Implementiert wurde es mit FastAPI in Python, wodurch es leichtgewichtig und performant ist.

### KI-Service (Azure OpenAI)

KI-Service ist der angebundene Dienst, der die komplexen NLP-Aufgaben übernimmt. Das Backend kommuniziert über definierte Endpunkte mit dem Azure OpenAI Service, um Texte zu **analysieren** (z.B. wichtige Informationen aus einem Fließtext zu extrahieren) und zu **generieren** (z.B. eine Antwort formulieren, ob Deckung besteht). Durch die Nutzung von Azure OpenAI muss das Modell selbst nicht gehostet werden; Skalierung und Performance werden von Azure gehandhabt.

### Schadenmanagement-System

Auf Seite der KLV existiert typischerweise ein Kernsystem für die Schadenverwaltung. In diesem System werden Schadensdaten (und Anhänge) angelegt. Durch den Aufruf unseres Frontends ist es der Benutzerin oder dem Benutzer danach möglich direkt auf alle gespeicherten Daten zuzugreifen. Die Benutzerin oder der Benutzer kann selbst entscheiden, welche bereits abgelegten Dokumente er in die KI-Deckungsprüfung mit einfließen lassen will.

Die Kommunikation zwischen den Komponenten erfolgt über **wohl definierte Schnittstellen**. Insbesondere spricht das FastAPI-Backend mit Pinecone über deren API (HTTP/REST) und mit Azure OpenAI über die REST-API bzw. das Python-SDK von Azure. Die Datentransfers sind durchgängig verschlüsselt (TLS). Zudem werden Zwischenergebnisse, wie etwa die von Pinecone zurückgelieferten Klauseltexte, im Backend nur temporär gehalten und können dort mit den eigentlichen Schadendaten zusammengeführt werden.

### 3.3.2. Datenbank *viva\_regelwerke* (Versicherungsregelwerke)

Die Datenbank *viva\_regelwerke* bildet das **fachliche Herz** der Anwendung, da in ihr das Regelwissen über die Versicherungsbedingungen vorgehalten wird. Abbildung 3.2 zeigt das UML-Diagramm der wichtigsten Entitäten und Beziehungen dieser Datenbank.

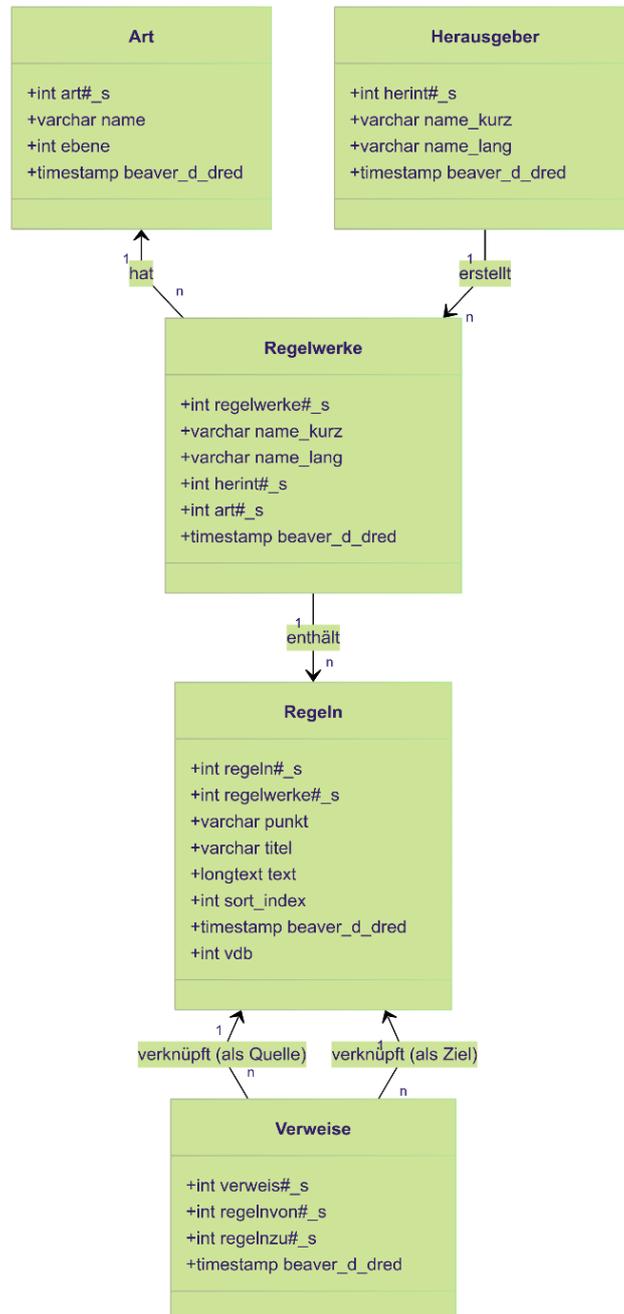


Abbildung 3.2 UML-Diagramm *viva\_regelwerke*

Eine Entität **Regelwerke** repräsentiert einen Satz von Versicherungsbedingungen, meistens zu einem bestimmten **Versicherungsprodukt** oder Tarif. Zum Beispiel könnte es ein Regelwerk für die Kfz-Vollkaskoversicherung 2023 geben, ein anderes für die Haushaltsversicherung 2015 etc. Jedes Regelwerk hat Metadaten wie einen Kurz-Namen, einen Namen, einen Herausgeber und eine Art des Regelwerks.

Jedes Regelwerk besteht aus mehreren **Regeln** (Paragrafen, Artikeln, Punkten, usw.). Die Entität **Regel** enthält typischerweise ein Textfeld mit dem Wortlaut der Bedingung oder Klausel, sowie eventuell Attribute wie Paragrafen-Nummer, Artikel und eine Verknüpfung (Foreign Key) zum übergeordneten Regelwerk. Eine Regel könnte z.B. lauten "Schäden durch grobe Fahrlässigkeit sind vom Versicherungsschutz ausgeschlossen."

Es gibt weitere Entitäten zur Klassifizierung. So könnten **Art** oder **Herausgeber** von Regeln erfasst werden, um ähnliche Klauseln zu gruppieren (etwa alle Gesetze). Die Zuordnung jeder einzelnen Regel wird anhand der im Versicherungsvertrag hinterlegten Regelungen bestimmt. Regeln können aber wiederum auf andere Regelungen verweisen, auch diese Verweise sind hinterlegt und werden in die Entscheidung der Auswahl mit einbezogen.

**Regelverwaltung**

Herausgeber: KLV (N B L) | Art der Regel: Allgemeine Bedingung (5) (N B L)

Regelwerke: ARB2018, Allgemeine Bedingungen für die Rechtsschutz-Versicherung (AR...) (N B L) | Regeln: Artikel 2 (N S L)

Punkt: Artikel 2 | Titel: Was gilt als Versicherungsfall und wann gilt er als eingetreten?

Text:

1. Für die Geltendmachung eines Personen-, Sach- oder Vermögensschadens, der auf einen versicherten Personen- oder Sachschaden zurückzuführen ist (Artikel 17.2.1., Artikel 18.2.1., Artikel 19.2.1. und Artikel 24.2.), gilt als Versicherungsfall das dem Anspruch zugrundeliegende Schadenereignis. Als Zeitpunkt des Versicherungsfalles gilt der Eintritt dieses Schadenereignisses. Bei Schäden infolge einer Umweltstörung, die auf einen vom ordnungsgemäßen, störungsfreien Betriebsgeschehen abweichenden, einzelnen, plötzlich eingetretenen Vorfall zurückzuführen sind, gilt dieser Vorfall (Störfall) als Versicherungsfall. Als Zeitpunkt des Versicherungsfalles gilt der Eintritt dieses Störfalles. Umweltstörung ist die Beeinträchtigung der Beschaffenheit von Luft, Erdreich oder Gewässern.

2. Im Beratungs-Rechtsschutz (Artikel 22.3.) und in bestimmten Fällen des Rechtsschutzes für Grundstückseigentum und Miete (Artikel 24.4.), des Rechtsschutzes für Familienrecht (Artikel 25.4.), des Daten-Rechtsschutz (Artikel 27.4.) und im Steuer-Rechtsschutz (Artikel 28.4.) gelten die dort beschriebenen Sonderregelungen für den Versicherungsfall.

Sort Index: 1 | Regeln ID: 619 | Letzte Änderung: 2025-02-08T16:48:20

**Verweise von Artikel 2**

VID	Regelwerk	Regel ID	Punkt	Titel
1	ARB2018	634	Artikel 17	Schadenersatz-, Straf- und Fahrschein Rechtsschutz - je nach Vereinbarung - mit oder ohne Fahrzeug Vertrags-Rechtsschutz für Motorfahrzeuge (Fahrzeug Rechtsschutz)
2	ARB2018	635	Artikel 18	Schadenersatz-, Straf- und Fahrschein Rechtsschutz für Fahrzeuglenker (Lenker- Rechtsschutz)
3	ARB2018	636	Artikel 19	Schadenersatz- und Straf-Rechtsschutz für den Privat-, Berufs- und Betriebsbereich
4	ARB2018	639	Artikel 22	Beratungs-Rechtsschutz
5	ARB2018	640	Artikel 23	Allgemeiner Vertrags- Rechtsschutz
6	ARB2018	641	Artikel 24	Rechtsschutz für Grundstückseigentum und Miete
7	ARB2018	642	Artikel 25	Rechtsschutz für Familienrecht
8	ARB2018	644	Artikel 27	Daten-Rechtsschutz

Abbildung 3.3 Frontend zur Wartung der viva\_regelwerke Datenbank

## Beziehungen

Zwischen *Regelwerk* und *Regel* besteht eine 1-n Beziehung – jedes Regelwerk hat viele Regeln, aber jede Regel gehört genau zu einem Regelwerk. Jedes Regelwerk hat wiederum genau einen Herausgeber und eine Art. Jede Regel kann auch mehrere Verweise auf mehrere andere Regelungen haben. Im UML-Diagramm (Abbildung 3.2) sind diese Relationen durch entsprechende Verbindungslinien dargestellt.

Die Inhalte der Tabellen werden von Fachexperten gepflegt. Für die Anwendung ist insbesondere relevant, dass aus jeder *Regel* der **relevante Text** extrahiert werden kann, da dieser später für die Embeddings genutzt wird. In vielen Fällen sind die Klauseltexte Freitexte mit juristischer Sprache. Hierfür steht ein eigenes Frontend (Abbildung 3.3) zur Verfügung.

Diese strukturierte Ablage ermöglicht es, gezielt Klauseltexte zu laden. Für die KI-Anwendung mussten die relevanten Tabellen (vor allem die Regel-Texte) aus *viva\_regelwerke* extrahiert werden. Außerdem kann man über die Regelwerke später filtern (z.B. für diese Polizze gilt für diesen Schaden das Regelwerk ARB2018).

### 3.3.3. Batch-Prozesse zur Pinecone-Aktualisierung

Damit die Pinecone-Vektordatenbank stets die aktuellen Regelwerke widerspiegelt, wurden **Batch-Prozesse** eingerichtet, die regelmäßig für eine **Datenaktualisierung** sorgen. Die Notwendigkeit ergibt sich daraus, dass Versicherungsbedingungen Änderungen unterliegen können (neue Tarife, geänderte Klauseln, rechtliche Anpassungen etc.). Der Ablauf eines solchen Batch-Prozesses gestaltet sich in etwa so:

#### Extraktion der Daten

In definierten Intervallen (z.B. jede Nacht) oder bei bestimmten Events (z.B. manuelle Auslösung nach einer Regelwerksänderung) werden alle relevanten Klauseltexte aus der *viva\_regelwerke*-Datenbank ausgelesen. Über eine eigens in den Regeln definierte Spalte, erkennt der Batch-Job ob eine Regel geändert wurde oder ob eine neue Regel eingefügt wurde.

#### Embedding-Erzeugung

Die extrahierten Texte werden dann durch ein **Embedding-Modell** geleitet, um numerische Vektoren zu erhalten. Hierfür kann ein vortrainiertes Modell von OpenAI genutzt werden (z.B. *text-embedding-3-large*) über die Azure OpenAI API. Der Prozess iteriert über jede Klausel (Regel) und ruft die Embedding-API auf, welche einen hochdimensionalen Vektor zurückliefert, der den semantischen Inhalt des Textes repräsentiert.

#### Index-Update in Pinecone

Die neu berechneten Embeddings werden dann in den Pinecone-Index **upsert**et (eingefügt oder aktualisiert). Pinecone erlaubt Batch-Upserts, sodass auch tausende von Einträgen effizient geladen werden können. Dabei wird jeder Vektor mit einer eindeutigen ID (z.B. der *regel\_id*) und optional Metadaten gespeichert. Metadaten könnten z.B. das *regelwerk\_id* oder die Kategorie sein, um spätere Filterungen zu ermöglichen. Wenn eine Klausel bereits existierte (gleiche ID), überschreibt der Upsert den alten Vektor (falls sich der Text geändert hat),

oder lässt ihn unverändert wenn kein Change. Neue Klauseln werden hinzugefügt, obsolet gewordene könnten bei Bedarf gelöscht werden.

### Verification & Logging

Nach dem Upsert kann der Batch-Prozess verifizieren, dass die Anzahl der Vektoren im Pinecone-Index der Anzahl der aktiven Klauseln entspricht, um Konsistenz sicherzustellen. Es wird ein Log oder Report erzeugt, damit nachvollziehbar ist, welche Änderungen durchgeführt wurden (z.B. "10 neue Klauseln hinzugefügt, 2 geändert, 1 gelöscht").

Durch diese Batch-Verarbeitung wird erreicht, dass *Pinecone* immer einen **aktuellen Wissensstand** hat. Die Deckungsprüfung selbst (zur Laufzeit) muss somit nicht direkt die relationale DB befragen, sondern arbeitet mit dem schnellen Vektorindex. Das entlastet die Datenbank und vereinfacht die Architektur, da zur Abfragezeit nur noch Pinecone und ggf. ein Cache genutzt werden. Die Frequenz der Updates kann je nach Änderungsaufkommen gewählt werden; in einer produktiven Umgebung könnte man z.B. bei seltenen Änderungen auch wöchentliche Batches fahren oder sie an das Deployment von neuen Bedingungswerken koppeln.

The screenshot displays the FastAPI API documentation for a system. At the top, it shows 'FastAPI 0.1.0 OAS 3.1' and a file path '/openapi.json'. Below this, there is a list of endpoints, each with a method (GET or POST), a path, and a brief description. The endpoints are grouped into sections: Serverstatus, Schadensabfrage, Schadensmeldungabfrage, Schadensbilderabfrage, Schadensrechnungsabfrage, Schadensgutachtenabfrage, Schriftverkehrabfrage, Dokumentenzusammenfassung, AI Report zum Schaden, and Deckungsprüfung. Each endpoint is represented by a blue box with a 'GET' or 'POST' label, a path, and a description. The 'Deckungsprüfung' endpoint is highlighted with a green background.

Method	Path	Description
GET	/status	Gibt den Status der Datenbank zurück
GET	/claim/{claim_id}	Gibt die gesamten Schadendaten für das Frontend zurück
GET	/claimreport/{claim_id}	Gibt die Schadenmeldungs-Dokumente für das Frontend zurück
GET	/claimimages/{claim_id}	Gibt die gesamten Bilder für das Frontend zurück
GET	/claimbill/{claim_id}	Gibt die Rechnungen/Kostenvorschläge für das Frontend zurück
GET	/claimreview/{claim_id}	Gibt die Gutachten/Protokolle für das Frontend zurück
GET	/claimmailing/{claim_id}	Gibt den Schriftverkehr für das Frontend zurück
GET	/claimdocs/{claim_id}	Gibt alle Dokumentenzusammenfassungen an das Frontend zurück
GET	/claimreport/{claim_id}	Gibt den AI Report als JSON zurück
POST	/claimcoverage/{claim_id}	Gibt die Deckungsprüfung für das Frontend zurück

Abbildung 3.4 Wichtige API-Endpunkte

### 3.3.4. Ablauf der Deckungsprüfung (API-Endpunkte)

Die entwickelte Lösung stellt mehrere **API-Endpunkte** bereit, über die die Deckungsprüfung und zugehörige Funktionen ausgelöst werden können. Abbildung 3.4 gibt einen Überblick über die wichtigsten Endpunkte und deren Zweck. Der zentrale Endpunkt ist dabei „**/claim-coverage**“, welcher die eigentliche Deckungsprüfung für einen gegebenen Schadenfall durchführt. Daneben existiert u.a. „**/claimimages**“, der speziell zur Verarbeitung von Bildanhängen dient (siehe Sequenzdiagramm in Abbildung 3.5). Im Folgenden wird der Ablauf von „**/claim-coverage**“ im Detail beschrieben, da hier die Kernlogik umgesetzt ist.

#### 3.3.4.1. Endpunkt „/claimcoverage“

Dieser Endpunkt kapselt den vollständigen Deckungsprüfungs-Workflow. Ein Aufruf von „**/claimcoverage**“ führt intern die folgenden Schritte aus:

##### Eingabe entgegennehmen

Der Service erhält eine Schadennummer und ein Datenobjekt mit allen relevanten Informationen zum Schaden. Im einfachsten Fall reicht die Schadennummer, sofern der Service darüber mit dem Schadenmanagement-System verbunden ist und darüber die Details abrufen kann. Nach Eingang der Anfrage werden zunächst alle nötigen **Schadendaten** bereitgestellt. Dazu gehört auch, den zugehörigen Versicherungsvertrag im System zu identifizieren (über Polizzenummer und Vertragsnummer), da die Deckungsprüfung sich immer auf einen bestimmten Vertrag innerhalb einer Police beziehen muss.

##### Dokumenten- und Bildvorverarbeitung

Falls zum Schadenfall **Dokumente oder Bilder** mitgeliefert wurden (etwa Fotos vom Schadenort, PDF-Berichte, Rechnungen), werden diese nun verarbeitet. Dokumente werden via OCR in Text umgewandelt (dies übernimmt intern einer der Endpunkte oder eine entsprechende Funktionalität. Dokumente in Textform (z.B. ein mehrseitiger Bericht als PDF) können direkt vom KI-Modul verarbeitet werden. Gegebenenfalls wird das Dokument ebenfalls per OCR erfasst, falls es sich um eingescannte Seiten handelt. Anschließend kommt eine **KI-gestützte Zusammenfassung** zum Einsatz: Längere Freitexte werden durch Azure OpenAI inhaltlich zusammengefasst, um die wesentlichen Fakten herauszufiltern. Dabei extrahiert das Modell wichtige Informationen wie *Was ist passiert?*, *Welches Objekt ist betroffen?*, *Wie ist die Schadenursache?* etc. (Details dazu im Abschnitt Dokumentenverarbeitung). Ergebnis der Vorverarbeitung ist ein konsolidierter **Schadentext**, der alle relevanten Informationen in komprimierter Form enthält – bestehend aus der ursprünglichen Schadenbeschreibung (sofern vorhanden) und den gewonnenen Informationen aus Anhängen.

##### Generierung einer Abfrage (Query-Formulierung)

Auf Basis dieser konsolidierten Schadensinformation wird nun eine semantische **Suchanfrage** für das Regelwerk formuliert. Diese Query ist typischerweise ein Text (oder direkt ein Embedding), der die Schlüsselbegriffe und -aspekte des Schadenfalls enthält. Beispielsweise könnte die zusammengefasste Beschreibung lauten: *„Überschwemmung im Keller eines Einfamilienhauses durch Starkregen am 12.08.2023.“* Aus einem solchen Text extrahiert das System intern die wichtigsten Komponenten: Schadenart = Überschwemmung, Schadenursache = Starkregen (Unwetter), betroffenes Objekt = Gebäude (Keller). Diese Informationen werden ggf.

nochmals als kompakter Prompt an das Embedding-Modell gegeben, um den Query-Vektor zu erhalten. Dieser Schritt stellt sicher, dass **Kernbegriffe** und **Zusammenhänge** des Falls in den Suchvektor einfließen. [41] Die Suche wird auf das Regelwerk eingeschränkt, welches dem aktuellen Vertrag zugrunde liegt (z.B. mittels Pinecone-Metadaten-Filter auf eine bestimmte regelwerk\_id).

### Semantische Suche in den Regelwerken

Nun wird die vorbereitete Anfrage an Pinecone geschickt, um die **ähnlichsten Klauseln** zu finden. Pinecone gibt eine Liste von z.B. top 5 oder top 10 Treffern (in config.py einstellbar) zurück, jeweils mit einer Ähnlichkeitskennzahl (Cosine-Similarity). Diese Treffer werden dann aus der *viva\_regelwerke*-Datenbank oder einem Cache geladen, sodass der volle Klauseltext zur Verfügung steht. Hier zeigt sich die Stärke des Ansatzes: Selbst wenn im Schadentext andere Worte stehen als in den Versicherungsbedingungen, findet die Vektorsuche relevante Übereinstimmungen auf inhaltlicher Ebene. So könnte bei Schaden "Überschwemmung durch Starkregen" die Klausel "Elementarschäden (Überschwemmung, Rückstau etc.) sind mitversichert" gefunden werden, obwohl exakt das Wort "Starkregen" evtl. nicht in der Klausel steht. Falls mehrere unterschiedliche Aspekte vorliegen, erscheinen entsprechend verschiedene Klauseln (Deckungserweiterungen, Ausschlüsse etc.) in den Treffern. Im Ergebnis hat das System nun einen kleinen Satz an **potenziell relevanten Versicherungsbedingungen** extrahiert [40], der als Wissensgrundlage für die KI-Entscheidung dient.

### KI-Analyse der Deckung

Die eigentliche Deckungsentscheidung wird durch das Sprachmodell (Azure OpenAI) vorbereitet. Dazu erstellt das Backend einen **Prompt**, der die Aufgabe klar definiert. Vereinfacht könnte der Prompt so aussehen: "Gegeben: 1) Schilderung des Schadenfalls: [Text]; 2) Auszüge aus Versicherungsbedingungen: [Klausel A], [Klausel B], ... . Frage: Ist der geschilderte Schadenfall gemäß diesen Bedingungen gedeckt? Antworte mit einer Begründung." Das KI-Modell erhält also sowohl die Schadenbeschreibung (bzw. deren Zusammenfassung) als auch die gefundenen Klauseltexte als Input. Anschließend generiert es eine **Zusammenfassung mit Entscheidung**. Beispiel einer KI-Antwort: "  **Deckung** - Da es sich um einen Überschwemmungsschaden handelt, und gemäß AStB Artikel 2.2 solche Schäden versichert sind." Oder im Ablehnungsfall: "  **Keine Deckung** - Laut Bedingungen (AStB Artikel 2.2) sind Überschwemmungen zwar versichert, jedoch liegt grobe Fahrlässigkeit vor (Fenster war offen), was in AStB Artikel 7 als Ausschluss definiert ist." – Das Modell zieht also Schlussfolgerungen aus den Bedingungen und dem Sachverhalt. Wichtig ist, dass das Modell nur auf Basis der gegebenen Klauseln urteilt und keine **halluzinierten** Regeln einbringt. Durch die Begrenzung auf echte Klauseltexte wird die Gefahr reduziert, dass das Modell etwas Falsches behauptet. Dieser Schritt der KI-Analyse entspricht dem, was auch in der Fachliteratur als sinnvoll erachtet wird: semantische Textanalyse der Verträge, um daraus automatisiert eine Deckungsentscheidung abzuleiten. [40]

### Ergebnisaufbereitung und -ausgabe

Die vom KI-Modell erstellte Antwort wird vom Backend entgegengenommen und in ein strukturiertes **Response-Format** gegossen. Typischerweise besteht die Antwort aus einem, einer **Begründung** sowie Referenzen auf die maßgeblichen Klauseln. Der gesamte Text ist für eine optimierte Anzeige im Frontend bereits **im Markdown formatiert**. Diese Information wird als

JSON oder als Text zurück an den Aufrufer geschickt. Ein Beispiel-Response (JSON siehe Listing 3.1):

```
{
  "Deckungsprüfung": "##  Deckung\n\n### Begründung:\n ...."
}
```

Listing 3.1 JSON-Response der Deckungsprüfung

Der aufrufende Client (z.B. unser Frontend) kann diese Informationen nutzen, um sie der Sachbearbeiterinnen oder dem Sachbearbeiter anzuzeigen (oder direkt automatisierte Entscheidungen zu treffen). Dezeit wird die Entscheidung durch die Sachbearbeiterinnen oder den Sachbearbeiter getroffen; das System kann z.B. das KI-Ergebnis präsentieren und die Sachbearbeiterinnen oder der Sachbearbeiter bestätigt oder überschreibt es. Durch die mitgelieferten Klauselreferenzen bleibt der **Entscheidungsprozess transparent** und nachvollziehbar.

Insgesamt ermöglicht der Endpunkt „/claimcoverage“ somit eine End-to-End-Automatisierung: von der Eingabe eines neuen Schadens bis zur Ausgabe einer Deckungseinschätzung vergeht nur sehr wenig Zeit, ohne dass manuell in Vertragsbedingungen nachgeschlagen werden muss. Die Kombination aus Vektorensuche und Sprach-KI stellt sicher, dass sowohl **explizites Wissen** (die niedergeschriebenen Regeln) als auch **implizites Verständnis** (Interpretation der freien Schilderung) zusammenwirken, um das Ergebnis zu erzielen.

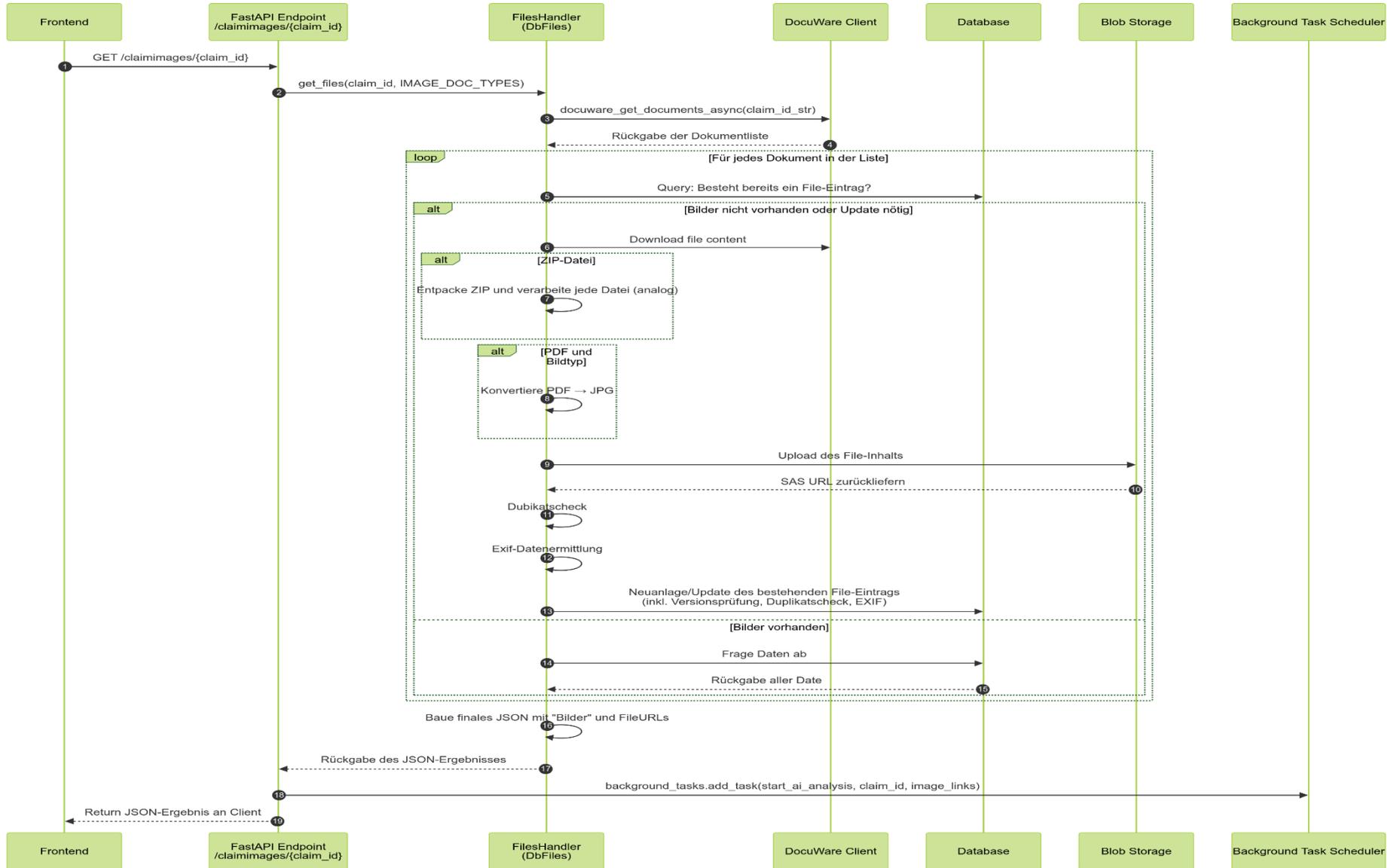


Abbildung 3.5 Sequenzdiagramm des Endpunktes „/claimimage/{claim\_id}“

### 3.3.4.2. Endpunkt „/claimimages“

Während „/claimcoverage“ den Gesamtprozess abbildet, erfüllt der Endpoint „/claimimages“ eine unterstützende Spezialfunktion – nämlich die Verarbeitung von Bilddateien, die im Rahmen einer Schadenmeldung hochgeladen werden. Abbildung 3.5 zeigt das zugehörige Sequenzdiagramm. Der typische Ablauf beim Aufruf von „/claimimages“ ist folgender:

**Ein Client** (unser Frontend) ruft die API am Endpoint „/claimimages“ auf und übergibt die **Schadensnummer**, zu der Bilder vorliegen.

Das FastAPI-Backend empfängt die Anfrage und **lädt über den FilesHandler (DbFiles) die entsprechenden Bilder**. Hierbei nutzt das System die **Schadensnummer**, um über den angebotenen **DocuWare-Client** die hinterlegten Bilddateien abzurufen – in vielen Versicherungssystemen werden Fotos von Schäden in einem separaten Dokumentenmanagement oder Blob Storage gespeichert, sodass ein API-Call notwendig ist, um diese Dateien zu holen.

**Für jedes empfangene Bild** wird zunächst geprüft, ob bereits ein **File-Eintrag in der Datenbank** vorliegt, der alle notwendigen Bearbeitungsschritte (wie Duplikatscheck, EXIF-Datenermittlung, Speicherung im Blob Storage) durchlaufen hat. Falls nicht, erfolgt der **Download des File-Inhalts vom DocuWare-System**. Bei Bedarf wird z.B. eine **ZIP-Datei entpackt** oder ein **PDF in ein JPG** konvertiert. Anschließend wird der Inhalt in den **Blob Storage hochgeladen**, wobei als Ergebnis eine **SAS URL zurückgeliefert** wird. Liegen die Bilder bereits vor, so werden direkt die existierenden Daten aus der Datenbank abgerufen.

Die aus den Bildern extrahierten bzw. **zugeordneten Informationen** werden gesammelt und in einem **finalen JSON-Objekt zusammengefasst** – dieses enthält beispielsweise ein Array von Bilddaten mit den zugehörigen FileURLs. **Gleichzeitig wird ein Background Task gestartet** (über den Background Task Scheduler), der den Prozess der **KI-gestützten Analyse** (start\_ai\_analysis) anstößt.

Abschließend wird das **JSON-Ergebnis** an das Frontend zurückgegeben. Dieser Response enthält alle relevanten Bildinformationen, sodass der aufrufende Client entscheiden kann, wie mit den gewonnenen Daten weiter verfahren wird.

Durch diesen separaten Ablauf für Bilder wird der **Hauptprozess der Deckungsprüfung entlastet** und gleichzeitig sichergestellt, dass Bilddaten nicht ungenutzt bleiben. Gerade in der Schadenregulierung werden z.B. Rechnungen, Kostenvoranschläge oder Fotos von beschädigten Objekten eingereicht – eine reine Textanalyse würde diese Informationen ignorieren. Mit „/claimimages“ und den anderen Endpunkten zur Auswertung der Dokumente steht eine Komponente zur Verfügung, die solche **Inputs maschinell verwertbar** macht und damit zur Vollständigkeit der Informationen für die Deckungsentscheidung beiträgt.

Das zugehörige Sequenzdiagramm (siehe **Fehler! Verweisquelle konnte nicht gefunden werden.**) verdeutlicht, dass nach dem Abruf und der Verarbeitung der Bilddateien und FileURLs

zurück an den aufrufenden Prozess fließen. Intern kann so der „/claimcoverage“ die Auswertung der Dokumente und Bilder nutzen, um zunächst alle Dokumente und Bilder auszuwerten, bevor er mit der semantischen Suche fortfährt.

### 3.3.5. Dokumentenverarbeitung und AI-gestützte Zusammenfassung

Neben strukturierten Daten und Bildern gibt es in Schadenfällen oft auch umfangreiche **Dokumente** oder längere Freitexte (z.B. E-Mails, PDF-Berichte). Die effiziente Verarbeitung dieser Informationen ist ein weiterer wichtiger Aspekt der Implementierung. Im Projekt "KLV goes AI" wird daher auf **AI-gestützte Zusammenfassungen** gesetzt, um lange Texte auf das Wesentliche einzudampfen.

Die Vorgehensweise ist wie folgt: Liegt ein langer Text vor (sei es aus einem Fließtextfeld oder aus OCR eines Dokuments), so wird dieser an das Azure OpenAI Sprachmodell mit der Aufforderung übergeben, eine **Zusammenfassung** bzw. **strukturierte Extraktion** durchzuführen. Das Sprachmodell kann z.B. angewiesen werden: *"Fasse den folgenden Schadensbericht zusammen und nenne relevante Informationen für eine Deckungsprüfung."* Dadurch erhalten wir einen deutlich kürzeren Text, der die wichtigsten Fakten enthält. Diese Zusammenfassung wird dann – wie in den obigen Schritten beschrieben – für die semantische Suche und Entscheidungsfindung herangezogen. Der Vorteil liegt auf der Hand: Selbst wenn ein Gutachten 10 Seiten umfasst, muss die KI im Deckungsprüfungsschritt nicht den gesamten Text gegen die Bedingungen matchen, sondern nur die kondensierte Form. Das reduziert die **kognitive Last** für das Modell und auch das Fehlerrisiko, weil irrelevante Details ausgefiltert werden.

Ein konkretes Beispiel: Ein medizinischer Bericht listet ausführlich Befunde und Historie eines Patienten auf, relevant für die Versicherung ist aber nur, **ob die behandelte Diagnose unter den versicherten Leistungen fällt**. Die Zusammenfassung könnte hier z.B. ergeben: "Patient erlitt am 01.03.2023 einen Beinbruch (Unfall) und wurde operativ versorgt." – Diese wenigen Sätze genügen, um in den Bedingungen nachzuschlagen, ob Unfallverletzungen gedeckt sind und ob bestimmte Ausschlüsse (z.B. Extremsport) greifen. Das Modell extrahiert also die **Schlüsselinformationen** (Datum, Art des Ereignisses, relevante Kennzahlen).

In internen Tests hat sich gezeigt, dass die Zusammenfassungsqualität von GPT-basierten Modellen sehr hoch ist – selbst aus unstrukturierten oder leicht fehlerhaften OCR-Texten kann das Modell Sinn konstruieren und geordnete Informationen erzeugen. Die Verwendung solcher **AI-Summarization**-Techniken ist ein entscheidender Schritt, um die **Praxistauglichkeit** der automatisierten Deckungsprüfung zu gewährleisten: Der letztendliche Entscheidungsalgorithmus bekommt nur noch relevante Inputs und kann dadurch ähnlich zielgerichtet arbeiten wie eine menschliche Sachbearbeiterin oder Sachbearbeiter, der auch zuerst die wichtigen Punkte aus einem Wust von Informationen herausfiltert.

Zusammenfassend kombiniert die Implementierung also mehrere Methoden: klassische Datenbankabfragen, semantische Vektorsuche, Bilderkennung (OCR) und generative KI für Textzusammenfassung und -analyse. Diese **Methodenintegration** ermöglicht es, den gesamten

Prozess der Deckungsprüfung abzubilden und zu automatisieren, von der Eingabe unterschiedlicher Datenquellen bis zur Ausgabe einer fundierten Entscheidung.

### **3.4. Ergebnisse / Konklusion**

Abschließend werden die erzielten **Ergebnisse** der Implementierung zusammengefasst und bewertet. Zudem wird ein Ausblick auf mögliche **Erweiterungen** gegeben.

#### **3.4.1. Erreichte Funktionalitäten**

Im Rahmen des Projekts "KLV goes AI" konnte erfolgreich eine prototypische Deckungsprüfungs-Engine inkl. Betrugserkennung entwickelt werden, die folgende Kernfunktionen umfasst:

##### **Automatisierte Deckungsentscheidung**

Das System ist in der Lage, selbständig anhand der angegebenen Schadendaten und hinterlegten Versicherungsbedingungen eine Einschätzung abzugeben, ob ein Schadenfall vom Versicherungsschutz gedeckt ist. Die Entscheidung wird zusammen mit einer Begründung und den relevanten Klauseltexten bereitgestellt, sodass sie nachvollziehbar ist.

##### **Semantische Regelwerk-Suche**

Durch den Einsatz der Pinecone-Vektordatenbank werden aus einer freien Schadenbeschreibung zuverlässig thematisch passende Klauseln gefunden, selbst wenn die Wortwahl nicht exakt übereinstimmt. Dies stellt sicher, dass keine manuelle Recherche in umfangreichen Bedingungswerken mehr nötig ist, sondern relevante Auszüge auf Knopfdruck vorliegen.

##### **KI-basierte Textanalyse**

Unstrukturierte Eingaben – seien es längere Fließtexte in der Schadenmeldung oder seitenlange Dokumente – werden mittels KI zusammengefasst und analysiert. Wichtige Informationen wie Schadenhergang, Ursache, Beteiligte und Umfang werden automatisiert extrahiert. Dadurch erhält das System eine strukturiertere Datengrundlage für die Prüfung als es die Rohdaten bieten.

##### **Bildverarbeitung (OCR)**

Eingereichte Dokumente werden durch Prozesse auslesbar gemacht. Texte, die auf Rechnungen, Gutachten oder Formblättern enthalten sind, werden erkannt und in die Deckungsprüfung einbezogen. Damit deckt die Lösung verschiedene Datentypen (Text, Bild) ab, die in realen Schadensfällen vorkommen.

##### **API-Integration**

Die gesamte Funktionalität ist über definierte API-Schnittstellen verfügbar. Dies erleichtert die Integration in bestehende Abläufe – z.B. in das bereitgestellte Frontend den Endpunkt /claim-coverage unmittelbar nach Eingang einer digitalen Schadenmeldung aufrufen. Die Ergebnisse können direkt dort weiterverarbeitet oder die Sachbearbeiterinnen oder dem Sachbearbeiter angezeigt werden. Durch die klare Trennung in Endpunkte (z.B. separater Aufruf für Bilder) ist die Nutzung flexibel und einzelne Module können auch unabhängig verwendet werden.

## Nachvollziehbarkeit und Dokumentation

Jede automatisierte Entscheidung wird durch Angabe der herangezogenen Vertragsklauseln untermauert. Dadurch lässt sich auch gegenüber Kundinnen und Kunden oder Auditoren transparent machen, auf welcher Grundlage eine Entscheidung getroffen wurde.

### 3.4.2. Leistungsbewertung

Im Folgenden werden die Ergebnisse der Leistungsbewertung anhand der Kriterien Genauigkeit, Geschwindigkeit sowie Grenzen und Qualität des entwickelten Prototyps zusammengefasst.

#### Genauigkeit

Bei Tests des Prototyps mit historischen Schadensfällen zeigte sich eine **hohe Genauigkeit** der automatisierten Deckungsprüfung. In einer Vielzahl von Fällen stimmte die KI-Einschätzung mit der Einschätzung einer erfahrenen Sachbearbeiterin oder Sachbearbeiters überein. Insbesondere *standardisierte Schäden* (wie Sturm-/Hagelschäden in der Gebäudeversicherung, Kfz-Unfälle etc.) konnten korrekt als gedeckt oder nicht gedeckt klassifiziert werden. Fälle mit eindeutigen Ausschlüssen (z.B. grobe Fahrlässigkeit, Leistungsbegrenzungen) wurden in den meisten Fällen vom System erkannt, da die einschlägigen Klauseln zuverlässig aus dem Vektorindex geliefert und vom Modell korrekt interpretiert wurden. Die **semantische Suche** erwies sich dabei als sehr hilfreich: selbst wenn die Kundenbeschreibung unpräzise war, fand Pinecone oft Klauseln, die zumindest nahe am Thema lagen, was dem Sprachmodell Anhaltspunkte gab. Natürlich gab es auch schwierige Fälle: Bei sehr komplexen Konstellationen oder seltenen Spezialdeckungen musste das Modell manchmal passen oder lieferte nur unsichere Antworten. Hier zeigt sich, dass die KI keine Domain-Erfahrung in dem Sinne hat, sondern strikt auf den gegebenen Texten operiert – fehlen diese oder sind sie mehrdeutig, kann die Antwort unklar sein. Insgesamt kann die Genauigkeit als vielversprechend bewertet werden, auch wenn eine **endgültige Validierung** mit einer größeren Stichprobe von echten Schadensfällen noch aussteht.

#### Geschwindigkeit

Hinsichtlich der **Geschwindigkeit** kann festgehalten werden, dass die automatische Deckungsprüfung enorm zeitsparend ist. In Tests lag die durchschnittliche Antwortzeit für einen kompletten „/claimcoverage“-Durchlauf im Bereich von wenigen Sekunden (typisch 5–20 Sekunden). Dieser Wert kann je nach Komplexität leicht variieren (viele Anhänge oder extrem lange Dokumente verlängern die Verarbeitung etwas), liegt aber um Größenordnungen unter dem manuellen Prüfprozess, der oft Stunden oder Tage benötigt. Damit wäre es theoretisch möglich, einen Großteil der Schadenmeldungen **in Echtzeit** zu triagieren und zu beantworten. Dies hätte spürbare positive Auswirkungen auf die Kundenerfahrung – schnelle Rückmeldungen steigern die Zufriedenheit und damit die Kundenbindung. [42] Auch intern bedeutet die Automatisierung einen Effizienzgewinn: Sachbearbeiterinnen oder Sachbearbeiter können deutlich mehr Fälle pro Tag bearbeiten, da Routineentscheidungen durch die KI vorbereitet werden.

## Grenzen und Qualitätsbetrachtung

Trotz der guten Performance wurden auch Grenzen der aktuellen Implementierung sichtbar. So neigt das generative Sprachmodell in seltenen Fällen dazu, **Halluzinationen** zu erzeugen – beispielsweise eine Klausel zu verallgemeinern oder Details hinzuzudichten, wenn die Eingaben lückenhaft sind. Dies wurde mitigiert, indem stets die Original-Klauseltexte mit ausgegeben werden und das Modell angewiesen wird, sich darauf zu beziehen. Zudem können **Ambiguitäten** in der Kundenschilderung zu Unsicherheiten führen: etwa wenn der Hergang unklar beschrieben ist, könnte das System vorschnell eine Deckung annehmen, obwohl bei genauerer Nachfrage herauskäme, dass ein Ausschluss greift. Solche Probleme sind allerdings auch in der menschlichen Bearbeitung bekannt. Durch eine geeignete Gestaltung des **Prompts** (z.B. der KI aufzutragen, im Zweifel "unsicher" zu antworten) und Schwellwerte kann gegengesteuert werden. Im aktuellen Prototyp wurde so implementiert, dass bei einer Unsicherheitsmeldung der KI etwaige Unklarheiten mit einem ⚠️ gekennzeichnet werden, sodass eine Mitarbeiterin oder ein Mitarbeiter diese erkennen kann – die KI agiert somit als Assistenzsystem und trifft keine endgültige Entscheidung. Insgesamt wurden die **Leistungsziele** erreicht: Die Deckungsprüfung läuft vollständig automatisiert, in akzeptabler Zeit und liefert inhaltlich plausible Ergebnisse.

### 3.4.3. Mögliche Erweiterungen für die Zukunft

Aufbauend auf dem Prototyp ergeben sich mehrere Ansatzpunkte, um die Lösung weiter zu verbessern und auszuweiten:

#### Feinjustierung und Training

Durch **Feintuning** des Sprachmodells auf unternehmensspezifische Schadenfälle könnte die Genauigkeit weiter erhöht werden. Man könnte historische Schadenfälle samt Entscheidungen nutzen, um das Modell gezielt auf die gewünschte Entscheidungsfindung zu trainieren. Auch die Verwendung von neueren Modellen (z.B. GPT-4.5 oder GPT-5) könnte die Qualität steigern, insbesondere bei komplexen Sprachverständnisaufgaben.

#### Weitere externe Datenquellen

Die Einbindung von **IoT- oder externen Daten** könnte die Entscheidungsgrundlage verbreitern. Zum Beispiel könnten Wetter-APIs genutzt werden, um bei einer Schadenmeldung die Wetterhistorie am Ort und Zeitpunkt abzurufen (zur Verifikation eines Sturmereignisses). Oder Fahrzeugtelematik-Daten könnten herangezogen werden, um einen Unfallhergang besser zu verstehen. Solche Daten ließen sich ebenfalls semantisch auswerten oder als strukturierte Fakten dem Modell geben.

#### Optimierung der Performance

Bei steigenden Nutzungszahlen könnte die Latenz weiter optimiert werden, etwa durch **Caching** von Embeddings (falls ähnliche Anfragen wiederholt kommen) oder das Vorberechnen von gewissen Prompt-Antwort-Kombinationen. Auch könnte man prüfen, ob bestimmte Teile des Prozesses parallelisiert werden können (z.B. OCR und schon Embedding-Erstellung der Beschreibung gleichzeitig). Skalierungstechniken in Azure (etwa mehr Throughput für das OpenAI API) wären in einem Livebetrieb relevant.

### User Interface und Feedback

Für die Praxis wäre ein **Benutzerfrontend** denkbar, in den Sachbearbeiterinnen oder Sachbearbeiter mit der KI-Lösung interagieren. Sie könnten dort z.B. Rückmeldungen geben – etwa markieren "KI-Entscheidung korrekt" oder "falsch" – um damit ein kontinuierliches Lernen zu ermöglichen. Diese *Human-in-the-loop*-Mechanismen würden helfen, das System im laufenden Betrieb zu verbessern und Vertrauen bei den Nutzern aufzubauen. Gleichzeitig könnten Erklärungsfunktionen (Explainable AI) ergänzt werden, die aufzeigen, warum die KI zu ihrer Empfehlung kam (z.B. Hervorhebung bestimmter Passagen in Klauseln).

### 3.4.4. Zusammenfassung

Zusammengefasst hat die Implementierung der Deckungsprüfung im Projekt gezeigt, dass **künstliche Intelligenz im Versicherungswesen** gezielt eingesetzt werden kann, um ehemals manuelle, zeitintensive Prozesse zu beschleunigen und qualitativ zu unterstützen. Die Ergebnisse demonstrieren bereits in diesem Prototypstadium ein hohes Potential: Schadenfälle können in Sekundenschnelle gegen komplexe Vertragsbedingungen geprüft werden – ein echter Mehrwert für Versicherer und Versicherungsnehmerinnen sowie Versicherungsnehmer. Für die Kärntner Landesversicherung bedeutet dies ein Schritt hin zu innovativer Schadenbearbeitung, bei der Routineaufgaben automatisiert und Mitarbeiterinnen und Mitarbeiter entlastet werden. Natürlich ist ein produktiver Einsatz an weitere Schritte gekoppelt (ausgiebige Tests, Anpassungen, Schulung der Mitarbeiterinnen und Mitarbeiter im Umgang damit), doch die **Grundlage** ist mit "KLV goes AI" geschaffen. Zukünftig könnten noch mehr Prozesse in der Schadensabwicklung – von der Betrugserkennung bis zur automatischen Reservenbildung – von ähnlichen AI-Technologien profitieren. Die Deckungsprüfung dient hier als **Pilotanwendung**, die den Weg für eine breitere Digitalisierung und KI-Durchdringung in der Versicherungsbranche ebnet.

## 4. Konfiguration einer cloudbasierten Projektinfrastruktur, sowie Implementierung eines KI-Systems mit zugehöriger API-Schnittstelle. (Michael Griesser)

KI-Anwendungen profitieren in vielen Fällen von den Vorteilen einer Cloud-Umgebung, da sie schnell auf leistungsstarke Rechenressourcen zugreifen und große Datenmengen effizient verarbeiten können. Wir haben uns entschieden das Projekt in der Azure Cloud zu realisieren.

### 4.1. Aufgabenstellung

Auf dem Server wird eine FastAPI-Anwendung implementiert, die als Schnittstelle zwischen der Azure-Infrastruktur und dem React-basierten Frontend dient. Die API stellt Endpunkte bereit, über die Anfragen empfangen, verarbeitet und Ergebnisse zurückgeliefert werden.

Ein weiterer Teil ist eine KI gestützte Bildanalyse mittels OpenAI und Erstellung eines zusammenfassenden Reports der analysierten Bilder.

Das Projekt legt den Grundstein für ein automatisiertes, KI-gestütztes Schadensfallmanagement, dass durch die Kombination aus einer robusten Cloud-Infrastruktur und einer leistungsfähigen API eine zukunftsweisende Lösung für die digitale Schadensbearbeitung darstellt.

### 4.2. Realisierung / Implementierung

Das Erstellen und Konfigurieren der Ressourcen in Microsoft Azure erfolgt fast ausschließlich über das Web Portal und gestaltet sich nach einer gewissen Einarbeitungszeit als relativ komfortabel. Aufgrund strenger Sicherheitskriterien ist der Zugang zum Cloud Portal nur einer Person gestattet. Um mit den Ressourcen zu interagieren ist es nicht nötig einen Zugang zu haben.

#### 4.2.1. Azure Cloud

In diesem Abschnitt werden die einzelnen Schritte für das Setup in der Azure Cloud erläutert. Die Konfiguration kann im Web Interface oder per „Command Line Interface“ (Azure CLI) durchgeführt werden.

#### Setup Azure DevOps

Azure DevOps bietet Entwicklerinnen und Entwicklern eine Möglichkeit Programmcode zentral zu lagern und zu verwalten. Auch das Projektmanagement kann über Azure DevOps realisiert werden und ersetzt somit Plattformen wie GitHub, GitLab oder Jira. In dem neu angelegten Projekt werden die Repositories erstellt. Hier wird darauf geachtet das Projekt so modular wie möglich aufzubauen. Deshalb wird ein Haupt Repository für die API erstellt. Für die weiteren Module, z.B. Frontend, werden eigene Repositories erstellt. GIT bietet mit der Funktion GIT Submodules die Möglichkeit, mehrere Repositories in einem Haupt Repository zu vereinen. So bleibt der Code übersichtlich, wartbar und leicht erweiterbar.

Für das Projektmanagement werden die Boards verwendet, die in jedem DevOps Projekt zur Verfügung stehen. Nach AGILE Regeln werden hier Issues, Epics und Sprints angelegt, verwaltet und abgearbeitet.

Ein weiteres Feature ist eine CI/CD Pipeline die automatische Integration und Bereitstellung ermöglicht.

### **Azure Subscription<sup>3</sup>**

Beim Erstellen einer Azure-Subscription spielt das Lizenzlevel eine zentrale Rolle, da es den Umfang der verfügbaren Dienste und die Abrechnungsmodelle bestimmt. Abhängig von den Projektanforderungen kann man beispielsweise zwischen dem Pay-As-You-Go-Modell oder einem Enterprise Agreement wählen, wobei letzteres oft zusätzliche Service-Level und Rabatte bietet. Diese Entscheidung beeinflusst maßgeblich den Zugang zu Ressourcen wie virtuellen Maschinen, Datenbanken und spezialisierten KI-Diensten und stellt sicher, dass die Infrastruktur optimal auf die spezifischen Bedürfnisse und das Budget des Projekts abgestimmt ist.

Das kostenlose Abonnement richtet sich an Einzelpersonen und kleine Unternehmen, die erste Erfahrungen mit der Plattform sammeln möchten. Es bietet \$200 Guthaben für die ersten 30 Tage sowie Zugang zu kostenlosen Diensten für 12 Monate. Aufgrund der eingeschränkten Funktionen und des Verzichts auf spezifische E-Lizenzen eignet sich dieses Modell vor allem für Testzwecke und den Einstieg in die Nutzung.

Das Pay-As-You-Go-Modell ermöglicht es Unternehmen, flexibel basierend auf ihrer tatsächlichen Nutzung zu zahlen, was besonders bei variablen Workloads von Vorteil ist. Alle erforderlichen Lizenzen sind im Dienstpreis enthalten, sodass keine separaten E-Lizenzen erworben werden müssen. Dieses Modell bietet eine kosteneffiziente Lösung, die sich optimal an den individuellen Bedarf anpasst und gleichzeitig Skalierbarkeit gewährleistet.

Das Enterprise Agreement richtet sich an große Organisationen, die eine langfristige Zusammenarbeit mit erweiterten Supportoptionen und attraktiven Rabatten anstreben. Neben den grundlegenden Leistungen können je nach Unternehmensanforderungen E3/E5-Lizenzen hinzugefügt werden, um ein umfassendes Lizenzmanagement zu realisieren. Dieses Modell bietet somit eine robuste und individuell anpassbare Lösung für Unternehmen mit umfangreichen IT-Anforderungen.

### **Ressourcen Gruppen**

Ressourcen Gruppen in Azure sind logische Container, in denen zusammengehörige Ressourcen wie virtuelle Maschinen, Datenbanken oder Netzwerke gebündelt werden. Sie vereinfachen die Verwaltung, indem sie zentrale Steuerung von Berechtigungen, Richtlinien und Abrechnungen ermöglichen. Außerdem bietet diese Struktur eine bessere Übersicht, erleichtert das Skalieren und ermöglicht das gemeinsame Verwalten oder Löschen von Ressourcen, was

---

<sup>3</sup> Perplexity Search (bearbeitet): Erkläre die Azure Subscriptions Free, PAYG und EA; Zugriff am 05.03.2025

den administrativen Aufwand reduziert. Für das Projekt werden folgende Ressourcen Gruppen benötigt:

- Server
- Datenbank MYSQL
- Datenbank Pinecone
- Datenbank Blob
- Azure OpenAI Instanz
- Netzwerk Sicherheit Gruppe (NSG)

### **Netzwerk Sicherheit Gruppe (NSG)**

Diese Ressource fungiert als Firewall für den Server und die Datenbanken und übernimmt somit eine zentrale Rolle beim Schutz vor unerwünschtem Datenverkehr. Über ein intuitives Konfigurationsmenü können gezielt IP-Adressen definiert werden, die autorisiert sind, auf den Server zuzugreifen. Dabei ist es von entscheidender Bedeutung, die Zugriffskontrollregeln so restriktiv wie möglich zu gestalten, um potenzielle Sicherheitslücken zu minimieren. Es wird empfohlen, bei jeder Regelkonfiguration sowohl die Quell- als auch die Zieladresse präzise anzugeben – ergänzt durch den entsprechenden Port und das verwendete Protokoll –, um eine feingranulare Steuerung des Datenverkehrs zu gewährleisten. Auf diese Weise wird sichergestellt, dass nur legitim autorisierte Verbindungen zugelassen werden und die Sicherheit der gesamten Infrastruktur erhöht wird.

### **Virtuelle Maschine (VM)**

Um das Backend und Frontend hosten zu können wird ein Server benötigt. In der entsprechenden Ressourcen Gruppe wird eine virtuelle Maschine mit dem Betriebssystem Ubuntu Server 22.04 aufgesetzt.

Spezifikation (Entwicklung):

- 2 Cores burstable CPU
- 8 GB RAM
- 30GB SDD

Der Zusatz „burstable“ bei der CPU sorgt dafür, dass die Maschine unter Lastspitzen kurzfristig hochskaliert. Die Kosten werden mit Credits gedeckt, die in Zeiten geringer Last angesammelt werden können.

Die VM ist für „Azure Patch Management“ konfiguriert und erhält somit automatisch Updates. Wenn automatische Updates gewählt werden, ist darauf zu achten die „Autoreboot“ Funktion zu deaktivieren, um ungeplante Downtimes zu vermeiden.

Während der Konfiguration wird auch der Admin Benutzer erstellt. Der Zugriff auf den Server erfolgt über SSH auf Port 22 mit einem SSH-Schlüssel. Root Login sowie Authentifizierung mittels Passworts wurden deaktiviert. In den Netzwerk Einstellungen des Konfiguration Wizards

wird die richtige NSG ausgewählt und ein Subnetz gewählt damit der Server eine private IP-Adresse erhält. Des Weiteren wird eine öffentliche IP-Adresse generiert und ein DNS-Name vergeben, um die Maschine später komfortabel über eine URL zu erreichen. Die öffentliche IP-Adresse wird auch benötigt, um ein gültiges Zertifikat zu erhalten und somit HTTPS zu nutzen, um eine sichere Kommunikation zu ermöglichen.

### **Datenbanken**

Azure bietet ein komfortables Setup-Interface, mit dem sich Datenbanken einfach und effizient im Browser konfigurieren lassen.

Für den Aufbau der MySQL-Datenbank wird in der entsprechenden Ressourcengruppe ein MySQL-Datenbankserver erstellt. Nach dem Start des Servers stellt Azure automatisch alle erforderlichen Verbindungsinformationen – wie Serveradresse, Port, Benutzername und Authentifizierungsdaten – bereit, sodass eine nahtlose programmatische Kommunikation ermöglicht wird. Das weitere Setup, einschließlich der Erstellung von Benutzerkonten, Datenbanken und Tabellen, erfolgt automatisiert im Backend, was zu einer konsistenten und zuverlässigen Datenbankinfrastruktur führt.

Parallel dazu wird eine Pinecone-Vektor-Datenbank in einer separaten Ressourcengruppe eingerichtet, um vektorbasierte Abfragen und Analysen zu unterstützen. Die Einrichtung dieser Datenbank erfordert zusätzlich eine Verknüpfung mit einem Pinecone-Account, über den die notwendigen API-Schlüssel und Zugangstoken generiert werden. Über das Pinecone-Webinterface können zudem spezifische Benutzerkonten für Entwicklerinnen und Entwickler angelegt werden, wodurch eine präzise und sichere Steuerung des Zugriffs gewährleistet wird. Auch das Anlegen der Indizes kann über das Web Interface vorgenommen werden.

Ergänzend zu den Datenbanken wird ein Blob Storage eingerichtet, der als zentraler Speicher für unstrukturierte Daten wie Bilder und Dokumente dient. Nach Abschluss der Konfiguration stellt Azure einen Verbindungsstring bereit, der alle relevanten Parameter enthält, um eine effiziente Kommunikation mit dem Blob Storage zu ermöglichen. Diese Konfiguration erlaubt es, Bilder sowohl aus dem Frontend als auch vom Server über eindeutig definierte Links und Zugangsdaten sicher auszutauschen.



### 4.2.2. Server

Nachfolgend werden die Schritte der Serverkonfiguration beschrieben. Der Server wurde über eine SSH Verbindung konfiguriert und administriert.

#### Initiales Server Setup

Im ersten Schritt werden die Systemupdates installiert, wobei alle nachfolgenden Aktualisierungen von Azure Update Management [45] übernommen werden. Anschließend erfolgt die Installation der benötigten Tools und Software, darunter htop als Ressourcenmonitor, iftop als Netzwerkmonitor, tree für die Anzeige von Ordnerstrukturen, git für die Versionskontrolle sowie die jeweils korrekten Versionen von Python, Python-venv und NodeJS. Nach der erfolgreichen Installation dieser Komponenten werden die Verzeichnisse für das FastAPI-Backend unter `~/fastapi`, für das React-Frontend unter `/var/www/react-app` und für die statische HTML-Seite unter `/var/www/regeln` angelegt.

#### NGINX

NGINX ist ein leistungsstarker Webserver, der unter anderem als Reverse Proxy eingesetzt wird, um den Zugriff auf unterschiedliche Dienste wie API und das Frontend zu koordinieren. Ein Reverse Proxy bietet Vorteile wie verbesserte Sicherheit, Lastverteilung, Caching und die zentrale Verwaltung von SSL-Zertifikaten, wodurch sämtliche Anfragen über Port 443 gesichert verarbeitet werden können. Port 80 wird nicht zugelassen bzw auf 443 umgeleitet. In unserem Szenario wird NGINX so konfiguriert, dass es eingehende HTTPS-Anfragen aus dem Internet entgegennimmt und anhand definierter Routen, beispielsweise `/api` für die API, `/regeln` für die Administrative Seite und `/reactapp` für das Frontend, an die entsprechenden Dienste auf der lokalen Maschine weiterleitet. Dies vereinfacht nicht nur die Infrastrukturverwaltung, sondern optimiert auch die Performance und Sicherheit der gesamten Anwendung.

#### Certbot

Certbot ist ein Open-Source-Tool, das automatisiert SSL-Zertifikate von „Let's Encrypt“ bezieht und verwaltet. In Kombination mit NGINX übernimmt Certbot die automatische Erneuerung und Installation der Zertifikate, sodass die HTTPS-Verbindungen stets abgesichert und auf dem neuesten Stand sind. Dies reduziert den administrativen Aufwand erheblich und minimiert das Risiko von Zertifikatsausfällen, da manuelle Eingriffe entfallen. Durch die Integration in die NGINX-Konfiguration wird das System bei jeder Zertifikatserneuerung automatisch neu geladen, was eine kontinuierliche und reibungslose Verschlüsselung der Datenübertragung gewährleistet. Für das erstmalige Erlangen eines Zertifikates müssen die entsprechenden Firewall Regeln gesetzt werden damit die Zertifikatsserver eine Verbindung aufbauen können.

### 4.2.3. Bereitstellung des FastAPI Backends

Es wird kurz die Vorgehensweise erläutert und auf die Punkte Ausfallsicherheit und Skalierbarkeit eingegangen.

## **GIT**

Das Backend, welches aus einer FastAPI App und mehreren Subrepositories besteht, wird mittels GIT von Azure DevOps auf den Server geklont. Anschließend wird die virtuelle Umgebung für Python erstellt, die Abhängigkeiten installiert und die Umgebungsvariablen gesetzt.

## **Ausfallsicherheit<sup>4</sup>**

Die App wird im „systemd“ registriert um eine Autostart und Autorestart Funktionalität zu erhalten.

„Systemd“ ist der zentrale Dienstmanager in vielen Linux-Distributionen, der dafür sorgt, dass Systemdienste und Anwendungen beim Bootvorgang automatisch gestartet werden. Es überwacht laufende Prozesse und startet sie bei Bedarf automatisch neu, wodurch eine hohe Verfügbarkeit und Stabilität gewährleistet wird. In unserem Fall stellt „systemd“ sicher, dass die API stets im Autostart läuft und bei einem Ausfall automatisch neu gestartet wird.

## **Skalierbarkeit**

Die FastAPI-Anwendung wird als Hintergrundprozess mit Uvicorn gestartet, in diesem Fall unter Verwendung des Prozessmanagers systemd, um eine dauerhafte Ausführung sicherzustellen. Uvicorn fungiert als moderner ASGI-Server, der als Brücke zwischen NGINX und der FastAPI-App dient, indem er eingehende HTTP/HTTPS-Anfragen effizient entgegennimmt und asynchron verarbeitet. NGINX übernimmt dabei die Rolle des Reverse-Proxy, der SSL/TLS-Termination auf Port 443 durchführt und so den sicheren Datenverkehr gewährleistet, bevor die Anfragen an Uvicorn weitergeleitet werden. Durch diese Architektur kann die Anwendung unter hoher Last dynamisch skaliert werden, wobei Uvicorn dafür sorgt, dass Ressourcen optimal genutzt werden und die Reaktionszeiten auch bei steigenden Benutzerzahlen kurz bleiben.

### **4.2.4. Bereitstellung des React Frontends**

Das React Frontend wird von DevOps über ein Git-Repository in einen dedizierten Ordner geklont, um sicherzustellen, dass stets die aktuelle Version des Codes vorliegt. Anschließend werden alle erforderlichen Abhängigkeiten, beispielsweise über den Paketmanager npm, installiert, wodurch sichergestellt wird, dass sämtliche Module und Bibliotheken korrekt integriert werden. Anschließend erfolgt der Build-Prozess, bei dem der Quellcode optimiert, minifiziert und für den produktiven Einsatz vorbereitet wird. Nach dem Start der Anwendung ist das Frontend unter der konfigurierten Domain und dem Pfad /reactapp erreichbar. Dadurch wird eine nahtlose und sichere Kommunikation mit dem Backend ermöglicht, was eine konsistente und skalierbare Benutzererfahrung unterstützt.

---

<sup>4</sup> GPT-o1: Erkläre bitte kurz „systemd“ in Linux System. Zugriff 04.03.2025

#### 4.2.5. Bereitstellung der Regel Eingabeseite<sup>5</sup>

Die HTML-basierte Administrationsseite wird in einen bereitgestellten Ordner geklont, wodurch eine klare Trennung von anderen Webinhalten und eine bessere Verwaltung der statischen Dateien ermöglicht wird. Anschließend wird die Seite durch die vorher konfigurierte NGINX-Instanz bereitgestellt, wobei ein Serverblock eingerichtet wird, der Anfragen an den definierten Pfad (z. B. /regeln) annimmt und an den entsprechenden Ordner weiterleitet. Die NGINX-Konfiguration beinhaltet darüber hinaus Einstellungen für SSL/TLS, um eine sichere HTTPS-Verbindung zu gewährleisten, sowie Routing-Regeln, die den Datenverkehr effizient verwalten und bei Bedarf Weiterleitungen vornehmen. Dadurch wird sichergestellt, dass die Seite technisch robust, sicher und performant unter der angegebenen Domain erreichbar ist.

### 4.3. Implementierung

Im Folgenden wird erläutert, wie die Backend-API implementiert wurde. Diese API fungiert als zentrales Repository, in das sämtliche weitere Komponenten als Subrepositories integriert werden, was eine klare modulare Struktur und eine effiziente Verwaltung der Codebasis ermöglicht. Dieser Ansatz erleichtert nicht nur die Erweiterbarkeit des Systems, sondern unterstützt auch eine konsistente Zusammenarbeit zwischen den verschiedenen Entwicklerinnen und Entwicklern. Anschließend wird der Aufbau einer speziellen Klasse für die Bildanalyse beschrieben, die mit KI Bildanalyse Fähigkeiten ausgestattet ist, um Bilddaten präzise zu verarbeiten und in aussagekräftige Berichte umzuwandeln. Diese Klasse spielt eine entscheidende Rolle im Gesamtprozess, indem sie die automatisierte Erkennung und Bewertung von Schadensfällen ermöglicht.

#### 4.3.1. API<sup>6</sup>

Eine API ermöglicht es unterschiedlichen Systemen miteinander zu kommunizieren. Sie stellt die notwendigen Funktionen und Daten zur Verfügung. Dadurch wird die Integration und Skalierbarkeit von Anwendungen, wie beispielsweise der Verbindung zwischen Frontend und Backend, erheblich erleichtert.

Die 3 populärsten Python Frameworks sind [46]:

- Flask
- FastAPI
- Django

Diese Frameworks bieten je nach Anforderungen unterschiedliche Vorteile.

Flask besticht durch seine Leichtigkeit und Flexibilität, was es ideal für schnelle Prototypen und kleinere bis mittelgroße Projekte macht. Da Flask nur ein minimales Grundgerüst bereitstellt, haben Entwicklerinnen und Entwickler die Freiheit, genau die Bibliotheken und

---

<sup>5</sup> GPT-o1: Verbessere die Textstruktur und verwende eine technischere Sprache. Zugriff: 04.03.2025

<sup>6</sup> Perplexity Search (bearbeitet): Erkläre die API Frameworks Flask, Django und FastAPI; Zugriff: 05.03.2025

Tools auszuwählen, die sie benötigen – was jedoch auch dazu führen kann, dass die Projektstruktur uneinheitlich und individuell stark variiert, da viele Funktionen erst hinzugefügt werden müssen.

Django hingegen ist ein umfassendes All-in-one-Framework, das mit zahlreichen eingebauten Features wie einem leistungsfähigen ORM, integrierter Authentifizierung und einem benutzerfreundlichen Admin-Interface bereits viele gängige Anforderungen abdeckt. Diese Voreinstellungen erleichtern den Aufbau komplexer, skalierbarer und sicherer Webanwendungen, allerdings ist die Lernkurve aufgrund des großen Funktionsumfangs etwas steiler und die Performance kann in einigen Fällen hinter schlankere Lösungen zurückbleiben.

FastAPI kombiniert moderne Konzepte wie asynchrone Programmierung mit einer hervorragenden Performance und bietet dabei die automatische Erstellung von interaktiver API-Dokumentation sowie Unterstützung für Type-Anmerkungen, was die Lesbarkeit und Wartbarkeit des Codes erheblich verbessert. Diese Eigenschaften machen FastAPI besonders attraktiv für Projekte, die schnelle Entwicklungszyklen, Prototyping oder den Aufbau von Microservices erfordern, auch wenn es als relativ neues Framework teilweise noch von einer wachsenden Community begleitet wird.

Auf Grund der Performance und der automatischen API-Dokumentation haben wir uns für FastAPI entschieden.

### Sicherheit

CORS (Cross-Origin Resource Sharing) ist ein Sicherheitsmechanismus, der es Webanwendungen ermöglicht, Ressourcen von einer anderen Domain anzufordern, während der Server gleichzeitig kontrolliert, welche Ursprünge Zugriff erhalten. Durch spezielle HTTP-Header legt der Server fest, welche Domains berechtigt sind, auf seine Daten zuzugreifen, was sowohl Flexibilität als auch Schutz vor unerlaubten Zugriffen bietet. Diese Regelung ist besonders wichtig, wenn Frontend- und Backend-Anwendungen auf unterschiedlichen Domains gehostet werden und dennoch eine reibungslose Kommunikation gewährleistet werden soll.

Implementiert wird diese Funktionalität über das von FastAPI zur Verfügung gestellte Modul „CORSMiddleware“. [47]

Um die API vor unbefugten Zugriffen zu schützen, wird neben der Netzwerk Sicherheit Gruppe auch ein API Key verwendet, um sich gegenüber der API zu authentifizieren (Listing 4.1 API-Key Validierung). Der Key wird als Umgebungsvariable hinterlegt und beim Starten geladen.

```
async def verify_api_key(request: Request):
    api_key = request.headers.get("X-API-Key")
    if api_key != API_KEY:
        raise HTTPException(status_code=403, detail="Unauthorized")
```

Listing 4.1 API-Key Validierung

Die verschiedenen Endpunkte prüfen zuerst den übermittelten API Key. Der Key wird im Header als „X-API-Key“ übertrage und aus der übergebenen Request ausgelesen. Danach wird der Key mit dem in der Umgebungsvariable gespeicherten Key verglichen. Wenn dieser valide ist, wird der Aufruf des Endpunktes zugelassen. Beispielhaft wird hier der Status Endpunkt gezeigt.

```
@app.get("/api_status", dependencies=[Depends(verify_api_key)])
async def root():
    try:
        return {"msg": "API is running"}
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

Listing 4.2 Status Endpunkt mit API Key

### 4.3.2. KI gestützte Bildanalyse

Die Bildanalyse mittels KI ermöglicht eine automatische Erkennung und Bewertung von Schäden an Objekten. Durch den Einsatz von GPT-4o werden visuelle Merkmale präzise analysiert, um die Art und den Umfang der Beschädigung zu bestimmen. Anhand der gewonnenen Daten generiert die KI einen detaillierten Bericht, der betroffene Bereiche, Schadensart und dokumentiert. Diese Berichte dienen als wichtige Informationsgrundlage für weitere Prozesse. Die Klasse für die Bildanalyse (**Fehler! Verweisquelle konnte nicht gefunden werden.** stellt 5 Funktionen bereit.

Benötigt wird das Python SDK von AzureOpenAI.

- pip install openai

**Klassendiagramm:**

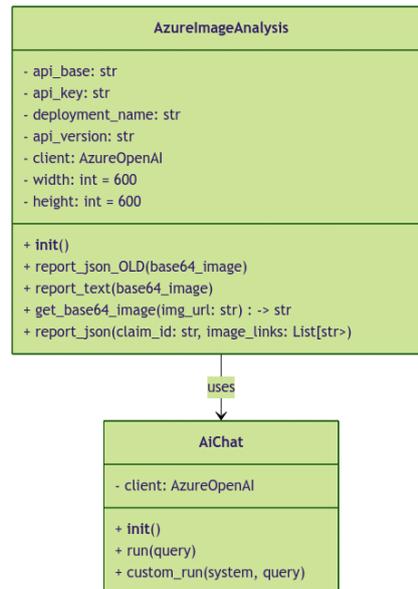


Abbildung 4.1 Klassendiagramm Bildanalyse

### Laden der Bilder

Die benötigten Bilder befinden sich in einem Blob Storage in der Azure Cloud. Die Funktion zum Laden erwartet ein Array mit Links. Die Bilder werden nach Erhalt des Arrays mittels der Bibliothek „requests“ aus dem Blob geladen.

Die Bibliothek requests ermöglicht eine einfache Durchführung von HTTP-Anfragen. Sie unterstützt verschiedene Methoden wie GET, POST, PUT und DELETE und vereinfacht so die Interaktion mit Web-APIs erheblich. Dank ihrer benutzerfreundlichen API und guten Fehlerbehandlung hat sich requests als eine der beliebtesten Bibliotheken für den Zugriff auf Webressourcen etabliert.

### Base64 Codierung

Damit OpenAI die Bilder verarbeiten kann müssen diese nach Base64 konvertiert werden. Die Base64-Codierung wandelt Binärdaten von Bildern in einen String um, sodass diese problemlos über textbasierte APIs übertragen werden können. Da OpenAI Bildanalysen über Endpunkte realisiert sind, die ausschließlich Text erwarten, ist die Base64-Codierung notwendig, um die Bilddaten korrekt und fehlerfrei zu übermitteln. Letztlich vereinfacht diese Methode die Integration und Verarbeitung von Bilddaten in KI-gestützten Analyseprozessen. Die Konvertierung der Bilder wurde mit der Bibliothek „Pillow“ und der Image Klasse (Listing 4.3) realisiert.

Pillow ist ein Fork der Python Imaging Library (PIL) und bietet viele Funktionen zur Bildverarbeitung in Python. Mit Pillow können Bilder bearbeitet, skaliert, gefiltert und in verschiedenen Formaten gespeichert werden. Dank der benutzerfreundlichen API und der breiten Unterstützung zahlreicher Bildformate ist Pillow ein unverzichtbares Werkzeug für die Bildmanipulation in Python-Anwendungen.

```
image = Image.open(BytesIO(response.content))
image = image.convert("RGB")
image.thumbnail((self.width, self.height))

buffer = BytesIO()
image.save(buffer, format="JPEG")
base64_str = base64.b64encode(buffer.getvalue()).decode("utf-8")
```

### Listing 4.3 Base64 Codierung

## OpenAI Client

Im Konstruktor (Listing 4.4) wird ein Client Objekt erzeugt, welches von verschiedenen Funktionen genutzt wird. Azure OpenAI benötigt folgende Parameter:

- URL des Azure OpenAI Endpunkts
- API KEY
- Den Namen der bereitgestellten Instanz
- Die Versionsnummer der API

Weiters wird im Konstruktor eine maximale Bildgröße von 600 x 600 Pixel festgelegt. Dies verringert die zu übertragende Datenmenge. Werte unter 500 x 500 sollten vermieden werden, da es zu bemerkbarer Verringerung der Präzision führt.

```
class AzureImageAnalysis:
    def __init__(self):
        self.api_base = os.getenv("AZURE_OPENAI_ENDPOINT")
        self.api_key = os.getenv("AZURE_OPENAI_API_KEY")
        self.deployment_name = os.getenv("AZURE_OPENAI_CHAT_DEPLOYMENT_NAME")
        self.api_version = os.getenv("AZURE_OPENAI_API_VERSION")
        self.client = AzureOpenAI(
            api_key=self.api_key,
            api_version=self.api_version,
            base_url=f"{self.api_base}openai/deployments/{self.deployment_name}",
        )

        self.width = 600
        self.height = self.width
        self.system_prompt = "You are an image expert in an insurance company."
        self.prompt = (
            "This is one of many pictures of an object in an insurance case. Describe the damaged "
            "object in detail. Determine the repair cost and the likelihood of a successful repair."
            "Also try to guess how expensive the repair will be. If you can, identify the brand "
            "and model. If the image is advertisement or not a damaged object ignore it "
            "and respond just with IMAGE_IGNORED"
        )
    )
```

Listing 4.4 Klasse Bildanalyse Konstruktor

## Bildanalyse

Der Endpunkt unterscheidet sich nur durch die übergebenen Messages von dem normalen Chat Endpunkt, wie in Listing 4.5 zu sehen ist. Zusätzlich wird hier bei der User Message ein Array mit 2 Listen übergeben. Die erste enthält den Prompt, die zweite enthält die Bild URL mit dem Base64 codierten Bild. Der Parameter „max\_tokens“ beschränkt die Ausgabe auf 800 Token. Die Analyse wird aus der erhaltenen Antwort extrahiert und an den Funktionsaufrufer zurückgegeben.

```
def report_text(self, base64_image):
    response = self.client.chat.completions.create(
        model=self.deployment_name,
        messages=[
            {"role": "system", "content": self.system_prompt},
            {"role": "user", "content": [
                {
                    "type": "text",
                    "text": self.prompt
                },
                {
                    "type": "image_url",
                    "image_url": {
                        "url": f"data:image/jpeg;base64,{base64_image}"
                    }
                }
            ]
        ],
        max_tokens=800
    )
    tokens = response.usage.total_tokens
    print(f"Total tokens used: {tokens}")
    data = response.choices[0].message.content
    return data
```

Listing 4.5 OpenAI SDK mit Bild URL

Mithilfe eines ThreadPoolExecutors werden alle Bilder gleichzeitig verarbeitet, wodurch die Ausführung beschleunigt wird, wie in Listing 4.6 dargestellt. Tritt während der Verarbeitung ein Fehler auf, beispielsweise aufgrund von Rate Limiting, wartet das Programm automatisch 10 Sekunden, bevor es den Vorgang wiederholt. Die erhaltenen Resultate werden zu einem String konkateniert.

Da die Schadensfälle oft Bilder enthalten, die nichts mit dem eigentlichen Schaden zu tun haben, wurde der KI mittels Prompt Engineering mitgeteilt, dass Bilder, die keine beschädigten Objekte zeigen ignoriert werden sollten.

*Prompt (Ausschnitt): [... If the image is advertisement or not a damaged object ignore it and respond just with IMAGE\_IGNORED ...]*

```
def process_image(args):
    index, img_url = args
    print(f"Processing image {index} {img_url}")

    base64_image = self.get_base64_image(img_url)

    while True:
        try:
            report = self.report_text(base64_image)
            break
        except Exception as ex:
            sleep_time = 10
            print(f"Rate Limit [wait {sleep_time}]: {ex}")
            time.sleep(sleep_time)

    if report.__contains__("IMAGE_IGNORED"):
        print("IMAGE_IGNORED (not relevant)")
        # report = "IMAGE_IGNORED"
    return report

with concurrent.futures.ThreadPoolExecutor() as executor:
    results = list(executor.map(process_image, enumerate(image_links)))
```

#### Listing 4.6 Parallele Bildanalyse

##### Report generieren

Der String der nun alle Reports enthält wird final noch einmal von der KI verarbeitet. Die Reports werden in einer abschließenden Schadensanalyse zusammengefasst und als Markdown Report und als JSON Objekt bereitgestellt.

Um die Ausgabe als JSON-Objekt zu erzwingen wird zuvor ein JSON-Template angelegt und mit dem Prompt übergeben. Das Modell befüllt nun die Values im JSON-Objekt basierend auf den Informationen aus dem Report. (Listing 4.7)

```
summary_json_prompt = (
    "Fill the JSON template with the given Information."
    "If you cant provide Information leave it blank."
    "Start with { and do not use ``json at the beginning and `` at the end of the output."
    f"Template: {report_template}"
)
```

#### Listing 4.7 Prompt JSON Template

Beide Reportarten werden anschließend in einem JSON-Objekt zusammengefasst und in der Datenbank abgelegt. Da Modelle dazu tendieren immer anzugeben in welcher Sprache die Ausgabe erfolgt (Python, Java, JSON) wird mittels Prompt Engineering explizit darauf hingewiesen auf solche Ausgaben zu verzichten.

**Sequenzdiagramm:**

Abbildung 4.2 beschreibt den Ablauf der Bildanalyse für einen Schadensfall mit mehreren Bildern.

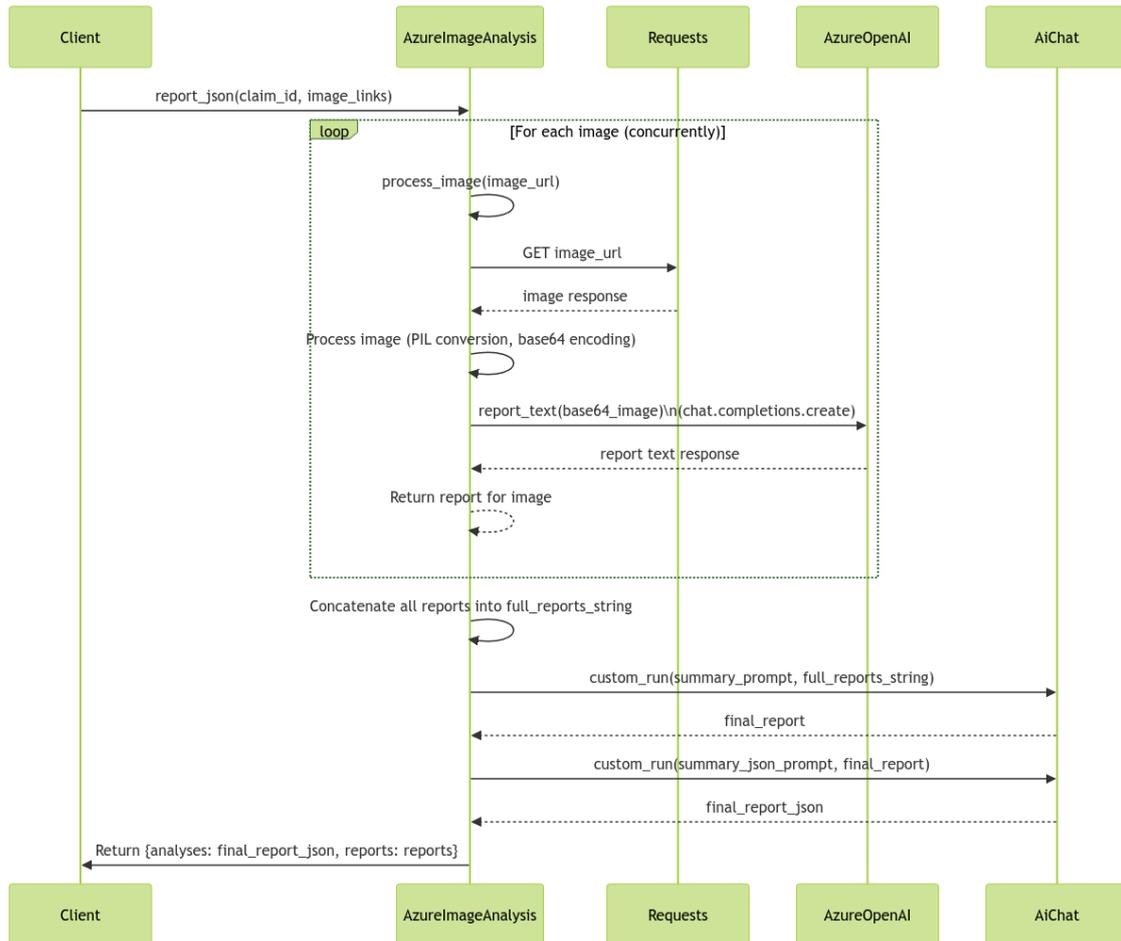


Abbildung 4.2 Sequenzdiagramm Bildanalyse

### 4.3.3. Beispiel

Als Beispiel wurde ein zufälliges Bild aus dem Internet gewählt. Der Prompt gibt dem Modell Anhaltspunkte, was es mit dem Bild machen sollte. Der Report wird als in Markdown formatierter String zurückgegeben



Abbildung 4.3 Beispielbild [48]

#### Prompt:

[This is one of many pictures of an object in an insurance case. Describe the damaged object in detail. Determine the repair cost and the likelihood of a successful repair. Also try to guess how expensive the repair will be. If you can, identify the brand and model. If the image is advertisement or not a damaged object ignore it and respond just with IMAGE\_IGNORED]

#### Report:

The image depicts a heavily damaged BMW vehicle, possibly a 1 Series (E87 or E81 model). The damage is concentrated on the front end, affecting multiple components:

#### Damage Description:

##### Hood:

- Heavily crumpled and bent, with a large dent in the middle.
- The metal has folded upwards, suggesting a significant frontal impact.

##### Front Bumper:

- Torn and detached in several areas, exposing internal components.
- The license plate is bent and partially detached.

**Headlights:**

- The left headlight (driver's side) appears shattered or damaged.
- The right headlight is cracked and misaligned.

**Grille:**

- Both kidney grilles are damaged or missing pieces.

**Radiator Support:**

- Likely bent or broken due to the impact.

**Fenders:**

- The front fenders seem slightly bent near the headlights.

**Structural Damage:**

- Possible damage to the frame or crumple zones, though not entirely visible.

**Estimated Repair Cost:**

- Minor Repairs: If the damage is only cosmetic and does not involve the engine, chassis, or airbags, repair costs could be around €3,000 - €5,000.
- Moderate Repairs: If structural or radiator support damage is present, costs could rise to €5,000 - €8,000.
- Severe Repairs: If the frame is bent or engine damage is involved, the cost could exceed €10,000, making the vehicle a possible total loss depending on its market value.

**Likelihood of Successful Repair:**

- If no major structural or mechanical damage exists, repairs are feasible, though expensive.
- If the frame is bent, repairs are possible but may compromise vehicle safety and resale value.
- If airbags deployed or engine damage occurred, insurance may consider it a total loss.

#### 4.3.4. Verbesserungen

Nachfolgend Hinweise und Anregungen für den Live-Betrieb.

**Hardware**

Da es sich aktuell um ein Entwicklungs-Setup handelt ist es notwendig für den Live Betrieb Anpassungen der Server Hardware entsprechend der Auslastung vorzunehmen.

**Azure Lizenz**

Um Wartezeiten zu verkürzen ist es ratsam für den produktiven Live-Betrieb auf ein höheres Subscription Modell in Azure zu wechseln. Dadurch erhält man die Möglichkeit mehrere Instanzen von OpenAI bereitzustellen, um einen noch höheren Parallelisierungsgrad zu erreichen. Dadurch wird die User Experience gesteigert und verringert die Wartezeiten des Endbenutzers.

Durch höhere Lizenzen werden auch die Tokenlimits erhöht. Mit der aktuellen S0 Lizenz sind 30.000 Tokens/Minute möglich. Zum Vergleich liegen die Limits bei der E5 Lizenz bei 150.000 Tokens/Minute.

### Implementierungen

OpenAI bietet mit der AssistantAPI eine neue Art mit KI-Modellen zu interagieren. Da sich diese API noch in der BETA-Phase befindet wurde in diesem Projekt darauf verzichtet.

Es zeigt sich aber, dass das momentane „Frage und Antwort“ Verfahren an seine Grenzen stößt da immer sehr viel Kontext im Prompt übergeben werden muss. Mit der AssistantAPI umgeht man dieses Manko und kann Nachrichten direkt auf einen „Thread“ legen der anschließend von einem, vorab konfigurierten, Assistenten abgearbeitet werden kann. Der Vorteil besteht nun darin, dass der User mehrere Aufgaben auf den „Thread“ legen kann und der Assistent diese abarbeiten kann. Sind alle Aufgaben erledigt meldet er sich zurück. Mittels der Funktion „FunctionCalling“ können so mehrere Agenten autonom an einem Projekt arbeiten.

Da in Zukunft deutlich stärkere Modell verfügbar sein werden ist es notwendig die alten durch diese zu ersetzen. Mit dem Erscheinen der Modelle o1 und o3, welche durch Nachdenkprozesse hohe kognitive Leistungen erbringen, können Aufgaben, die keine Multimodalen Fähigkeiten benötigen an diese Modelle ausgelagert werden.

Wann von OpenAI ein neues und stärkeres Multimodales Modell erscheint und GPT-4o abgelöst wird ist derzeit noch nicht bekannt.

## 5. Implementierung eines Frontend für Sachbearbeiter/innen zur Darstellung und Analyse der Ergebnisse. (Gordon Telsnig)

Im Folgenden wird das Frontend mit React TypeScript implementiert. Die Anwendungsfälle wurden analysiert und die Bedürfnisse von Management und Endanwendern berücksichtigt. Die Umsetzung ist skalierbar und benutzerfreundlich.

### 5.1. Aufgabenstellung

Für die Entwicklung eines User-Interfaces und die Auswahl eines geeigneten Frontend-Frameworks müssen im Vorfeld umfangreiche Informationen über den Anwendungsfall erhoben werden. Hierbei können verschiedene Abstraktionsebenen eine Rolle spielen. So unterscheiden sich oft die Sichtweisen des Managements von denen der Mitarbeiterinnen und Mitarbeiter, die das Interface täglich nutzen. Für die Entwicklung des Frontends waren die folgenden zwei Punkte wesentlich:

- Berücksichtigung der unterschiedlichen Anforderungen von Management und Endanwendern.
- Auswahl eines Frontend-Frameworks, das sowohl Skalierbarkeit als auch Benutzerfreundlichkeit gewährleistet.

### 5.2. Evaluierung der darzustellenden Daten und Anwendungsfälle für KI-Systeme

Das Projektteam sowie der Vertreter des beteiligten Unternehmens, Ing. Harald Klary, Bakk. MSc. von der KLV hatten die Aufgabe, Anwendungsfälle für KI-basierte Systeme zu identifizieren. In mehreren Teammeetings wurden Themen wie die Automatisierung von Geschäftsprozessen, das Auslesen von Exif-Daten, die Schadenanalyse durch KI-Systeme sowie die Kostenkalkulation als realisierbare Ansätze diskutiert und bewertet.

### 5.3. Bewertung der Sachbearbeiterinnen und Sachbearbeiter Anforderungen

Um die Auswahl des geeigneten Frontends unter Berücksichtigung technischer Aspekte weiter zu validieren, war es entscheidend, die Sachbearbeiterinnen und Sachbearbeiter der KLV im Arbeitsalltag mindestens einen Tag lang zu begleiten und Interviews durchzuführen. Ziel war es, Eindrücke über die bereits genutzten Softwareanwendungen zu gewinnen. Ebenso war es unerlässlich, das Tagesgeschäft der Sachbearbeiterinnen und Sachbearbeiter im Detail zu verstehen, um die Anforderungen an das geplante User-Interface besser erfassen zu können. Ein weiterer wichtiger Aspekt bestand darin, die bestehenden Probleme und Wünsche der Sachbearbeiterinnen und Sachbearbeiter aufzugreifen, um diese gezielt in die Entwicklung der Applikation einfließen zu lassen.

## 5.4. Zusammenfassung der Anforderungen

Die folgenden Vorgaben haben sich im Rahmen der Aufgabenstellung ergeben:

- Microsoft Edge Browser
- Interne Designrichtlinien
- Intuitive Menüführung
- Vermeidung exotischer Frameworks
- Wartbarer und skalierbarer Code
- Reibungslose API Integration

## 5.5. Überblick und Architektur

Das Flussdiagramm (siehe Abbildung 5.1) zeigt eine Gesamtübersicht des Nutzerflusses von der Eingabe der Schadennummer im WelcomeScreen bis zur Darstellung der Daten in den verschiedenen Sections.

Die Architektur der Anwendung folgt einem modularen Ansatz, bei dem jede wesentliche Funktionalität in eigenständige Bereiche (Sections) unterteilt wird. Jede Section wird von wiederverwendbaren Komponenten getragen, die in übergeordneten Container-Komponenten zusammengefasst sind. Dieser strukturierte Aufbau trennt die Verantwortlichkeiten klar und erleichtert die Wiederverwendung sowie Erweiterung einzelner Komponenten – was den Entwicklungsaufwand reduziert und die Wartbarkeit verbessert.

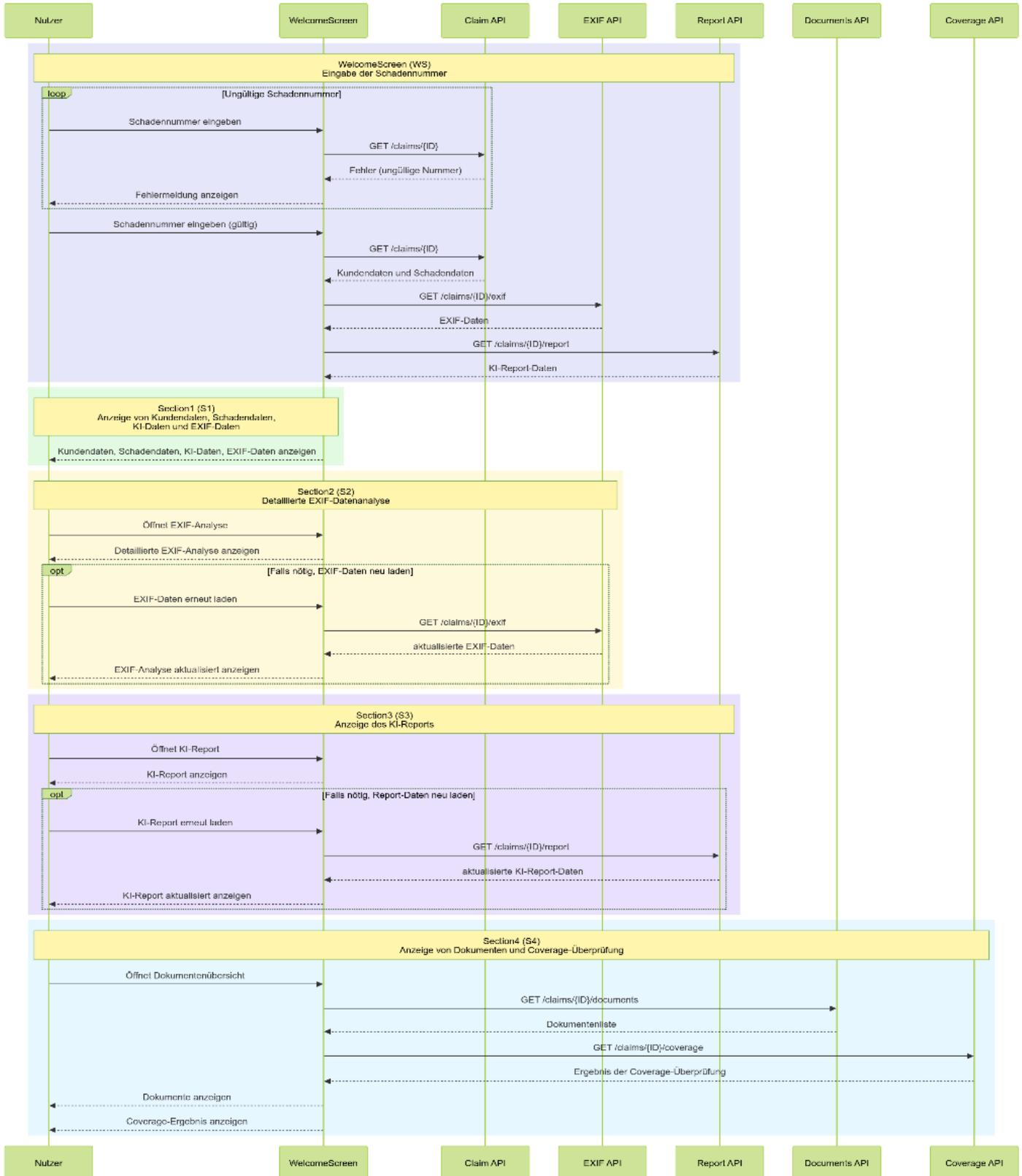


Abbildung 5.1 Flussdiagramm-Frontend

### 5.5.1. Welcomescreen

Der Willkommensbildschirm ist das Erste, was die Nutzerin oder der Nutzer sieht. Neben dem Logo in Abbildung 5.2 kann eine Schadennummer eingegeben werden. Eine Regex-Prüfung überprüft die Gültigkeit dieser Eingabe. Bei erfolgreicher Prüfung werden die Daten für Section1 im Hintergrund geladen.



Abbildung 5.2 Welcomescreen

### 5.5.2. Section 1 – Übersicht

In diesem Bereich werden Container dynamisch bereitgestellt (siehe Abbildung 5.3). Auf diese Weise werden Kundendaten, Versicherungsinformationen, Schadensdaten, Bilddaten und ein KI-Bericht in Listenform angezeigt. Auf der linken Seite werden in separaten Containern die Kundendaten und Schadensinformationen bereitgestellt. Die Navigationsleiste liefert wesentliche Versicherungsdetails. Ein schneller Überblick wird dadurch ermöglicht. Große Datensätze werden in Listen dargestellt. Informationen wie Namen, Adressen, Schadennummern und Vertragsdetails liegen klar strukturiert vor.

Ein zentraler Bildcontainer wird eingesetzt. Ein interaktiver ImageSlider unterstützt die Betrachtung der Bilddaten. Eine Fullscreen-Option kann aktiviert werden, um einzelne Bilder in einer vergrößerten Detailansicht zu prüfen. Die zugehörigen EXIF-Daten werden in Echtzeit aktualisiert. Das passende Set an Kamera- und Aufnahmedaten wird jeweils angezeigt. Eine integrierte Tooltip-Funktion erläutert jede Spalte. Neue Benutzerin oder Benutzer erhalten durch Mouse-Hover zusätzliche Informationen, die den Inhalt der Datenzellen erklären.

Die Container können bei Bedarf geschlossen werden. Der freigewordene Platz wird von den verbleibenden Elementen genutzt. Diese flexible Container-Struktur ermöglicht eine dynamische Anpassung der Arbeitsfläche. Kundendaten oder Schadensinformationen lassen sich minimieren, wenn der Fokus auf den Bilddaten oder einer detaillierten Dokumentenübersicht liegt.

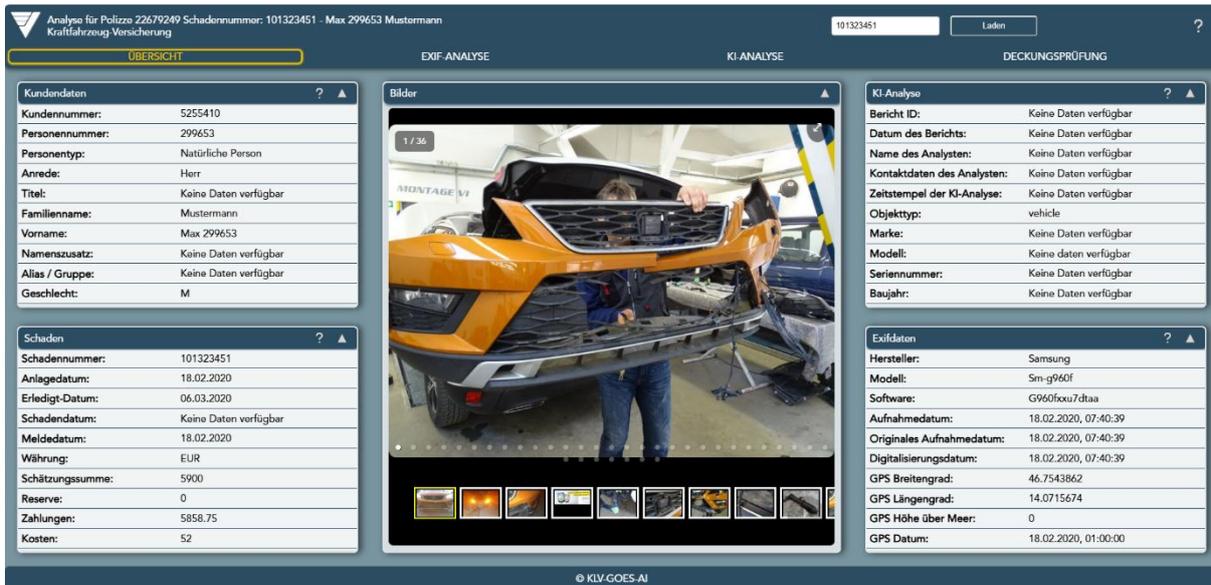


Abbildung 5.3 Section1-Übersicht

### 5.5.3. Section 2 – Detaillierte EXIF-Analyse

In Section2 werden alle EXIF-Daten tabellarisch präsentiert (siehe Abbildung 5.4). Checkboxes ermöglichen die individuelle Auswahl der angezeigten Informationen wie Aufnahmedatum, Kameramodell oder GPS-Daten. Eine Hilfsklasse sorgt für dynamische Formatierung und farbliche Kennzeichnung fehlender oder abweichender Werte. Datumsangaben werden präzise verglichen, um Unstimmigkeiten wie ein falsches Schadensdatum schnell zu erkennen. Dadurch wird eine genauere Analyse ermöglicht. Eine Sortierfunktion erleichtert das Auffinden spezifischer Daten. Beim Mouse-Hover über einen Tabellenheader erscheint eine vergrößerte Vorschau, was eine komfortable Detailprüfung unterstützt.

The screenshot shows the 'EXIF-ANALYSE' section of the software. It features a table of detailed EXIF data for various images. The table has columns for Bildname, Software, GPSDatum, Digitalisierungsdatum, Aufnahmedatum, Originales Aufnahmedatum, Blitz, and Blendenzahl. The data is color-coded: green for correct values and red for missing or incorrect values.

Bildname	Software	GPSDatum	Digitalisierungsdatum	Aufnahmedatum	Originales Aufnahmedatum	Blitz	Blendenzahl
20200217_182555.jpg.jpg	G960FXXU7DTAA	17.02.2020, 17:25:42	2020-02-17 18:25:55	2020-02-17 18:25:55	2020-02-17 18:25:55	true	1.5
20200217_182555.jpg.jpg	G960FXXU7DTAA	17.02.2020, 17:25:42	2020-02-17 18:25:55	2020-02-17 18:25:55	2020-02-17 18:25:55	true	1.5
20200217_182605.jpg.jpg	G960FXXU7DTAA	17.02.2020, 17:26:01	2020-02-17 18:26:05	2020-02-17 18:26:05	2020-02-17 18:26:05	true	1.5
20200217_182605.jpg.jpg	G960FXXU7DTAA	17.02.2020, 17:26:01	2020-02-17 18:26:05	2020-02-17 18:26:05	2020-02-17 18:26:05	true	1.5
20200218_074039.jpg.jpg	G960FXXU7DTAA	18.02.2020, 06:40:31	2020-02-18 07:40:39	2020-02-18 07:40:39	2020-02-18 07:40:39	true	2.4
20200218_074039.jpg.jpg	G960FXXU7DTAA	18.02.2020, 06:40:31	2020-02-18 07:40:39	2020-02-18 07:40:39	2020-02-18 07:40:39	true	2.4
image002.jpg.jpg	Keine Daten verfügbar	false	Keine Daten verfügbar				
00_FAHRZEUG_BILD_017.JPG.JPG	Keine Daten verfügbar	false	Keine Daten verfügbar				
00_FAHRZEUG_BILD_026.JPG.JPG	Keine Daten verfügbar	false	Keine Daten verfügbar				
00_FAHRZEUG_BILD_027.JPG.JPG	Keine Daten verfügbar	false	Keine Daten verfügbar				
00_FAHRZEUG_BILD_028.JPG.JPG	Keine Daten verfügbar	false	Keine Daten verfügbar				
00_FAHRZEUG_BILD_029.JPG.JPG	Keine Daten verfügbar	false	Keine Daten verfügbar				
00_FAHRZEUG_BILD_001.JPG.JPG	Keine Daten verfügbar	false	Keine Daten verfügbar				

Abbildung 5.4 Section2-Exifanalyse

#### 5.5.4. Auslesen von Exifdaten:

Ein Bild wird mittels Pillow geöffnet und dessen EXIF-Daten werden ausgelesen. Werden keine Daten gefunden, gibt die Anwendung ein JSON mit einem Fehlerstatus zurück (Listing 5.1 JSON-EXifdaten). Sind EXIF-Daten vorhanden, überführt das Programm diese in ein Dictionary. Standard-Tags (z. B. Make, Model, DateTime) werden mithilfe der Bezeichnungen aus ExifTags zugeordnet und durch die Funktion `sanitize_value` bereinigt. Anschließend wird das Dictionary in Teilbereiche strukturiert:

- **camera:** enthält Angaben zu Hersteller, Modell und verwendeter Software
- **timestamps:** fasst Datum und Uhrzeiten (Aufnahme, Original und Digitalisierungszeit) zusammen
- **gps:** ruft via `process_gps_data` die GPS-Informationen ab, wandelt Koordinaten mithilfe von `get_decimal_from_dms` in Dezimalwerte um und formatiert Zeit und Datum
- **photo:** extrahiert und konvertiert foto-spezifische Daten wie Brennweite, Blendenzahl, ISO, Belichtungszeit, Flash, Weißabgleich, Messmodus, sowie Bildabmessungen (über `process_photo_data`)

Das resultierende JSON-Objekt (Status 1) wird so aufgebaut, dass es dem Frontend als strukturierte Datenbasis dient – idealerweise zur weiteren Verarbeitung oder Speicherung in der Datenbank.

```
{
  "Error": null,
  "status": 1,
  "camera": {
    "make": "NIKON",
    "model": "COOLPIX P6000",
    "software": "Nikon Transfer 1.1 W"
  },
  "timestamps": {
    "datetime": "2008:11:01 21:15:08",
    "datetime_original": "2008:10:22 16:38:20",
    "datetime_digitized": "2008:10:22 16:38:20"
  },
  "gps": {
    "latitude": 43.467081666663894,
    "longitude": 11.884538333330555,
    "altitude": null,
    "datum": "WGS-84  ",
    "timestamp": "14.0:36.0:47.23",
    "date": "2008:10:23"
  },
  "photo": {
    "focal_length": 16.6,
    "f_number": 4.7,
    "iso_speed": 64,
    "exposure_time": 0.01044932,
    "exposure_bias": 0.0,
    "flash": false,
    "white_balance": "auto",
    "metering_mode": "pattern",
    "scene_type": "\u0001",
    "contrast": 0,
    "saturation": 0,
    "sharpness": 0,
    "lens_model": null,
    "width": 640,
    "height": 480
  }
}
```

Listing 5.1 JSON-EXifdaten

### 5.5.5. Section 3 – KIReport und Dokumentenanzeige

In Section3 wird ein interaktiver Dokumenten-Viewer mit einem Container für KI-Reports kombiniert (siehe Abbildung 5.5). Dadurch stehen sämtliche Schadensinformationen gebündelt an einer Stelle zur Verfügung. PDF-Dokumente, wie Schadenmeldungen, Rechnungen oder Gutachten, werden in Listenform präsentiert. KI-basierte Analysen werden in einer Markdown-Struktur angezeigt. Flexible Layout-Optionen ermöglichen ein müheloses Umschalten zwischen Bildern und textbasierten Reports. Ein Bilderslider erlaubt eine Fullscreen-Ansicht der Bilder, während LoadButtons ein gezieltes Wechseln zwischen verschiedenen Dokumenttypen ermöglichen.

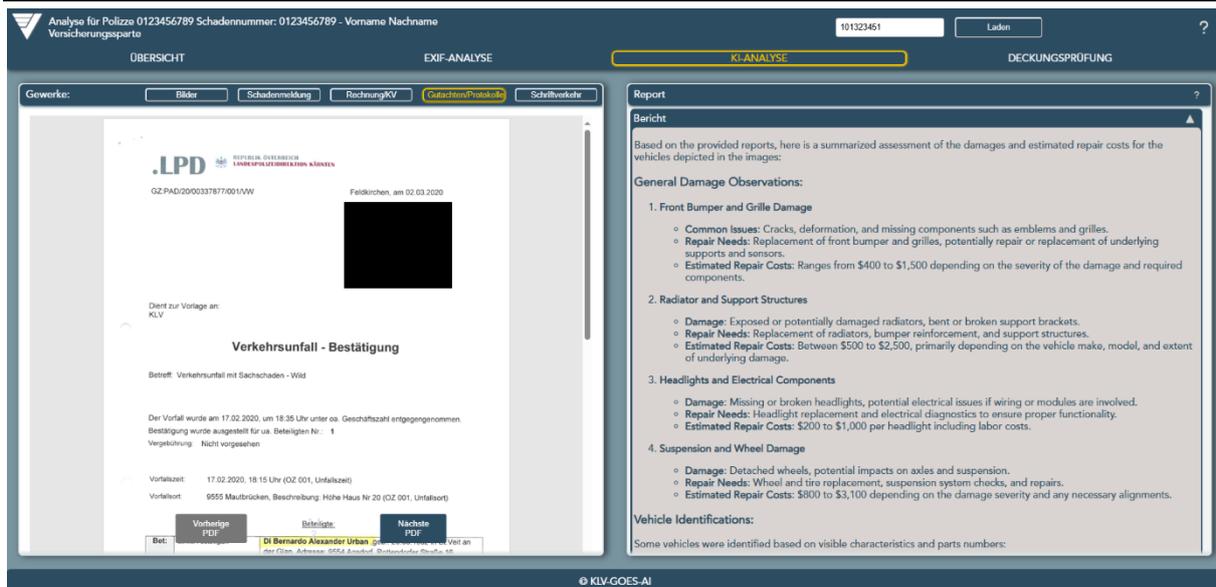


Abbildung 5.5 Section3-KiAnalyse

### 5.5.6. Section 4 – Deckungsprüfung

In Section4(siehe Abbildung 5.6) werden Dokumente aus Section3 genutzt, die zuvor von einem KI-Agenten zusammengefasst wurden. Per Checkbox kann ausgewählt werden, welche Zusammenfassungen in die finale Deckungsprüfung einfließen. Anschließend wird die automatisierte Versicherungsprüfung per Button-Klick gestartet. Die Ergebnisse werden im SummaryContainer übersichtlich dargestellt. Die Container-Struktur erlaubt das Minimieren oder Öffnen einzelner Bereiche, um eine aufgeräumte Arbeitsfläche zu erhalten. Ein integrierter Chat führt einen vollständigen Nachrichtenverlauf und ermöglicht eine direkte Kommunikation zum Schadensfall. Ein automatischer Scroll-Mechanismus rückt dabei stets die neuesten Nachrichten ins Blickfeld.

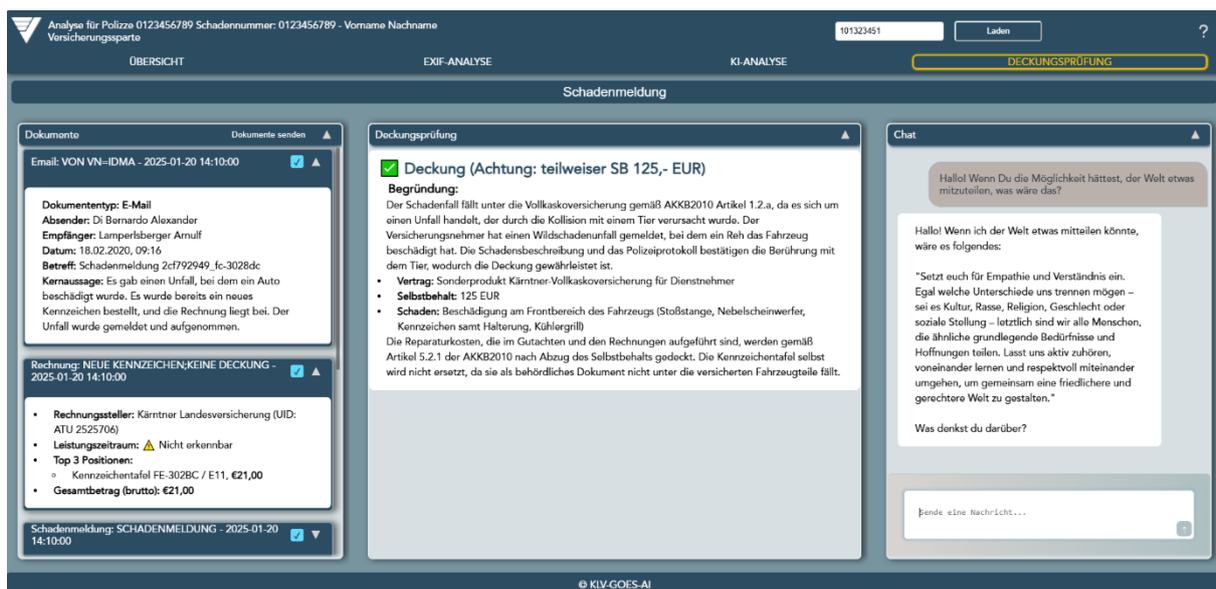


Abbildung 5.6 Section4-Deckungsprüfung

## 5.6. Routing und Zugriffsschutz

Die Komponente `ProtectedRoute` (Listing 5.3) fungiert als Zugangsschutz für geschützte Bereiche. Sie prüft über den globalen State, ob eine gültige Schadennummer (Listing 5.2) (`claimID`) vorhanden ist. Fehlt diese, wird mittels des `Navigate`-Elements aus `react-router` automatisch die Benutzerin oder der Benutzer auf den `WelcomeScreen` umgeleitet. Diese Validierung verhindert, dass unautorisierte Nutzerinnen oder Nutzer auf sensible Inhalte zugreifen.

Im App-Routing wird `react-router` genutzt, um die Navigation zwischen den verschiedenen Bereichen (Section1 bis Section4) dynamisch zu steuern. Alle Routen, die sensible Inhalte enthalten, werden in `ProtectedRoute` eingebettet. Dadurch wird sichergestellt, dass der Zugriff auf den Hauptinhalt nur dann erfolgt, wenn die Validierung in `ProtectedRoute` erfolgreich war. Gleichzeitig ermöglicht die Verwendung von Routes ein deklaratives Routing, bei dem der Zusammenhang zwischen URL und angezeigtem Layout direkt im JSX-Code festgelegt wird. Dies führt zu einem modularen, wartungsfreundlichen Routing-System, das sowohl die Benutzerführung optimiert als auch den Zugriffsschutz sicherstellt. [49]

### Validierung

Im Listing 5.2 wird die Funktion `validateClaimID` definiert. Sie prüft anhand dieses regulären Ausdrucks, ob die eingegebene Schadennummer genau acht Ziffern hat. Nur wenn diese Prüfung erfolgreich ist, gilt die `claimID` als gültig.

```
const validateClaimID = (id: string): boolean => {  
  return /^[0-9]{8,20}$/ .test(id);  
};
```

Listing 5.2 Validierung

## Geschützter Zugriff via ProtectedRoute

Die Komponente ProtectedRoute kontrolliert den Zugriff auf sensible Bereiche. Sie verwendet die zuvor validierte claimID aus dem Global State und leitet den Nutzer andernfalls automatisch auf die Willkommenseite um. Dadurch wird sichergestellt, dass nur autorisierte Benutzerinnen oder Benutzer den Hauptinhalt sehen.

```
// src/Components/Route/ProtectedRoute.tsx
import React from 'react';
import { Navigate } from 'react-router-dom';
import { useGlobalState } from '../Context/GlobalStateContext';

interface ProtectedRouteProps {
  children: React.ReactElement;
}

const ProtectedRoute: React.FC<ProtectedRouteProps> = ({ children }) => {
  const { claimID } = useGlobalState();

  if (!claimID) {
    return <Navigate to="/" replace />;
  }

  return children;
};

export default ProtectedRoute;^
```

Listing 5.3 Beispiel ProtectedRoute

## Routing mit React Router

Über die Routes und Route-Elemente (Listing 5.4) werden die verschiedenen Bereiche (z. B. /section1) definiert. Indem sensible Inhalte in ProtectedRoute-Elemente eingebettet werden, garantiert das Routing, dass nur bei gültiger claimID Zugriff gewährt wird. Das Ergebnis ist ein deklaratives und wartungsfreundliches Navigationskonzept.

```
<Routes>
  <Route
    path="/section1"
    element={
      <ProtectedRoute>
        <MainContent />
      </ProtectedRoute>
    }
  />
  { /* Weitere Routen folgen in ähnlicher Weise */ }
</Routes>
```

Listing 5.4 Beispiel Routes

## 5.7. Globale Zustände und Context

React Context ist ein zentrales Konzept in React, das es ermöglicht, Daten über die gesamte Komponenten-Hierarchie hinweg bereitzustellen, ohne dass diese explizit von Eltern- an Kind-Komponenten „weitergereicht“ werden müssen. Anstatt Daten bei jeder Ebenenübergabe als Props zu übergeben – ein Vorgehen, das häufig als „Prop Drilling“ bezeichnet wird und mit zunehmender Komponentenverschachtelung schnell unübersichtlich werden kann – stellt der Context eine Art globalen Speicher bereit, in dem relevante Zustände zentral abgelegt werden. [50]

### Was bedeutet das konkret für unser Projekt?

In unserem Projekt nutzen wir einen globalen Context (GlobalStateContext) und mehrere spezialisierte Data Provider, um unterschiedliche Datensätze zu verwalten. Dadurch gilt:

#### Zentralisierte Datenhaltung

Der GlobalStateContext speichert Schlüsselwerte wie die Schadennummer (ClaimID), den aktuell aktiven Navigationsbereich (Section), API-Status und weitere Flags. Alle Komponenten, unabhängig von ihrer Tiefe in der Komponentenstruktur, können direkt auf diese zentralen Daten zugreifen. Dies vereinfacht nicht nur die Datenverwaltung, sondern verhindert auch Fehler, die durch das mehrfache Weiterreichen von Props entstehen könnten.

#### Modularität und Trennung der Zuständigkeiten

Neben dem globalen Context werden verschiedene Data Provider in eigenen Kontexten organisiert, um unterschiedliche Datenarten getrennt zu verwalten. Zum Beispiel:

- **DataProviderClaim:** Kümmt sich um alle relevanten Claim-Daten.
- **DataProviderImageAndExif:** Stellt Bild- und EXIF-Daten bereit.
- **DataProviderChat:** Verarbeitet Chat-Nachrichten.
- **DataProviderKI:** Bezieht KI-Reports und Analysen.
- **DocumentsProvider:** Organisiert Dokumente aus verschiedenen Quellen.
- **CoverageVerificationProvider:** Führt die Deckungsprüfung durch.

Diese Aufteilung erlaubt es, die einzelnen Datendomänen isoliert zu behandeln. Dadurch wird der Code wartbarer und Erweiterungen lassen sich einfacher umsetzen, weil Änderungen in einem Kontext nicht direkt andere Bereiche beeinflussen.

#### Effizienz und Reduktion von Redundanzen

Indem Daten zentral im Context gespeichert werden, müssen sie nicht mehrfach oder redundant via API-Aufrufen in unterschiedlichen Komponenten angefragt werden. Alle Komponenten greifen auf dieselbe Quelle zu, was nicht nur den Code vereinfacht, sondern auch eine konsistente Datenbasis gewährleistet. Dies führt zu einer besseren Performance und reduziert die Gefahr von Synchronisationsproblemen.

## Verbessertes Fehlermanagement

Mit einem zentralen State-Management-System können Fehler oder Inkonsistenzen leichter identifiziert und behandelt werden, da der gesamte Zustand an einem Ort gepflegt wird. Dies erleichtert auch die Implementierung von globalen Fehlergrenzen (Error Boundaries) und anderen Sicherheitsmechanismen.

### 5.8. React Hooks [51]

React Hooks sind Funktionen, die es ermöglichen, in funktionalen Komponenten Zustände und Nebenwirkungen (Side Effects) zu verwalten – Funktionen, für die man früher Klassenkomponenten benötigte. Die wichtigsten Vorteile von Hooks sind:

**Einfachheit und Lesbarkeit:** Hooks erlauben einen deklarativen Programmierstil. Anstatt komplexe Klassen mit `this.state` und `this.setState` zu verwenden, können wir Zustände direkt in der Funktion definieren.

**Wiederverwendbarkeit:** Mit benutzerdefinierten Hooks (Custom Hooks) können wiederkehrende Logiken in isolierten Funktionen gekapselt und in verschiedenen Komponenten wiederverwendet werden.

**Vermeidung von Prop Drilling:** Durch den Einsatz von React Context in Kombination mit Hooks lassen sich globale Zustände zentral verwalten und in beliebigen Komponenten nutzen, ohne diese durch mehrere Ebenen als Props weiterreichen zu müssen.

**Modularität:** Hooks fördern eine klare Trennung zwischen Logik und Darstellung. So wird die Komplexität in einzelnen Komponenten reduziert und der Code bleibt wartungsfreundlich.

**Grundprinzip:** Ein Hook ist im Grunde eine JavaScript-Funktion, die intern einen Zustand oder andere React-Funktionalitäten (z. B. Nebenwirkungen) verwaltet.

#### 5.8.1. useEffect() Hook

Dieses Codebeispiel zeigt, wie der `useEffect`-Hook (Listing 5.5) die Bild- und EXIF-Daten nur bei Wechsel zu „section2“ und gültiger `claimID` lädt, solange der entsprechende API-Aufruf noch nicht ausgeführt wurde. Nach erfolgreichem Abruf wird der Status auf „image“ gesetzt, um Mehrfachanfragen zu verhindern und Performance zu optimieren.

#### Funktion des `useEffect`-Hooks:

**Syntax:** `useEffect` nimmt zwei Argumente entgegen: eine Effektfunktion und ein Array von Abhängigkeiten.

**Effektfunktion:** Diese Funktion wird nach jedem Render ausgeführt, sofern sich mindestens eine der im Abhängigkeits-Array angegebenen Variablen geändert hat.

**Abhängigkeiten:** Durch das Übergeben eines Arrays (hier: `[currentSection, claimID, apiCallsTriggered.image, markApiCallTriggered, fetchImageAndExifData]`) wird sichergestellt, dass der Effekt nur dann erneut ausgeführt wird, wenn eine dieser Variablen einen neuen Wert erhält.

```
useEffect(() => {
  if (currentSection === "section2" && claimID && !apiCallsTriggered.image) {
    fetchImageAndExifData(claimID).catch((err: any) =>
      console.error("Fehler im /imageandexif-Call in Section2:", err)
    );
    markApiCallTriggered("image");
  }
}, [
  currentSection,
  claimID,
  apiCallsTriggered.image,
  markApiCallTriggered,
  fetchImageAndExifData,
]);
```

Listing 5.5 Beispiel useEffect();

### 5.8.2. useState() Hook [51]

Diese Code-Passage zeigt, wie der useState-Hook den Sichtbarkeitsstatus mehrerer Container in einem einzigen Objekt verwaltet. In Kombination mit useCallback entsteht eine Funktion (toggleContainer), die gezielt den Zustand einzelner Container steuert und unnötige Re-Render vermeidet.

```
const [containers, setContainers] = useState<ContainersState>(initialState);

const toggleContainer = useCallback(
  (id: keyof ContainersState, forceOpen: boolean | null = null) => {
    setContainers(prev => ({
      ...prev,
      [id]: forceOpen !== null ? forceOpen : !prev[id]
    }));
  },
  []
);
```

Listing 5.6 useStateHook();

#### Funktion des useState-Hooks?

Der useState-Hook ermöglicht die lokale Verwaltung von Zuständen in Funktionskomponenten. Er gibt ein Array mit dem aktuellen Zustand und einer Aktualisierungsfunktion zurück.

#### Syntax

Der Aufruf useState(initialState) liefert zwei Elemente:

- **containers**: Der aktuelle Zustand
- **setContainers**: Die Funktion, um diesen Zustand anzupassen

Die Funktion `toggleContainer` nutzt `useCallback`, um den Zustand eines bestimmten Containers zu ändern. Hierbei wird entweder ein vorgegebener Wert (`forceOpen`) übernommen oder der aktuelle Zustand invertiert.

### Zustandsverwaltung

`useState` speichert im Beispiel den Sichtbarkeitsstatus mehrerer Container in einem Objekt. `toggleContainer` verwaltet den jeweiligen Container, indem ein direkter Wert gesetzt oder der bisherige Status umgekehrt wird.

### Vorteile

- **Einfachheit:** Lokale Zustände in einer Komponente ohne komplexes State-Management.
- **Lesbarkeit:** Alle Änderungen sind in der Funktion `toggleContainer` gekapselt.

### 5.8.3. `useReducer()` Hook [52]

Der `useReducer`-Hook ist eine Alternative zu `useState`, die insbesondere bei komplexen Zustandslogiken mit mehreren abhängigen Werten oder wiederkehrenden Statusänderungen vorteilhaft ist. Die folgende Abbildung zeigt den Ablauf der Statusaktualisierung durch den `useReducer`-Hook anhand eines Tooltipsystems.

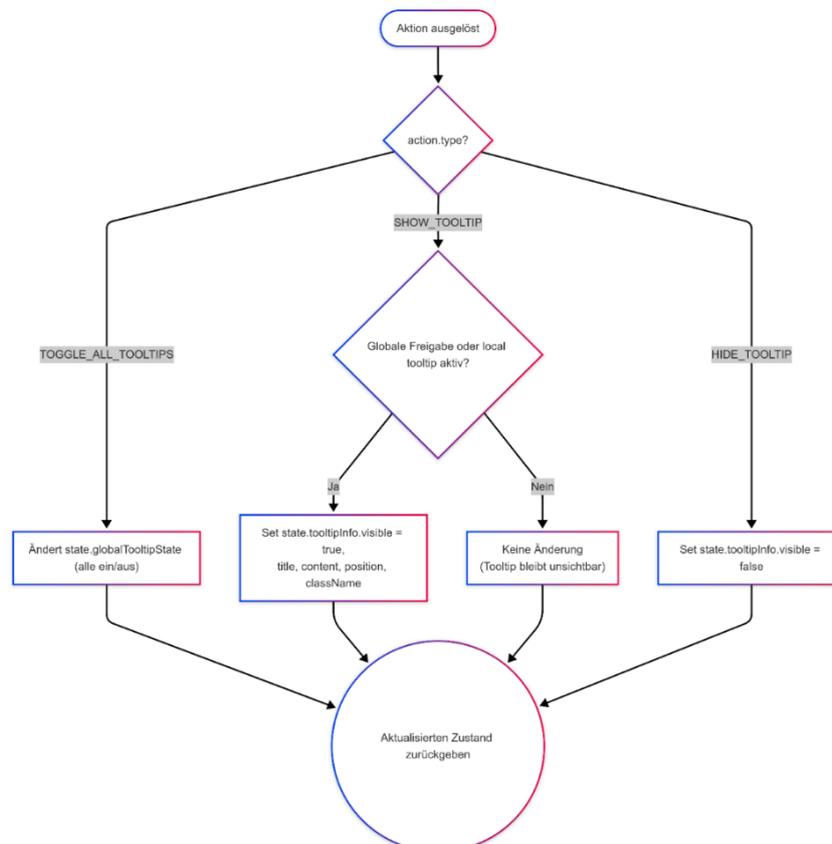


Abbildung 5.7 `useReducerHook()`;

## Funktionsweise des useReducer-Hooks<sup>7</sup>

### Syntax

Der Hook `useReducer(reducer, initialState)` nimmt zwei Argumente entgegen:

- **Reducer-Funktion:** Eine Funktion (hier `tooltipReducer`), die basierend auf dem aktuellen state und einer action den neuen Zustand zurückgibt.
- **Initialzustand:** Ein Startwert (hier `tooltipInitialState`), der die Ausgangswerte für den State definiert.

### Rückgabewerte

Der Hook gibt ein Tupel aus zwei Werten zurück:

- **state:** Das aktuelle Zustandsobjekt (z. B. `state.tooltipInfo.visible`).
- **dispatch:** Eine Funktion, die eine Aktion (z. B. `{ type: 'SHOW_TOOLTIP', payload: {...} }`) an den Reducer übergibt.

### Ablauf der State-Aktualisierung mit useReducer

#### 1. Action Dispatch

Wird `dispatch` aufgerufen, überprüft der Reducer den Typ der Aktion (z. B. `TOGGLE_ALL_TOOLTIPS`).

#### 2. State-Aktualisierung

Abhängig vom `action.type` wird der Zustand angepasst (z. B. Umschalten der globalen Tooltip-Einstellung).

#### 3. Re-Render der Komponenten

Falls der Reducer einen neuen Zustand zurückgibt, führt React ein Re-Rendering der betroffenen Komponenten durch.

### Beispiel: `toggleAllTooltips()`

1. Der Funktionsaufruf `dispatch({ type: 'TOGGLE_ALL_TOOLTIPS' })` wird ausgelöst.
2. Der Reducer erkennt den Aktionstyp `TOGGLE_ALL_TOOLTIPS` und invertiert `state.globalTooltipState` (z. B. von `false` auf `true` oder umgekehrt).
3. Die Tooltips werden daraufhin global aktiviert oder deaktiviert.

---

<sup>7</sup> ChatGPT (GPT-4), Datum der Abfrage: 03.03.2025, Prompt: „Überprüfe den useReducer-Hook im Vergleich mit der offiziellen React-Dokumentation. Link dazu: <https://react.dev/reference/react/useReducer>. Ist useReducer richtig beschrieben? Finde alle Fehler! Mach einen Vorschlag für bessere Struktur und Wording.“

### 5.8.4. React.Memo [53]

React.memo ist ein Higher-Order Component (HOC), das es ermöglicht, funktionale Komponenten zu „memoisieren“. Dies bedeutet, dass React die Props der Komponente beim Rendervorgang vergleicht und, wenn diese unverändert bleiben, den Re-Render der Komponente unterbindet. Dadurch werden unnötige Renderzyklen verhindert und die Performance verbessert.

#### Memoisierung

Beim Rendern einer Komponente, die mit React.memo umhüllt ist, werden die aktuellen Props mit den vorherigen Props verglichen. Ändern sich diese nicht, wird die Komponente aus dem virtuellen DOM wiederverwendet, ohne dass die Funktion erneut ausgeführt wird.

#### Parent-Child-Szenario

Normalerweise führt ein Re-Render der Parent-Komponente dazu, dass alle Child-Komponenten ebenfalls neu gerendert werden – auch wenn deren Props gleich bleiben. Mit React.memo wird dies verhindert.

#### Beispiel

Ein Container-Komponente enthält eine ScrollAbleDataList. Wird der Container neu gerendert, aber die Props an die ScrollAbleDataList bleiben identisch, so wird diese Child-Komponente nicht neu gerendert. Dadurch wird unnötige Arbeit vermieden.

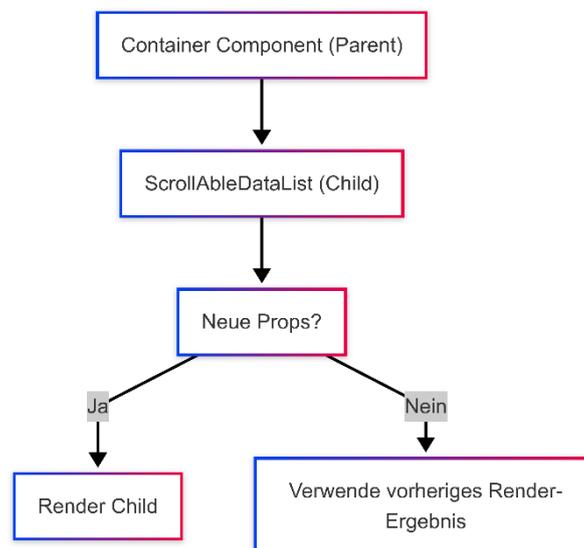


Abbildung 5.8 React.Memo

#### Initialer Render

Der Container (Parent) rendert die Child-Komponente (ScrollAbleDataList) mit bestimmten Props. Beim ersten Render wird die Child-Komponente natürlich gerendert.

## Re-Render

Wenn der Parent erneut rendert, prüft React.memo in der Child-Komponente, ob die übergebenen Props sich geändert haben.

- Falls **nein** ("Nein" Pfad): Wird das vorherige Render-Ergebnis beibehalten (Child wird nicht neu gerendert).
- Falls **ja** ("Ja" Pfad): Wird die Child-Komponente neu gerendert, da sich die Props geändert haben.

## Stabilität

Damit React.memo den gewünschten Effekt hat, müssen alle übergebenen Props stabil sein. Das bedeutet, dass Callback-Funktionen mittels useCallback und komplexe Objekte mittels useMemo memoisiert werden sollten, sodass sie bei einem Parent-Re-Render nicht als neu erzeugt gelten.

## Performance

React.memo umschließt eine funktionale Komponente und vergleicht beim Rendern die übergebenen Props. Bleiben die Props unverändert, führt React.memo keinen erneuten Render aus. So werden unnötige Re-Renders reduziert, was gerade in großen Anwendungen mit komplexen Komponentenstrukturen wichtig ist.

## Sauberer Code

Komponenten bleiben in ihrer Logik unverändert, während die Optimierung auf der Ebene der Re-Render-Entscheidungen erfolgt.

## Wiederverwendbarkeit

HOC-Muster ermöglicht es, die Optimierungslogik an vielen Stellen der Anwendung einheitlich anzuwenden.

## 5.9. TypeScript in React-Anwendungen [54]

TypeScript ist in modernen React-Projekten mittlerweile Standard. Durch die statische Typisierung bietet TypeScript mehrere Vorteile:

### Früherkennung von Fehlern

Bereits während der Entwicklung werden Typfehler erkannt, bevor der Code im Browser ausgeführt wird.

### Verbesserte Code-Vervollständigung und Dokumentation

Intellisense in modernen Editoren ermöglicht eine präzisere Autovervollständigung und erleichtert das Verständnis der API und der Komponenten.

### Erhöhte Wartbarkeit

Durch explizite Typen und Interfaces wird die Codebasis besser dokumentiert. Dies erleichtert die Zusammenarbeit in Teams und die zukünftige Erweiterung der Anwendung.

## Integration in React

TypeScript ergänzt React-Komponenten, indem es die Props, den State und die Rückgabewerte klar definiert. Das Ergebnis ist ein sicherer und besser wartbarer Code.

## 5.10. Erreichte Funktionalitäten

In diesem Abschnitt werden die umgesetzten Funktionen in vier Bereichen vorgestellt. Section 1 zeigt eine dynamische Datendarstellung, Section 2 befasst sich mit der EXIF-Datenanalyse, Section 3 kombiniert KI-Funktionen mit einem Dokumentenmanagement, und Section 4 rundet das System mit einem integrierten Chat und flexiblen Container-Optionen ab.

### 5.10.1. Section 1 – Übersicht und Datendarstellung

**Dynamische Container:** Container werden dynamisch bereitgestellt, um verschiedene Datenbereiche (Kundendaten, Versicherungsinformationen, Schadensdaten, Bilddaten und einen KI-Bericht) in übersichtlicher Listenform anzuzeigen.

**Detaillierte Präsentation:** Auf der linken Seite werden Kundendaten und Schadensinformationen in eigenständigen Containern detailliert präsentiert – die Navigationsleiste liefert bereits die wesentlichen Versicherungsdetails, sodass der Sachbearbeiterinnen und Sachbearbeiter einen schnellen Überblick erhält.

**Bildbetrachtung:** Ein zentraler Bildcontainer ermöglicht über einen interaktiven ImageSlider die Betrachtung der Bilddaten. Eine Fullscreen-Option erlaubt die Prüfung einzelner Bilder in vergrößerter Detailansicht, während die zugehörigen EXIF-Daten in Echtzeit aktualisiert werden.

**Tooltip-Unterstützung:** Eine integrierte Tooltip-Funktion erläutert jede Spalte, unterstützt insbesondere neue Benutzerinnen oder Benutzer und sorgt so für eine intuitive Orientierung.

**Flexible Container-Verwaltung:** Container können bei Bedarf geschlossen werden, sodass freigebliebener Platz optimal von den übrigen Elementen genutzt wird.

### 5.10.2. Section 2– Detaillierte EXIF-Datenanalyse

**Tabellarische Darstellung:** Alle EXIF-Daten werden in einer übersichtlichen, tabellarischen Ansicht präsentiert, was eine strukturierte Übersicht aller Bildattribute ermöglicht.

**Individuelle Filterung:** Über integrierte Checkboxen kann die Nutzerin oder der Nutzer gezielt bestimmen, welche Informationen – wie Aufnahmedatum, Kameramodell oder GPS-Daten – eingeblendet werden sollen.

**Dynamische Formatierung:** Eine speziell entwickelte Hilfsklasse sorgt für eine dynamische Formatierung und farbliche Kennzeichnung fehlender oder abweichender Werte.

**Präziser Datumsvergleich:** Datumsangaben werden miteinander verglichen, um Unstimmigkeiten – etwa, ob das Schadensdatum vor oder nach dem Bildaufnahmedatum liegt – sofort sichtbar zu machen.

**Schnelle Sortierung & Vorschau:** Eine Sortierfunktion ermöglicht die schnelle Auffindbarkeit bestimmter Daten, und beim Mouse-Hover über einen Tabellenheader erscheint eine vergrößerte Vorschau, die eine komfortable Detailprüfung erlaubt.

### 5.10.3. Section 3 – KI-Analyse und Dokumentenmanagement

**Interaktiver Dokumenten-Viewer:** Section3 kombiniert einen interaktiven Dokumenten-Viewer mit einem separaten Container für KI-Reports, sodass sämtliche Schadensinformationen an einer Stelle verfügbar sind.

**Übersichtliche Listen:** Sowohl PDF-Dokumente (z. B. Schadenmeldungen, Rechnungen, Gutachten, Schriftverkehr) als auch KI-basierte Analysen werden als übersichtliche Listen dargestellt. Insbesondere werden die KI-Ergebnisse in Markdown-Form als Liste präsentiert, was eine klare und strukturierte Übersicht automatisierter Analysen ermöglicht.

**Flexible Layout-Optionen:** Dank flexibler Layout-Optionen lässt sich mühelos zwischen Bildern und textbasierten Reports wechseln.

**Bilderslider & LoadButtons:** Ein Bilderslider ermöglicht darüber hinaus eine Fullscreen-Ansicht der Bilder, während LoadButtons der Benutzerin oder dem Benutzer erlauben, gezielt zwischen den unterschiedlichen Dokumenttypen zu wechseln.

### 5.10.4. Section 4 – Dokumentenmanagement und integrierter Chat

**Zusammengefasste Dokumente:** Section4 greift auf sämtliche Dokumente aus Section3 zu, die zuvor von einem KI-Agenten zusammengefasst wurden.

**Selektive Integration:** Die Sachbearbeiterinnen oder der Sachbearbeiter kann per Checkbox auswählen oder ausschließen, welche Zusammenfassungen in die finale Deckungsprüfung einfließen sollen.

**Automatisierte Versicherungsprüfung:** Mit einem einfachen Button-Klick kann die automatisierte Versicherungsprüfung gestartet werden – deren Ergebnisse werden im SummaryContainer übersichtlich dargestellt.

**Flexible Arbeitsfläche:** Dank der flexiblen Container-Struktur können einzelne Bereiche miniert oder wieder geöffnet werden, sodass stets eine übersichtliche Arbeitsfläche entsteht.

**Integrierter Chat:** Ergänzt wird dieser Bereich durch einen integrierten Chat, der einen vollständigen Nachrichtenverlauf führt und die direkte Kommunikation zum Schadensfall ermöglicht. Ein automatischer Scroll-Mechanismus stellt sicher, dass immer die neuesten Nachrichten sichtbar sind.

## 5.11. Leistungsbewertung

**Moderne Technologie & Framework:** Die Anwendung basiert auf React und TypeScript, wodurch ein hohes Maß an Wartbarkeit und Erweiterbarkeit erreicht wird.

**Effizientes Rendering:** Durch den Einsatz von React Hooks, Memoization (z. B. *React.memo* und *useMemo*) und einer klaren Trennung der Komponenten wird ein flüssiges und reaktions-schnelles Benutzererlebnis gewährleistet.

**Datenhandling:** Große Datensätze werden effizient in Listen und Tabellen dargestellt, während asynchrone API-Aufrufe und virtuelle Scroll-Container (z. B. *SimpleBar*) mögliche Performance-Kritikalitäten minimieren.

### 5.12. Mögliche Erweiterungen für die Zukunft

**Erweiterte Filter- und Suchfunktionen:** Integration von dynamischen Filtern und einer Suchfunktion zur gezielten Suche innerhalb der Datensätze.

**Responsive Optimierungen:** Weitere Anpassung des Layouts für mobile Geräte und Tablets sowie die Implementierung eines Dark-Mode-Switches.

**Performance-Optimierungen:** Zusätzliche Virtualisierungs- und Caching-Strategien, um auch bei extrem großen Datenmengen eine hohe Performance sicherzustellen.

### 5.13. Konklusion

Die entwickelte Anwendung stellt eine leistungsfähige und flexible Plattform dar, die den Ansprüchen moderner Versicherungs- und Schadensfallprozesse gerecht wird. Durch die dynamische Bereitstellung von Containern, die interaktive Datenvisualisierung und die Integration von KI-Analysen bietet das System eine intuitive und effiziente Benutzeroberfläche. Die klare Trennung in vier Hauptbereiche ermöglicht es, komplexe Informationen übersichtlich darzustellen und flexibel zu verwalten. Gleichzeitig bietet das System ausreichend Erweiterungspotenzial, um zukünftigen Anforderungen gerecht zu werden – sei es durch verbesserte Filter- und Suchfunktionen, Echtzeit-Kommunikation oder erweiterte KI-Analysen. Insgesamt schafft diese modulare und moderne Architektur eine solide Grundlage, auf der sich die Anwendung kontinuierlich weiterentwickeln und an neue technologische Entwicklungen anpassen lässt.

## 6. Projektmanagement

In diesem Kapitel wird erläutert, wie das Projektmanagement im Rahmen unseres Softwareprojektes organisiert wurde. Zunächst erfolgt eine kurze Erklärung der einzelnen Bereiche des Projektmanagements, die anschließend in separaten Kapiteln vertieft werden.

### 6.1. Aufgabenstellung des Gesamtprojekts

Die KLV hat das Problem, dass es bei der Schadenbearbeitung nicht möglich ist, einfach und schnell Betrugs- und Missbrauchsversuche bei Schadensmeldungen aufzudecken. Schäden werden nach Gefühl der jeweiligen Bearbeiter/innen manuell eingeschätzt und variieren von Person zu Person und deren Erfahrung. Aufgrund zukünftig möglicher personeller Engpässe wird dieses Problem verschärft.

### 6.2. Herausforderungen und Lösungsansätze

Im Folgenden werden wesentliche Herausforderungen beschrieben, die während des Projekts auftraten, sowie die Lösungsansätze des Teams dargestellt.

#### 6.2.1. Verzögerungen durch Datenzugang

Eine der größten Herausforderungen war der verzögerte Zugriff auf externe Daten der KLV. Um trotzdem voranzukommen, wurden frühzeitig technische Platzhalter wie Testserver und Testdaten eingesetzt. Zudem passten wir die Sprintplanung flexibel an und zogen unabhängige Tasks vor.

#### 6.2.2. Einschränkungen durch externe Vorgaben

Externe Vorgaben wie NDAs, regelmäßige Projektmeetings und Kundenwünsche, sowie die technische Pflicht, Microsoft Azure, React und FastAPI zu nutzen, schränkten den Handlungsspielraum ein. Durch frühzeitige und transparente Integration dieser Vorgaben sowie kontinuierliche Kommunikation mit Auftraggebern und Betreuern konnten wir dennoch agil bleiben und das Vertrauen aller Beteiligten sichern.

#### 6.2.3. Effektive Kommunikation

Die Kommunikation im Entwicklerteam erfolgte regelmäßig über Microsoft Teams, Signal und persönlichen Austausch in der Schule. Diese Kanäle erwiesen sich als ausreichend, um alle Teammitglieder stets aktuell zu informieren.

#### **6.2.4. Fazit und Lessons Learned**

Scrum und Azure DevOps erwiesen sich für unser Softwareprojekt als sehr effektiv. Die kurzen Planungszyklen und regelmäßigen Reviews führten zu hoher Produktqualität. Azure DevOps reduzierte zusätzlich den Verwaltungsaufwand erheblich.

Wöchentliche Meetings und Retrospektiven wurden anfangs als zusätzlicher Aufwand wahrgenommen, erwiesen sich jedoch als zeitsparend und fehlerreduzierend. Wichtigste Erkenntnisse: klare Kommunikation, diszipliniertes Tool-Management und Flexibilität bei der Prozessgestaltung sind wesentliche Erfolgsfaktoren für zukünftige Projekte.

#### **6.3. Scrum-Projektplan**

DevOps fördert die enge Zusammenarbeit zwischen Softwareentwicklung und IT-Betrieb, um die Softwarequalität und die Geschwindigkeit der Auslieferung zu verbessern. Im Projektmanagement werden dabei Anforderungen oft hierarchisch strukturiert: Ein Epic beschreibt eine Anforderung auf hoher Abstraktionsebene und wird in mehrere Issues unterteilt, die spezifische Funktionalitäten darstellen. Diese werden weiter in Tasks aufgeteilt, die konkrete Arbeitsschritte definieren. Issues dienen dazu, Probleme oder Fehler im System zu dokumentieren und zu verfolgen. Durch diese Strukturierung können Teams ihre Arbeit effizient planen und priorisieren, was zu einer effektiveren Umsetzung von Projekten führt.

### 6.3.1. Michael Griesser

The image displays four Jira epic cards for Michael Griesser. Each card shows the epic name, a 'Done' status indicator, the assignee 'GM\_EXT BPNsec.io', a progress bar, and a list of issues with their status (checked or unchecked).

- Backend Infrastruktur (1 issue):**
  - Done
  - GM\_EXT BPNsec.io
  - 11/11
  - + Add Issue
  - Setup DevOps
  - Setup Azure Tenant
  - NSG
  - Setup VM
  - NGINX Setup
  - Zertifikate / Certbot
  - Deploy Backend Repositories
  - Setup MySQL Database
  - Deploy Frontend
  - Setup Pinecone Database
  - Setup BlobStorage
- Bildanalyse (11 issues):**
  - Done
  - GM\_EXT BPNsec.io
  - 5/5
  - + Add Issue
  - API Endpoints
  - Insurance Chat Agent
  - Image analysis
  - Debugging Chat / Image
  - Parallele Analyse
- Azure OpenAI (282 issues):**
  - Done
  - GM\_EXT BPNsec.io
  - 5/5
  - + Add Issue
  - Setup Azure OpenAI
  - Sprachmodell GPT-4o
  - Embedding Modell
  - API Schnittstellen implementieren
  - Maximieren aller Limits in Azure
- API (273 issues):**
  - Done
  - GM\_EXT BPNsec.io
  - 8/8
  - + Add Issue
  - Backend FastAPI
  - Einbinden der Subrepos
  - Update all repositories
  - API Logging
  - Debugging Multiple Frontend Aufrufe
  - CORS
  - Multithreading.Uvicorn
  - systemd konfigurieren

Abbildung 6.1 Epics und Issues Griesser



**201 Dokumentenverwaltung** ...

● Done

AB ADB BPNsec.io

📅 22/22

+ Add Issue

- ✓ 📅 AB [End-to-End Integrationstest der Dokument...](#)
- ✓ 📅 AB [Implementierung des DocuWare-Clients](#)
- ✓ 📅 AB [Entwicklung der asynchronen Funktion do...](#)
- ✓ 📅 AB [Implementierung des Datei-Downloads \(d...](#)
- ✓ 📅 AB [MSG-zu-PDF-Konvertierung in msg\\_to\\_pdf...](#)
- ✓ 📅 AB [PDF-zu-JPG-Konvertierung für Bilder](#)
- ✓ 📅 AB [Extraktion von EXIF-Daten aus Bildern](#)
- ✓ 📅 AB [Integration von Azure Blob Storage für Upl...](#)
- ✓ 📅 AB [Generierung von SAS-URLs für hochgelade...](#)
- ✓ 📅 AB [Implementierung der Duplicate-Check-Logik](#)
- ✓ 📅 AB [Aktualisierung der Dateimetadaten in der ...](#)
- ✓ 📅 AB [Asynchrone Datei-Verarbeitung in databas...](#)
- ✓ 📅 AB [Fehlerbehandlung und Logging im Dokum...](#)
- ✓ 📅 AB [Unit-Tests für den DocuWare-Client](#)
- ✓ 📅 AB [Integration von Testdaten \(Fake-Daten\) im ...](#)
- ✓ 📅 AB [Verbesserung der ZIP-Dateiverarbeitung](#)
- ✓ 📅 AB [Konvertierung von Anhängen \(Emails, Rech...](#)
- ✓ 📅 AB [Validierung von Dateinamen und -typen](#)
- ✓ 📅 AB [Performance-Optimierung der asynchrone...](#)
- ✓ 📅 AB [Dokumentation des Dokumentenverwalту...](#)
- ✓ 📅 AB [Integration von Monitoring und Fehlerberi...](#)
- ✓ 📅 AB [Optimierung der DB-Schnittstelle für Datei...](#)

**224 KLV Clone** ...

● Done

AB ADB BPNsec.io

📅 22/22

+ Add Issue

- ✓ 📅 AB [End-to-End Integrationstest des KLV Clone...](#)
- ✓ 📅 AB [Einrichtung der Offline-Datenbankverbind...](#)
- ✓ 📅 AB [Implementierung des KLV Clone Engines](#)
- ✓ 📅 AB [Schema-Konvertierung: Bezeichner in Klein...](#)
- ✓ 📅 AB [Implementierung der Datenübertragung](#)
- ✓ 📅 AB [Deaktivierung der Fremdschlüsselprüfung i...](#)
- ✓ 📅 AB [Blockweise Datenübertragung \(z. B. 100.00...](#)
- ✓ 📅 AB [Überprüfung der Zeilenzahlen \(Source vs. ...](#)
- ✓ 📅 AB [Aktualisierung des Ziel-Datenbankschemas](#)
- ✓ 📅 AB [Integration von ThreadPoolExecutor für pa...](#)
- ✓ 📅 AB [Fehlerbehandlung und Rollback-Mechanis...](#)
- ✓ 📅 AB [Detailliertes Logging des Datenübertragun...](#)
- ✓ 📅 AB [Generierung eines Transfer-Reports](#)
- ✓ 📅 AB [Überprüfung der Datenkonsistenz](#)
- ✓ 📅 AB [Performance-Optimierung bei großen Dat...](#)
- ✓ 📅 AB [Integration von Konfigurationsparametern ...](#)
- ✓ 📅 AB [Unit-Tests für den Datenübertragungsproz...](#)
- ✓ 📅 AB [Dokumentation der KLV Clone Funktionalit...](#)
- ✓ 📅 AB [Absicherung gegen Parallelzugriffe](#)
- ✓ 📅 AB [Implementierung automatisierter Prüfunge...](#)
- ✓ 📅 AB [Integration von Mechanismen zum Zurück...](#)
- ✓ 📅 AB [Optionale Löschung von Pinecone-Daten](#)

Abbildung 6.3 Epics und Issues Di Bernardo 2

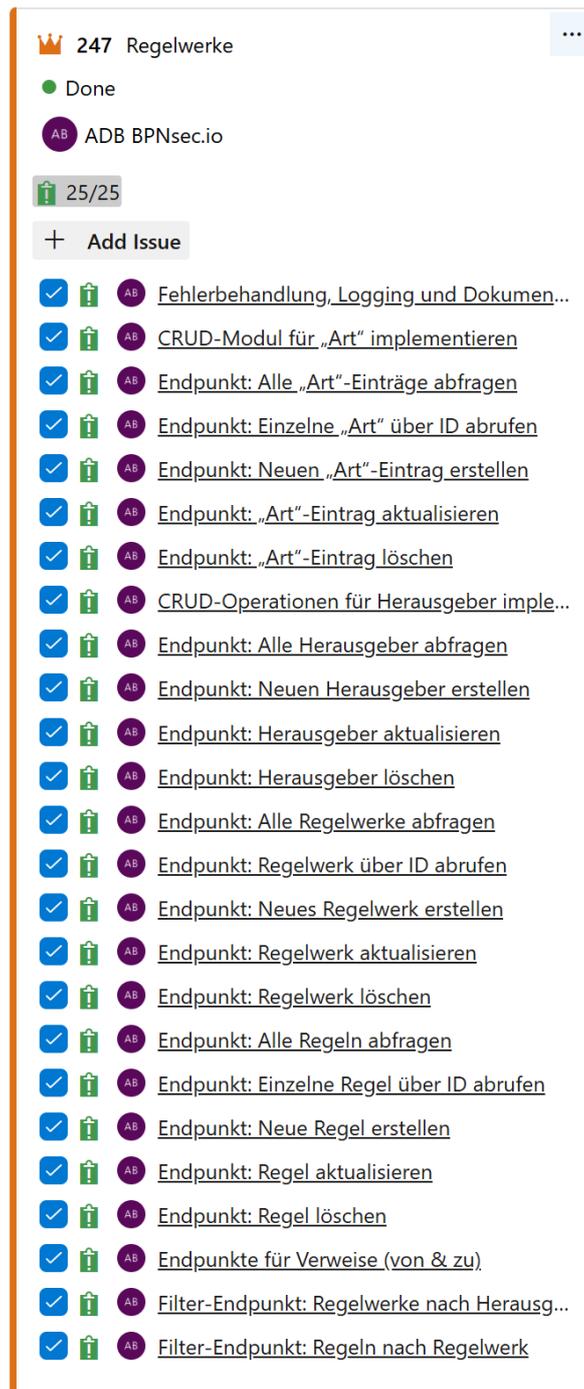


Abbildung 6.4 Epics und Issues Di Bernardo 3

### 6.3.3. Gordon Telsnig

**101** Generate, Review, and Explore AI Reports, Images, and Trade Sections in Section3

Done

GT BPNsec.io

26/26

+ Add Issue

- [Enable AI Report Generation in Section3](#)
- [Display Image Carousel for Report Explorat...](#)
- [Add Generate Report Button with Status In...](#)
- [Create Dynamic Sections for AI Report Data](#)
- [Provide Scrollable View for Large Reports](#)
- [Tooltip Support for Report Headers](#)
- [Add Header Controls for Navigation within...](#)
- [Display Latest AI Report Automatically afte...](#)
- [Handle Missing Image Error for Report Req...](#)
- [Manage API Call Status for Report Generati...](#)
- [Enable Keyboard Navigation for Image Car...](#)
- [Optimize AI Data Handling with Memoizati...](#)
- [Allow Expand/Collapse of Report Sections](#)
- [Dynamic Tooltip Content for Report Fields](#)
- [Persist User Preferences for Open/Closed S...](#)
- [Refine Tooltip Positioning for Enhanced Vis...](#)
- [Display Placeholder Text for Missing AI Data](#)
- [Use SimpleBar for Smooth Scrolling in Rep...](#)
- [Add Detailed Error Messages for API Failures](#)
- [Show Data Loading Indicator in Each Repo...](#)
- [Dynamic Button Labels Based on Report C...](#)
- [Simulate Typing Animation for AI Report D...](#)
- [Customize Typing Speed for Different Rep...](#)
- [Implement Section State Initialization in Se...](#)
- [Refactor CSS Styling for Section3 Report La...](#)
- [Enhance Scrollable Data List in Report Sect...](#)

**134** Develop the "Section3" feature that integrates an interactive image slider with a PDF document viewer for claim analysis.

Done

GT BPNsec.io

15/15

+ Add Issue

- [Create Base Component: Set up the Sectio...](#)
- [Implement Image Slider: Integrate an inter...](#)
- [Persist Image Index: Store and update the ...](#)
- [Add View Mode Toggle: Develop load butt...](#)
- [Integrate PDF Viewer: Embed a PDF viewer...](#)
- [Connect KI API: Implement the API call to t...](#)
- [Display KI Report: Render the retrieved KI...](#)
- [Add Tooltip for Slider: Provide contextual t...](#)
- [Tooltip for Toggle Buttons: Attach tooltips ...](#)
- [Implement Loading Indicators: Show appr...](#)
- [Handle API Errors: Gracefully display error...](#)
- [Enable Swipe Navigation: Ensure the imag...](#)
- [Optimize PDF Interaction: Enable zooming...](#)
- [Manage View State: Create a unified state ...](#)
- [Rework Markdown design](#)

Abbildung 6.5 Epics und Issues Telsnig 1

**71** Users can view and analyze individual EXIF metadata in a clear interface. ⋮

● Done

GT BPNsec.io

29/29

**+ Add Issue**

- [Exploration of ExiReader\(Python, Java\)](#)
- [Display detailed EXIF data within Section2,...](#)
- [mplement the ToggleCheckbox componen...](#)
- [Integrate the ImageSlider component with...](#)
- [Add functionality to expand and collapse t...](#)
- [Implement tooltips in Section2 to provide ...](#)
- [Add sorting functionality to Section2, allo...](#)
- [Parse and validate dates \(Schadendatum, B...](#)
- [Refactor and implement helper functions \(i...](#)
- [Implement formatExifData function to nor...](#)
- [Use calculateColorMap to assign color clas...](#)
- [Apply custom date formatting logic to dis...](#)
- [Implement sorting functions \(generateSort...](#)
- [Enable tooltips for headers in Section2, dis...](#)
- [Implement countMany function to classify...](#)
- [Use.getCellClass to validate and classify da...](#)
- [Develop functions in formatUtils.tsx to for...](#)
- [Implement generateTooltipContent functio...](#)
- [Implement mapDataToDisplay function to ...](#)
- [Set up getImageUrls function to extract an...](#)
- [Implement generateReport to create a rep...](#)
- [Develop getAdjustedImagePosition utility t...](#)
- [Create functions in dateUtils.ts to standard...](#)
- [Improve the ExifData component by imple...](#)
- [Refactor the tooltip handling in ExifData to...](#)
- [Update the ExifData component to persist...](#)
- [Use the colorMap generated in the ExifDat...](#)
- [Enhance ExifData by displaying a user-frie...](#)
- [Refactor tooltip and sorting functions \(gen...](#)

**6** WebFrontend for viewing and analyzing insuran claims. ⋮

● Done

GT BPNsec.io

28/28

**+ Add Issue**

- [Implement a Navbar with Icons](#)
- [Implementing licensing checking library](#)
- [Setup React/TS Project](#)
- [Implement a feature to display claim data](#)
- [Exploration of React Hooks](#)
- [Exploration of React Context API](#)
- [Exploration of CSS Stylesheet Rules](#)
- [Establish Project and Folder Structure](#)
- [Exploration Typescript vs. Proptypes](#)
- [Create and Implement an Input Field and L...](#)
- [Create and Adapt .gitignore File](#)
- [Implement responsive container logic](#)
- [Exploration MS Azure DevOps](#)
- [Create a Logo for the Application](#)
- [Exploration React Context-API](#)
- [Implement a Tooltipfunction](#)
- [Implement a Container component for data](#)
- [Exploration global-css and css styles](#)
- [Create separate utility files](#)
- [Implement Tooltip in Section2](#)
- [Implement global Tooltip](#)
- [Implement the image slider with all neces...](#)
- [Implement fullscreen mode for the image ...](#)
- [Implement the thumbnail tray for the imag...](#)
- [Implement a welcome screen that includes...](#)
- [Refactor App.tsx to improve data passing t...](#)
- [Connect and configure tooltip logic within ...](#)
- [Update key labels for improved mapping a...](#)

Abbildung 6.6 Epics und Issues Telsnig 2

The screenshot displays a Jira board interface. At the top, an epic is defined with a crown icon, ID 135, and the title "Create the 'Section4' dashboard that consolidates key claim data into a multi-column layout. Add a chatbot to Section4." The epic is currently in the "Doing" state. The assignee is listed as "GT BPNsec.io". A progress indicator shows "22/22" issues completed. Below the epic title is a button labeled "+ Add Issue". A list of 22 issues follows, each with a blue checkmark icon indicating completion. Each issue includes a green 'i' icon, a red 'GB' icon, and a truncated title. The issues listed are:

- State Persistence: Container states are inco...
- ReOpenUI Feedback: The ReOpenUI comp...
- Chat Auto-Scroll: The chat container's auto...
- Toggle Logic Duplication: Container minim...
- Develop Documents Container: Build a co...
- Integrate Documents API: Connect the Doc...
- Create Claim Report Container: Develop a ...
- Fetch Claim Report Data: Implement API in...
- Build Summary Container: Create a compo...
- Integrate Coverage Verification API: Conne...
- Add ReOpenUI Trigger: Embed a "ReOpen"...
- Develop Chat Container: Build a chat comp...
- Implement Chat Input: Create an auto-resi...
- Connect Chat API: Integrate backend APIs...
- Enable Container Toggling: Add functionali...
- Persist Container States: Store and restore...
- Create TopCenterMinimized Component: ...
- Add Loading Indicators: Implement clear l...
- Implement Error Handling: Display user-fri...
- Optimize API Calls: Ensure redundant netw...
- Add ReOpenUI Button in Summary Contai...
- Display Loader During API Call

Abbildung 6.7 Epics und Issues Telsnig 3

## 6.4. Inbetriebnahme

Die nachfolgende Installationsanleitung ist auch im Haupt-Repository als README.md beigelegt.

### 6.4.1. Features

- **REST API:** Endpunkte zur Abfrage und Verarbeitung von Schadens- und Dokumentendaten.
- **KI-gestützte Analysen:** Integration von Azure OpenAI für AI-Reports und Zusammenfassungen.
- **Umfassende Datenbankankbindung:** Asynchrone und synchrone Operationen mit diversen Datenbanken (z. B. viva\_docs, viva\_prd\_clone, viva\_regelwerk).
- **Dokumentenverarbeitung:** PDF-Konvertierung, EXIF-Auslesung, MSG-zu-PDF-Konvertierung, Azure Blob Storage-Uploads.
- **Modulare Architektur:** Klar strukturierte Module für API, Datenbanken, Dokumentenverarbeitung, Regelwerksverarbeitung und externe Integrationen.

### 6.4.2. Installation & Setup

#### Systemvoraussetzungen:

- Python 3.11 oder höher
- pip (idealerweise virtuelle Umgebung)

#### Installationsschritte:

1. Repository klonen:
2. `git clone <repository-url>`
3. `cd KLV_DATA_PROCESSING`
4. Virtuelle Umgebung erstellen und aktivieren:
  - Windows:
    - `python -m venv venv`
    - `venv\Scripts\activate`
  - Linux/Mac:
    - `python -m venv venv`
    - `source venv/bin/activate`
5. Abhängigkeiten installieren:
6. `pip install -r requirements.txt`
7. Umgebungsvariablen konfigurieren:
  - `.env`-Datei im Projektstamm anlegen.
  - Variablen (z. B. DB-Zugangsdaten, Azure Keys) in `.env` eintragen.

### 6.4.3. Konfiguration

#### Datenbanken:

- Offline-Datenbank (viva\_prd\_clone)
- Online-Datenbank (Azure MySQL)

Einstellungen erfolgen in config.py.

#### Azure Blob Storage:

- Container-Namen, Connection-String etc. in config.py konfigurieren.

#### KI-Integration:

- Azure OpenAI API-Endpunkte in config.py einstellen.

#### Testbetrieb vs. Livebetrieb:

- Steuerung über Parameter TESTBETRIEB in config.py.

### 6.4.4. Verzeichnisstruktur

- api/: FastAPI-Endpunkte
- claim\_coverage\_assessment/: Deckungsprüfungsmodule
- batch\_job/: Skripte für Batch-Verarbeitung
- database\_management/: Datenbank-Engines und Utilities
- database\_processing/: Verarbeitung von Datenbankabfragen
- document\_processing/: Dokumentenverarbeitung
- external\_integration/: Externe Services (z. B. Azure Image Analysis)
- html\_regelverwaltung\_seite/: Web-Oberfläche Regelverwaltung
- logging\_setup.py: Zentrales Logging
- models/: SQLAlchemy-Datenmodelle
- regelwerk\_processing/: CRUD-Operationen für Regelwerk
- tests/: Unit- und Integrationstests
- requirements.txt: Python-Abhängigkeiten

### 6.4.5. Nutzung der API

#### Server starten:

```
uvicorn api.endpoint:app --reload
```

Standardmäßig unter: <http://127.0.0.1:8000>

### API-Endpunkte (Auswahl):

- Schadensdaten: /claim/{claim\_id}
- Schadenmeldung: /claimreport/{claim\_id}
- Dokumentenzusammenfassung: /claimdocs/{claim\_id}
- AI Report: /claimaireport/{claim\_id}
- Deckungsprüfung: /claimcoverage/{claim\_id} (POST)
- Regelwerk-CRUD: /art, /herausgeber, /regelwerke, /regeln

Vollständige Dokumentation über Swagger-UI unter /docs.

### 6.4.6. Tests

Tests mittels pytest ausführen:

pytest

Stelle sicher, dass TESTBETRIEB aktiv ist (in config.py).

### 6.4.7. Weiterentwicklung & Contributing

#### Code Style:

- Vorhandenen Stil beachten
- Funktionen und Prozesse klar kommentieren

#### Modularität:

- Neue Features strukturiert in Modulen integrieren und dokumentieren

#### Tests:

- Unit- und Integrationstests ergänzen

### 6.5. Kostenaufstellung

Da es sich um ein reines Software - und Cloud Projekt handelt sind somit keine Hardwareanschaffungen nötig. Die entstehenden Kosten für das Hosting der Ressourcen werden zentral über Azure abgerechnet. Die monatlichen Kosten, die für den Betrieb in der Cloud entstehen können in Azure durch den Kostenmanager visualisiert werden (siehe Abbildung 6.8).

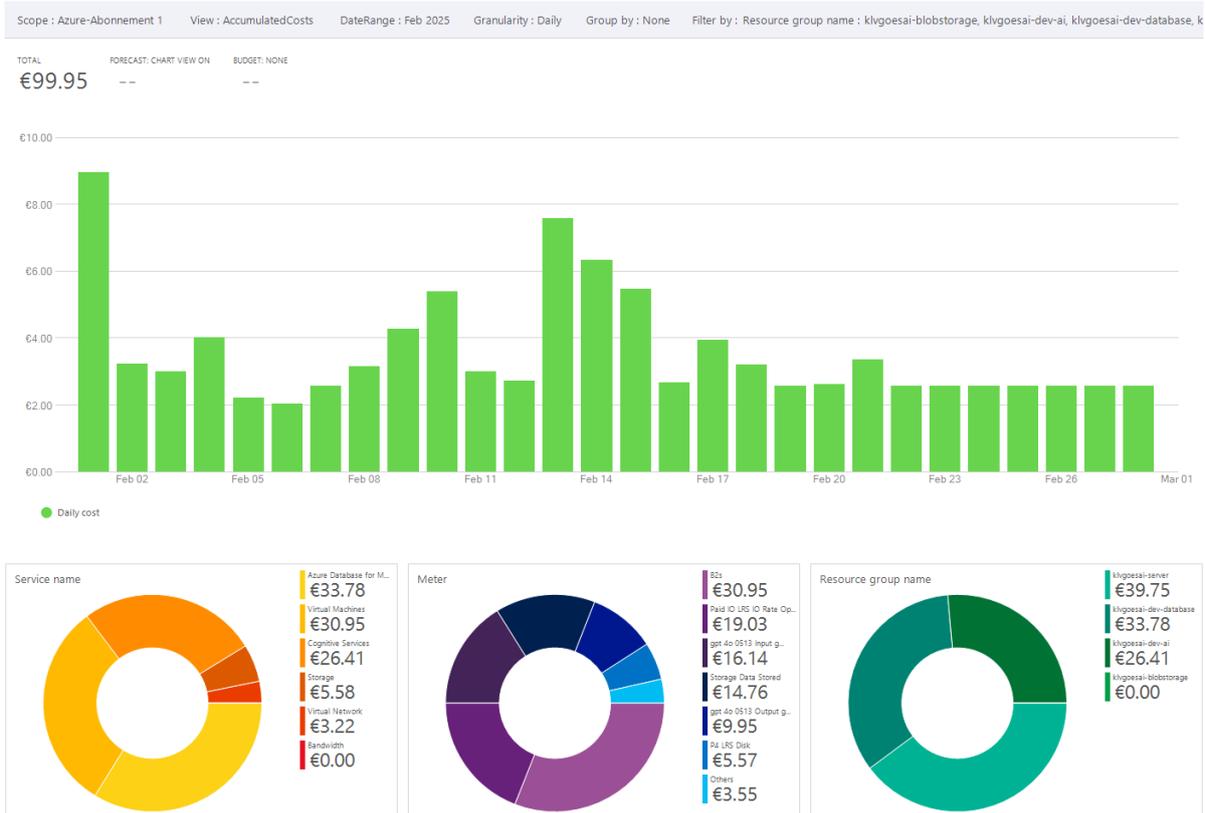


Abbildung 6.8 monatliche Kosten der App (Entwicklung)

Diplomarbeit: KLV goes AI, 31.03.2025		
Kostenart	Stückpreis / Einheiten	Kosten in € / Monat
Cloud Server	1	30.95
Datenbank MySQL	1	33.78
Datenbank Pinecone	1	0.37
Datenbank Blob Storage	1	< 0.01
OpenAI Instanz	1	26.41
Sonstiges (Traffik / Storage)		8.44
<b>Gesamtkosten</b>		<b>99.95</b>

Tabelle 6.1 Kostenaufstellung

## 6.6. Besprechungsprotokolle

HTL   MÖSSINGERSTRASSE		ERGEBNISPROTOKOLL ZUR BESPRECHUNG		Eingangs- vermerke
AN		Veranstalter		
Teilnehmer und Unterrichtete		Protokollführer	Telsnig Gordon	
		Projektbetreuer	MSc. Peter Gigler	Erstellungsdatum: 14.10.2024
	am Tag, Datum	am Montag, 14.10.2024		Ort/Raum HTL Mössingstraße, W101-1
<b>Thema:</b> <b>Projektstartgespräch / Projektstatusgespräch</b>				
Projektname: KLV goes AI - Einsatz von Künstlicher Intelligenz für die automatisierte Betrugserkennung und Schadensbewertung.				
Teilnehmer (alphabetisch)		Unterrichtete (alphabetisch)		
Name	Klasse, Standort	Name	Klasse, Standort	
Hr. Dr. Daniel Wischounig-Struel	--,Klagenfurt	Hr. Di Bernardo Alexander	7AAIF, Klagenfurt	
Hr. Msc. Peter Gigler	--,Klagenfurt	Hr. Griesser Michael	7AAIF, Klagenfurt	
		Hr. Telsnig Gordon	7AAIF, Klagenfurt	
<p><b>Di Bernardo:</b> <b>Hauptaufgabe:</b> Entwicklung einer <b>vektorierten Datenbank</b> für KI-basierte Systeme. Integration von <b>API-Schnittstellen</b> zur Abfrage der bestehenden Datenbank sowie Hosting und Skalierung der Lösung mittels <b>cloudbasierter Infrastruktur</b>.</p> <p><b>Erledigt:</b></p> <ul style="list-style-type: none"> <li>Konvertierung der KLV-Datenbank (Oracle) zu einer Azure MySQL-Datenbank – vollautomatisch mittels Scripts und unter Einhaltung aller Azure MySQL-Datenbank-Regeln</li> <li>Erstellung kompletter Datenmodelle für den Backend-Einsatz</li> <li>Entwicklung eines Endpunktes zur Übergabe der Daten an das Frontend</li> </ul> <p><b>TODO:</b></p> <ul style="list-style-type: none"> <li>Erstellung einer Datenbank sowie eines Frontend zur Regelerfassung</li> <li>Aufbau einer Vektordatenbank zur Überprüfung der Regeln in der Deckungsprüfung</li> <li>Implementierung weiterer Endpunkte für Dokumente und Bilder im Frontend</li> <li>Entwicklung der KI-basierten Deckungsprüfung</li> <li>Optimierung der API-Performance sowie Erweiterung und Detaillierung der Logging- und Monitoring-Mechanismen</li> </ul> <p><b>Feedback:</b></p> <ul style="list-style-type: none"> <li>Detailliertere Beschreibung der Epics und ToDos im Azure Board</li> <li>Kleine Änderungen in der Präsentation notwendig</li> <li>Das Arbeitspensum ist anspruchsvoll, wird sich das Ausgehen?</li> </ul>				<b>Erledigung von, Termin</b>
HTL Mössingerstraße Besprechung 1_7AAIF				Seite 1 von 2 V01

Abbildung 6.9 Besprechungsprotokoll 14.10.2024 Seite 1

## HTL - Mössingerstraße

### Griesser:

**Hauptaufgabe:** Entwicklung und Implementierung einer **cloudbasierten Infrastruktur** sowie eines skalierbaren **Backend-Systems** zur Unterstützung eines **KI-Systems** für eine automatisierte Schadensanalyse.

### Ergebnisse:

- Bereitstellung der MySQL Datenbank
- Bereitstellung der OpenAI Instanz
- Bereitstellung der VM
- DevOps und Repositories

### TODO:

- Bereitstellen der Vector Datenbank
- Verbesserungen der API
- Implementieren der Bildanalyse
- Frontend bereitstellen

### Feedback:

- Folien besser strukturieren

### Telsnig:

**Hauptaufgabe:** Evaluierung moderner Frontend-Frameworks und Implementierung einer Webanwendung. Zudem erfolgt eine Analyse von Bild Metadaten zur Betrugsbekämpfung.

### Erledigt:

- G.u.M. – fertiggestellt
- Section1 (Übersicht) – fertiggestellt
- Section2 (Exifdaten V1.0) – fertig implementiert
- Test API – eingebunden

### TODO:

- Exiflogik – Optimierung erforderlich
- Section3 & Section4 – Finalisierung ausstehend

### Feedback:

- PP-Folie Projektübersicht – besser strukturieren
- Exiflogik – weitere Verbesserungen notwendig

14.10.2024   
Datum gez. Name

AN  Teilnehmer und Unterrichtete	<b>ERGEBNISPROTOKOLL ZUR BESPRECHUNG</b>	Eingangs- vermerke          Erstellungsdatum: 18.11.2024
	Veranstalter	
	Protokollführer Telsnig Gordon	
	Projektbetreuer MSc. Peter Gigler	

am Tag, Datum am Montag, 18.11.2024	Ort/Raum HTL Mössingstraße, W101-1
--	---------------------------------------

**Thema:**  
**Projektstartgespräch / Projektstatusgespräch**

Projektname: KLV goes AI - Einsatz von Künstlicher Intelligenz für die automatisierte Betrugserkennung und Schadensbewertung.

Teilnehmer (alphabetisch)		Unterrichtete (alphabetisch)	
Name	Klasse, Standort	Name	Klasse, Standort
Hr. Dr. Daniel Wischounig-Strucl	--,Klagenfurt	Hr. Di Bernardo Alexander	8AAIF, Klagenfurt
Hr. Msc. Peter Gigler	--,Klagenfurt	Hr. Griesser Michael	8AAIF, Klagenfurt
		Hr. Telsnig Gordon	8AAIF, Klagenfurt

**Di Bernardo:**

**Hauptaufgabe:** Entwicklung einer **vektorierten Datenbank** für KI-basierte Systeme. Integration von **API-Schnittstellen** zur Abfrage der bestehenden Datenbank sowie Hosting und Skalierung der Lösung mittels **cloudbasierter Infrastruktur**.

**Erledigt:**

- Aufbau der Regeldatenbank abgeschlossen
- Frontend für die Regeldatenbank größtenteils fertiggestellt
- Erste Ansätze für zusätzliche API-Endpunkte erkennbar

**TODO:**

- Fertigstellung und Integration der zusätzlichen Endpunkte in die bestehende Architektur
- Vervollständigung der fehlenden Funktionalitäten im Regel-Frontend
- Sequenzdiagramm für /claimimages
- Sicherheitsoptimierung: Austausch des Plaintext-Passworts, da dieses nur zum Testen gedacht war
- Aufbau einer Vektordatenbank zur Überprüfung der Regeln in der Deckungsprüfung
- Entwicklung der KI-basierten Deckungsprüfung
- Bilder zukünftig im Blob speichern
- Optimierung der API-Performance sowie Erweiterung und Detaillierung der Logging- und Monitoring-Mechanismen

**Feedback:**

- Das Plaintext-PW darf auf keinen Fall in Produktion gehen
- Wünscht sich ein Sequenzdiagramm für leichteres Verständnis
- Bilder sollten nicht als Base64 in Datenbank gespeichert werden (Alternative: Blob)
- Insgesamt sehr guter Aufbau – weiter so!

Erledigung  
von, Termin

Abbildung 6.11 Besprechungsprotokoll 18.11.2024 Seite 1

## HTL - Mössingerstraße

### Griesser:

**Hauptaufgabe:** Entwicklung und Implementierung einer **cloudbasierten Infrastruktur** sowie eines skalierbaren **Backend-Systems** zur Unterstützung eines **KI-Systems** für eine automatisierte Schadensanalyse.

### **Ergebnisse:**

- Backend API erweitert
- Regelseite gehostet
- OpenAI Bildanalyse

### **TODO:**

- Vektor Datenbank
- Embedding Modell
- Blob Storage

### **Feedback:**

- Keine Base64 Bilder in der Datenbank -> Blob Storage
- Bessere Diagramm auf Folien

### Telsnig:

**Hauptaufgabe:** Evaluierung moderner Frontend-Frameworks und Implementierung einer Webanwendung. Zudem erfolgt eine Analyse von Bild Metadaten zur Betrugsbekämpfung.

### **Erliegt:**

- Section2: Finalisierung abgeschlossen
- Welcomescreen: Implementierung abgeschlossen
- API-Endpunkte: Alle Endpunkte von der Test-API auf die Produktions-API umgestellt

### **TODO:**

- Design: Weitere Optimierungen vornehmen
- Section3: Buttons responsiv gestalten

### **Feedback:**

- API: Sehr gute Umsetzung, etwa durch den Einsatz von Singleton-Pattern

18.11.2024  
Datum gez. Name

HTL Mössingerstraße  
Besprechung\_2\_7A/IF

Seite 2 von 2  
V01

Abbildung 6.12 Besprechungsprotokoll 18.11.2024 Seite 2

AN  Teilnehmer und Unterrichtete	<b>ERGEBNISPROTOKOLL ZUR BESPRECHUNG</b>	Eingangs- vermerke
	Veranstalter	
	Protokollführer Telsnig Gordon	
	Projektbetreuer MSc. Peter Gigler	Erstellungsdatum: 17.02.2025

am Tag, Datum am Montag, 17.02.2025	Ort/Raum HTL Mössingstraße, W101-1
--	---------------------------------------

**Thema:**  
**Projektstartgespräch / Projektstatusgespräch**

Projektname: KLV goes AI - Einsatz von Künstlicher Intelligenz für die automatisierte Betrugserkennung und Schadensbewertung.

Teilnehmer (alphabetisch)		Unterrichtete (alphabetisch)	
Name	Klasse, Standort	Name	Klasse, Standort
Hr. Dr. Daniel Wischounig-Struel	--,Klagenfurt	Hr. Di Bernardo Alexander	8AAIF, Klagenfurt
Hr. Msc. Peter Gigler	--,Klagenfurt	Hr. Griesser Michael	8AAIF, Klagenfurt
		Hr. Telsnig Gordon	8AAIF, Klagenfurt

**Di Bernardo:**

**Hauptaufgabe:** Entwicklung einer vektorisierten Datenbank für KI-basierte Systeme. Integration von API-Schnittstellen zur Abfrage der bestehenden Datenbank sowie Hosting und Skalierung der Lösung mittels cloudbasierter Infrastruktur.

**Erledigt:**

- Fertigstellung und Integration der zusätzlichen Endpunkte in die bestehende Architektur
- Vervollständigung der fehlenden Funktionalitäten im Regel-Frontend (inkl. Sicherheit)
- Sequenzdiagramm für /claimimages erstellt
- Aufbau der Vektordatenbank zur Überprüfung der Regeln in der Deckungsprüfung abgeschlossen
- Batch zur Übertragung der Daten von der MySQL Datenbank in Vektordatenbank
- Entwicklung der KI-basierten Deckungsprüfung abgeschlossen
- Bildverwaltung: Bilder werden jetzt in einem Blob gespeichert und Session-Links ans Frontend übergeben
- Erweiterung und Detaillierung der Logging- und Monitoring-Mechanismen

**TODO:**

- Optimierung der API-Performance
- Prompt-Optimierung für KI-Logiken

**Feedback:**

- Super Präsentation
- Sehr erfreut über die sofortige Umsetzung des Sequenzdiagramms
- Hinweis: Die Folie für die englische Präsentation sollte auch die Diagramme in Englisch enthalten

Erledigung  
von, Termin

Abbildung 6.13 Besprechungsprotokoll 17.02.2025 Seite 1

## HTL - Mössingerstraße

### Griesser:

**Hauptaufgabe:** Entwicklung und Implementierung einer **cloudbasierten Infrastruktur** sowie eines skalierbaren **Backend-Systems** zur Unterstützung eines **KI-Systems** für eine automatisierte Schadensanalyse.

### Ergebnisse:

- Vektor Datenbank bereitgestellt
- Embedding Modell bereitgestellt
- Blob Storage bereitgestellt
- API erweitert
- Parallele Bildanalyse

### TODO:

- Performance

### Feedback:

- Gute englische Präsentation
- Loop im Flowchart nicht klar gekennzeichnet

### Telsnig:

**Hauptaufgabe:** Evaluierung moderner Frontend-Frameworks und Implementierung einer Webanwendung. Zudem erfolgt eine Analyse von Bild Metadaten zur Betrugsbekämpfung.

### Erledigt:

- Section3 finalisiert
- Section4 Chatbot integriert und designend
- 2 API Endpunkte von Test API auf Produktions-Endpunkte umgestellt

### TODO:

- Exiflogik – Optimierung erforderlich
- Section4 – Finalisierung ausstehend

### Feedback:

- Gute Präsentation
- Präsentation: Die Präsentation auf Englisch war überzeugend
- Projektumfang nicht ausufern lassen.

17.2.2025   
Datum gez. Name  
HTL Mössingerstraße  
Besprechung 3\_7AAIF

Seite 2 von 2  
V01

Abbildung 6.14 Besprechungsprotokoll 17.02.2025 Seite 2

AN  Teilnehmer und Unterrichtete	<b>ERGEBNISPROTOKOLL ZUR BESPRECHUNG</b>	Eingangs- vermerke          Erstellungsdatum: 10.03.2025
	Veranstalter	
	Protokollführer Telsnig Gordon	
	Projektbetreuer MSc. Peter Gigler	

am Tag, Datum	Ort/Raum
am Montag, 10.03.2025	HTL Mössingstraße, W101-1

**Thema:**  
**Projektstartgespräch / Projektstatusgespräch**

Projektname: KLV goes AI - Einsatz von Künstlicher Intelligenz für die automatisierte Betrugserkennung und Schadensbewertung.

Teilnehmer (alphabetisch)		Unterrichtete (alphabetisch)	
Name	Klasse, Standort	Name	Klasse, Standort
Hr. Dr. Daniel Wischounig-Struel	--,Klagenfurt	Hr. Di Bernardo Alexander	8AAIF, Klagenfurt
Hr. Msc. Peter Gigler	--,Klagenfurt	Hr. Griesser Michael	8AAIF, Klagenfurt
		Hr. Telsnig Gordon	8AAIF, Klagenfurt

**Di Bernardo:**

**Hauptaufgabe:** Entwicklung einer **vektorsierten Datenbank** für KI-basierte Systeme. Integration von **API-Schnittstellen** zur Abfrage der bestehenden Datenbank sowie Hosting und Skalierung der Lösung mittels **cloudbasierter Infrastruktur**.

**Erledigt:**

- Optimierung der API-Performance
- Prompt-Optimierung für KI-Logiken

**TODO:**

- Diplomarbeit fertigstellen

**Feedback:**

- Großartige Arbeit

Erledigung  
von, Termin

Abbildung 6.15 Besprechungsprotokoll 10.03.2025 Seite 1

## HTL - Mössingerstraße

### Griesser:

**Hauptaufgabe:** Entwicklung und Implementierung einer **cloudbasierten Infrastruktur** sowie eines skalierbaren **Backend-Systems** zur Unterstützung eines **KI-Systems** für eine automatisierte Schadensanalyse.

### **Ergebnisse:**

- Monitoring Server
- Prompts optimiert

### **TODO:**

- Optimierungen der API
- Erhöhen der Lizenz
- Diplomarbeit fertigstellen

### **Feedback:**

- Sehr gut

### Telsnig:

**Hauptaufgabe:** Evaluierung moderner Frontend-Frameworks und Implementierung einer Webanwendung. Zudem erfolgt eine Analyse von Bild Metadaten zur Betrugsbekämpfung.

### **Erledigt:**

- Section4: Finalisierung abgeschlossen
- Welcomescreen: Implementierung abgeschlossen
- API-Endpunkte: Alle Endpunkte von der Test-API auf die Produktions-API umgestellt

### **TODO:**

- Design: Weitere Optimierungen vornehmen
- Section3: Buttons responsiv gestalten

### **Feedback:**

- API: Sehr gute Umsetzung, etwa durch den Einsatz von Singleton-Pattern

10.03.2025  
Datum gez. Name  
HTL Mössingerstraße  
Besprechung 4\_7AAIF



Seite 2 von 2  
V01

Abbildung 6.16 Besprechungsprotokoll 10.03.2025 Seite 2

## 6.7. Arbeitszeitnachweis

### 6.7.1. Alexander Di Bernardo

Datum	Task	Beschreibung	Dauer in h
03.03.2024	001	Projektbesprechung mit Projektteam KLV und BPN	2,00
19.06.2024	002	Erstellen eines Datenkonzept zur Freigabe der Echtdateien der KLV	3,50
21.06.2024	004	Erstellen Datenschutzfolgeabschätzung mit KLV	3,50
25.06.2024	005	Präsentation Vorstand KLV + Genehmigung Datenfreigabe	0,50
27.06.2024	003	Begleitung Hr. Telsnig Besuch bei KLV zur Bedarfsanalyse für das Frontend	3,50
01.07.2024	006	Udemy Kurzs - Master Vector Databases	2,00
08.07.2024	007	Udemy Kurzs - Master Vector Databases	2,00
15.07.2024	008	Udemy Kurzs - Master Vector Databases	2,00
22.07.2024	009	Udemy Kurzs - Master Vector Databases	1,50
29.07.2024	010	G.u.M - Einführung in Vektordatenbanken, Definition, Funktionsweise	2,50
31.07.2024	011	G.u.M - Untersuchung der Leistungsfähigkeit verschiedener Vektordatenbanken	4,00
02.08.2024	012	G.u.M - Einsatzmöglichkeiten von Vektordatenbanken im Versicherungsbetrieb	2,00
06.08.2024	013	G.u.M - Datenschutzerfordernungen und deren Umsetzung in Vektordatenbanken	2,50
09.08.2024	014	G.u.M - Vergleichende Analyse der Leistungsfähigkeit und Skalierbarkeit	3,00
12.08.2024	015	G.u.M - Pinecone in Azure: Integration und Vorteile	3,00
14.08.2024	016	G.u.M - Optimierung von Entscheidungsprozessen durch Vektordatenbanken	2,00
16.08.2024	017	G.u.M - KI-gestützte Systeme und deren Verbindung zu Vektordatenbanken	2,50
20.08.2024	018	G.u.M - Recherche zu datenschutzkonformen Vektordatenbank-Lösungen	3,00
02.09.2024	019	G.u.M - Zusammenfassung der Analyseergebnisse zur Skalierbarkeit	3,00
04.09.2024	020	G.u.M - Schlussfolgerungen zur Datenschutzkonformität	3,50
04.09.2024	021	Teamsmeeting mit Projektteam (Wie geht's weiter?)	1,00
05.09.2024	022	Teamsmeeting mit KLV SISO(Ing. Harald Klary) und Projektteam	0,50
06.09.2024	023	G.u.M - Vorbereitung der finalen Evaluation	2,00
11.09.2024	024	Klärungen zur Datenlieferung der KLV	0,50
12.09.2024	025	Recherche-FastAPI	2,00
13.09.2024	026	Recherche-FastAPI	2,00
14.09.2024	027	Vertraut machen mir der Programmiersprache Python	1,00
15.09.2024	028	Vertraut machen mir der Programmiersprache Python	1,00
20.09.2024	029	Teamsmeeting mit Projektteam (Azure Devops)	2,00
27.09.2024	030	Aufsetzen von Projekt, Gitignore, Git-> Remote Work	1,50
27.09.2024	031	MS-Azure DevOps, Anlegen Ordnerstruktur, main/dev	1,50
28.09.2024	032	Erstellung erner Testdatenbank (ELGA) bis live DB von KLV verfügbar	1,00
05.10.2024	033	Bau von Abfragen auf die ELGA DB	2,00
06.10.2024	034	Aufbau der FastAPI mit den Testdaten	3,00
10.10.2024	035	Recherche SQLAlchemy + Codegen	4,00
11.10.2024	036	Teamsmeeting mit Projektteam(Präsentation 1.Meilenstein)	2,00

Datum	Task	Beschreibung	Dauer in h
12.10.2024	037	Automatisierte Datenmodelle mittels Codegen	2,00
12.10.2024	038	Neuaufbau der API mit den ELGA Testdaten	2,00
13.10.2024	039	Teamsmeeting mit Projektteam(Präsentation 1.Meilenstein) Überarbeitung	0,50
14.10.2024	040	Code-Review und Refactoring der API	5,50
15.10.2024	041	Erweiterung der Batch-Prozesse	7,00
16.10.2024	042	Integration von zusätzlichen Tests	5,00
17.10.2024	043	Optimierung der Datenbankabfragen	5,50
21.10.2024	044	Implementierung von Logging-Verbesserungen	4,50
22.10.2024	045	Erweiterung der Code-Dokumentation	3,50
23.10.2024	046	Teamsmeeting zur Abstimmung	2,50
28.10.2024	047	Optimierung der Pinecone-Integration	7,00
29.10.2024	048	Fehlerbehebung in der Schnittstellenlogik	5,50
30.10.2024	049	Integration von Performance-Monitoring	3,50
02.11.2024	050	Code-Optimierung und Cleanup	5,50
05.11.2024	051	Erweiterung der API um neue Endpunkte	5,50
07.11.2024	052	Testing und Bugfixing	5,50
10.11.2024	053	Deployment-Vorbereitung	4,50
12.11.2024	054	Integration in Azure DevOps	3,50
14.11.2024	055	Dokumentation der technischen Umsetzung aktualisieren	3,50
18.11.2024	056	Durchführung von Lasttests	6,00
20.11.2024	057	Optimierung der Batch-Job Performance	5,50
20.11.2024	058	Gliederung und Konzept der Diplomarbeit erstellen – Struktur, zentrale Fragestellungen und Methodik definieren	7,00
22.11.2024	059	Weitere Tests und Debugging	4,50
25.11.2024	060	Code-Review und Optimierung	3,50
28.11.2024	061	Integration zusätzlicher Monitoring-Tools	5,50
02.12.2024	062	Anpassungen an der API	3,50
05.12.2024	063	Bugfixing und Stabilitätstests	5,50
08.12.2024	064	Umfassende Literaturrecherche und Quellenanalyse durchführen; vorläufiges Literaturverzeichnis erstellen	10,00
09.12.2024	065	Vorbereitung des Rollouts	3,50
11.12.2024	066	Durchführung eines Deployment-Dry-Runs	4,50
13.12.2024	067	Nachbesprechung des Dry-Runs	3,00
15.12.2024	068	Kapitelentwürfe verfassen – Erstellung von Entwürfen für Einleitung, Methodik, Theorieteil und erste Ergebnisdarstellungen	8,00
16.12.2024	069	Anpassung der Systemkonfiguration	3,00
19.12.2024	070	Finales Debugging vor Rollout	4,50
22.12.2024	071	Rollout in die Produktionsumgebung	6,00
03.01.2025	072	Stabilitätstests im Live-Betrieb	4,00
05.01.2025	073	Anpassung und Optimierung basierend auf Feedback	5,50
08.01.2025	074	Weitere Tests und Code-Optimierungen	3,50
10.01.2025	075	Vorbereitung für Abschlusspräsentation	3,00
10.01.2025	076	Überarbeitung und inhaltliche Optimierung der Kapitel; Integration von Feedback und Verbesserungsvorschlägen	6,00
12.01.2025	077	Durchführung eines letzten Teamsmeetings	2,50
15.01.2025	078	Finalisierung der technischen Dokumentation	3,50
18.01.2025	079	Letzte Anpassungen an der API	3,50

Datum	Task	Beschreibung	Dauer in h
19.01.2025	080	Integration von Abbildungen, Tabellen und Ergebnissen; erste Gesamtzusammenstellung der Diplomarbeitsdokumentation	8,50
21.01.2025	081	Code-Refactoring und abschließender Cleanup	4,50
24.01.2025	082	Finales Testing und Bugfixing	5,50
27.01.2025	083	Optimierung der Datenbankanfragen	4,50
30.01.2025	084	Durchführung eines Final Reviews	3,50
02.02.2025	085	Letzte Performance-Tests	3,50
05.02.2025	086	Vorbereitung der Abschlussdokumentation (technisch)	5,50
08.02.2025	087	Abschlussabnahme der technischen Umsetzung	3,50
10.02.2025	088	Erweiterte Integrationstests und Performance-Analysen (Deckungsprüfung)	5,50
10.02.2025	089	Abschlusskorrektur und Formatierung der Diplomarbeit – Rechtschreibprüfung, Layout-Optimierung und Zitierstil anpassen	7,00
11.02.2025	090	Optimierung der Sicherheitsfeatures und intensives Bugfixing (Deckungsprüfung)	4,50
12.02.2025	091	Rollout-Abschlussbesprechung	3,00
12.02.2025	092	Zusammenführung und Optimierung des Monitoringsystems (Deckungsprüfung)	3,50
13.02.2025	093	Durchführung umfassender Lasttests (Deckungsprüfung)	4,00
14.02.2025	094	Finales Code-Refactoring und umfassende technische Dokumentation erstellen (Deckungsprüfung)	11,50
15.02.2025	095	Abschlussbesprechung und Übergabe an den Betrieb (technisch, Deckungsprüfung)	5,50
16.02.2025	096	Letzte finale Überprüfung	2,00
16.02.2025	097	Letzte finale inhaltliche Überprüfung, Abgabevorbereitung und Erstellung des finalen Dokumentation	3,00
17.02.2025	098	Diplomarbeit finalisieren: Kapitelfeinschliff	0,50
19.02.2025	099	Diplomarbeit finalisieren: Überprüfung & Feinschliff	0,50
22.02.2025	100	Diplomarbeit finalisieren: intensives Schreiben & Layout	2,50
23.02.2025	101	API-Performance optimieren: Datenbank-Queries und Profiling	2,00
26.02.2025	102	API-Performance optimieren: Caching-Konzept und erste Tests	1,00
01.03.2025	103	Prompts optimieren: Feintuning & KI-Logik	2,00
02.03.2025	104	Prompts optimieren: Edge Cases & weitere Tests	2,00
05.03.2025	105	Abschließende Arbeiten Diplomarbeit: Formatierung und Layout	1,00
07.03.2025	106	Abschließende Arbeiten Diplomarbeit: Rechtschreibprüfung	0,50
08.03.2025	107	Abschließende Arbeiten Diplomarbeit: Letzter Check & Abgabevorbereitung	1,50
<b>Gesamtarbeitszeit (Ist)</b>			<b>377,50</b>
Soll			160,00
Offen (Ist-Soll)			+217,50

Tabelle 6.2 Arbeitszeitnachweis von Alexander Di Bernardo

## 6.7.2. Michael Griesser

Datum	Task	Beschreibung	Dauer in h
14.08.2024	001	Grundlagenarbeit AI Teil 1	8,00
15.08.2024	002	Grundlagenarbeit AI Teil 2	8,00
17.08.2024	004	Grundlagenarbeit AI Teil 3	8,00
19.08.2024	005	Grundlagenarbeit AI Teil 4	8,00
18.09.2024	003	Setup Azure Entra ID for team member	4,00
18.09.2024	006	Meeting: BPN-KLV Tenant	5,00
19.09.2024	007	Setup Azure DevOps / GIT Repositories	2,00
19.09.2024	008	Exploration Azure AI Services	4,00
20.09.2024	009	Setup PyCharm / Branches / venv / API	0,50
20.09.2024	010	Setup Azure Wiki	0,50
20.09.2024	011	Meeting: Writing the text for the project submission to the ABA portal	3,00
22.09.2024	012	Rewrite parts of the ABA submission	1,50
22.09.2024	013	Image analysis class	3,00
24.09.2024	014	Debugging OpenAI Bildanalyse	4,00
30.09.2024	015	Setup MySQL DB in Azure	2,50
05.10.2024	016	Code: main.py (API routes / API key validation)	2,00
05.10.2024	017	Setup AZURE VM / NGINX / SSL / Backend Repo	7,00
06.10.2024	018	Exploration Azure Subrepos	5,00
07.10.2024	019	System plan in Lucid Charts	3,00
11.10.2024	020	Pentest: DDoS Attack	6,00
11.10.2024	021	Meeting: Milestone presentation	3,50
12.10.2024	022	Exploration SQL Alchemy	3,00
13.10.2024	023	Setup Azure AI + API endpoint + Insurance Chat	4,00
15.10.2024	024	Testing API / Chat	6,00
19.10.2024	025	Full migration to Azure AI	4,50
25.10.2024	026	API autostart and restart functionality	2,00
08.11.2024	027	Server Frontend	6,00
12.11.2024	028	Implement AI full text report	3,00
18.11.2024	029	Poster / Website	3,00
25.11.2024	030	DB Endpoints	2,00
25.11.2024	031	Deploy subrepos on the server	6,00
30.11.2024	032	Deploy Admin Page	3,00
01.12.2024	033	Meeting Prototype / Things to do	2,00
15.12.2024	034	Subrepo Bedingungen + Admin page hosten	4,00
16.12.2024	035	Debugging Admin portal / Change server config	4,00
05.01.2025	036	EXIF Meeting	2,50
12.01.2025	037	EXIF Deployment / Images Deployment	4,00
27.01.2025	038	Server Managment (repo updates)	2,00
30.01.2025	039	Setup PineCone VectorDB (Azure)	5,50
30.01.2025	040	Setup Embedding Model (Azure) / Testing	5,00
31.01.2025	041	Server Managment (repo updates)	3,00
31.01.2025	042	Implement Blob <-> Image Agent	12,00
01.02.2025	043	Debugging Bildanalyse	5,50
02.02.2025	044	Server arbeiten, Frontend multi calls	8,00

Datum	Task	Beschreibung	Dauer in h
05.02.2025	045	Research Azure OpenAI Region Deployments	4,00
14.02.2025	046	Umbau Subrepos -> Singelrepo	6,00
16.02.2025	047	Iteration 3 ppx	2,00
02.03.2025	048	Schreiben der Diplomarbeit	8,00
03.03.2025	049	Schreiben der Diplomarbeit (Teams Meeting)	8,00
04.03.2025	050	Schreiben Diplomarbeit	5,00
05.03.2025	051	Schreiben Diplomarbeit	5,00
09.03.2025	052	Kostenanalyse Diplomarbeit	2,00
09.03.2025	053	Überarbeiten Abbildungen / Zitate	3,00
09.03.2025	054	Final DA Meeting / Überarbeiten der DA	8,00
<b>Gesamtarbeitszeit (Ist)</b>			<b>239,50</b>
Soll			160,00
Offen (Ist-Soll)			+79,50

Tabelle 6.3 Arbeitszeitnachweis von Michael Griesser

### 6.7.3. Gordon Telsnig

Datum	Task	Beschreibung	Dauer in h
03.03.2024	001	Projektbesprechung mit Projektteam KLV und BPN	2,00
27.06.2024	002	Besuch bei KLV zur Bedarfsanalyse für das Frontend	6,50
04.09.2024	003	Teamsmeeting mit Projektteam (Wie geht's weiter?)	1,00
05.09.2024	004	Teamsmeeting mit KLV SISO(Ing. Harald Klary) und Projektteam	0,50
09.09.2024	005	Recherche-Unterschiede Frameworks vs. Vanillia Javascript	2,00
10.09.2024	006	Recherche-Vorteile/Nachteile von Frameworks	2,00
11.09.2024	007	Recherche-Verschiedener Frameworks (React,Angular, Ember,..)	4,00
12.09.2024	008	Recherche-Verschiedener Lizenzen	2,00
13.09.2024	009	Recherche-verschiedener EXIF-Daten-Reader	4,00
13.09.2024	010	Recherche-Sicherheitsaspekte verschiedener Frameworks	3,00
14.09.2024	011	Recherche von Lizenzen verschiedener Frameworks	1,50
14.09.2024	012	G.u.M- Kurzbeschreibung und Aufgabenstellung	1,50
15.09.2024	013	G.u.M- Frameworks vs Vanillia Javascript, Vorteile und Nachteile v. Fw.	3,00
15.09.2024	014	G.u.M- Vergleiche von Frameworks	2,00
16.09.2024	015	G.u.M- Merkmale der Sicherheitsaspekte	2,00
16.09.2024	016	G.u.M- React,Angular, Vue (Vorteile,Nachteile,...)	3,00
17.09.2024	017	G.u.M- Svelte, Ember und Konklusion	2,50
17.09.2024	018	G.u.M- Exifdaten Reader Aufbau und Funktion	2,00
18.09.2024	019	G.u.M- Vergleich und Auswahl Exifreader	2,00
19.09.2024	020	Recherche-Vergleich Typescript vs Proptypes(Typsicherheit)	2,00
20.09.2024	021	Teamsmeeting mit Projektteam (Azure Devops)	2,00
21.09.2024	022	Aufsetzten von Projekt, Gitignore, Git-> Remote Work	2,00
21.09.2024	023	MS-Azure DevOps Probleme, Anlegen Ordnerstruktur, 1. Komponente	3,00
21.09.2024	024	Globale CSS Variablen, weitere Komponente erstellt(Footer)	1,50
22.09.2024	025	Implementierung Navbar(Logo, Icons,Buttons)	7,00
22.09.2024	026	Aufteilung Navbar in kleinere Komponenten	5,00
25.09.2024	027	Recherche-Globale CSS Variablen und Ihre Vorteile	1,00
25.09.2024	028	Implementierung KLV Schriftdesigns und Globale Variablen	5,00
24.09.2024	029	Recherche-React Hooks	5,00
25.09.2024	030	Implementierung Inputfield, Loadbutton,SectionButton in Navbar	6,00
26.09.2024	031	Implementierung einer Container-Komponente zur Datenanzeige	8,00
27.09.2024	032	Implementierung einer Headerbar für Container	2,00
27.09.2024	033	Implementierung einer Togglefunktion für Container	2,00
27.09.2024	034	Implementierung einer Tooltipfunktion	4,00
30.09.2024	035	Recherche-React Context API	2,00
01.10.2024	036	Implementierung einer globalen Tooltip-Logik	6,00
02.10.2024	037	Überarbeitung Container-Komponente	3,00
03.10.2024	038	DatenItem-Komponente und verknüpfung mit Tooltiplogik	5,00
04.10.2024	039	Implementierung einer Datenliste die wiederverwendbar ist	6,00
05.10.2024	040	Recherche Imageslider	2,00
06.10.2024	041	Implementierung Imageslider	6,00
06.10.2024	042	Implementierung Fullscreenmodus für Imageslider und bugfix	4,00
06.10.2024	043	Implementierung Thumbnailtray für Imageslider und CSS-Design	2,50
09.10.2024	044	Implementierung Welcomescreen mit valdierung des Inputs	4,00
10.10.2024	045	Refactor App.tsx und debug für Datenweitergabe an Section1.tsx	3,50
11.10.2024	046	Implementierung eines Tabelle in Section2 für Bild-Metadaten	4,00
12.10.2024	047	Implementierung CoverageVerificationContext	3,00

Datum	Task	Beschreibung	Dauer in h
13.10.2024	048	Teamsmeeting mit Projektteam(Präsentation 1.Meilenstein)	2,00
18.10.2024	049	Performance-Optimierung in Container.tsx	2,00
19.10.2024	050	Implementierung getClass in exifUtils für Datumscheck	4,00
20.10.2024	051	Integration globaler ApiStatusType in Context	5,00
25.10.2024	052	Ausbau ExifDataContainer mit Sortierfunktionen	2,00
26.10.2024	053	Anlegen ProtectedRoute und Routing-Refactor	5,50
27.10.2024	054	Implementierung Section3.tsx Ki Section-Container	3,00
01.11.2024	055	Erstellung und implementierung von Reports für Ki als Liste	4,00
02.11.2024	056	Erstellung und implementierung von Reports für Ki mit Markdown	5,50
03.11.2024	057	Recherche React.Context	8,00
08.11.2024	058	Migration diverser Hilfsfunktionen in formatUtils	4,00
09.11.2024	059	Debugging im ExifHeaderCheckboxes	5,00
15.11.2024	060	Refactoring: ExifHeaderCheckboxes verkürzt	5,50
16.11.2024	061	Style-Optimierung: Tabellenlayout in Section2.css	1,00
17.11.2024	062	Bugfix: Doppelte setState-Aufrufe in Section2.tsx entfernt	5,00
22.11.2024	063	Performance: useCallback im SHandler von Section2 eingefügt	2,50
23.11.2024	064	Code-Cleanup: Unnötige Logs in Section2.tsx entfernt	1,00
24.11.2024	065	Refactoring: handleCheckboxToggle in Section2 gekürzt	1,50
30.11.2024	066	Style-Optimierung: Hover-Effekt in ExifDataContainer angepasst	2,50
01.12.2024	067	Bugfix: Falsche Key-Namen in Section2ContainerProps korrigiert	2,50
06.12.2024	068	Performance: useMemo für Exif-Schlüssel in Section2.tsx	2,50
07.12.2024	069	Code-Cleanup: Variablennamen in Section2 vereinheitlicht	2,00
08.12.2024	070	Performance: ImageSliderContainerSection3 lazy loading	5,00
13.12.2024	071	Bugfix: allDocumentsError in Section3 richtig abgefangen	4,00
14.12.2024	072	Style-Optimierung: custom-header-actions Section3.css responsive	4,00
15.12.2024	073	Performance: computeImageDataList in Section3 optimiert	4,00
20.12.2024	074	Refactoring: DataFetcher mit BasedataFetcher	5,00
21.12.2024	075	Recherche React.Memo	2,00
22.12.2024	076	Bugfix: PDF-Viewer Implemented	3,00
23.12.2024	077	Style-Optimierung: Buttons in ImageSliderContainerSection3	2,00
27.12.2024	078	Refactoring: AnimatedButton an Serverresponse geknüpft	3,00
28.12.2024	079	Performance: React.Memo in div. Klassen adaptiert	4,00
29.12.2024	080	Bugfix: ToggleTooltip-Aufrufe in Section3 korrigiert	3,00
30.12.2024	081	Code-Cleanup: KeyLabels überarbeitet	5,00
02.01.2025	082	Refactoring: 3 Endpunkte auf Azure API umgestellt	5,00
03.01.2025	083	Style-Optimierung: Margin in Section3.css vereinheitlicht	1,00
03.01.2025	084	Performance: Laden von PDFs Section3 per lazy loading und bugfix	8,00
03.01.2025	085	Refactoring: Singletonpattern für API Calls	4,00
04.01.2025	086	Implementierung Container Section4 und ReportSections	6,00
04.01.2025	087	Implementierung Endpunkt für Chat und design Chatcontainer	4,50
04.01.2025	088	Implementierung ReportSections mit Checkbox in Section4	3,50
05.01.2025	089	Performance: useMemo für docTypeToEndpoint in Section3   2,0	1,00
05.01.2025	090	Refactoring: DocumentsContainer und ChatContainer	5,00
05.01.2025	091	CoverageResult und Summary als Markdown in Section4 angezeigt	4,00
10.01.2025	092	Style-Optimierung: Chat container	2,00
10.01.2025	093	Refactoring: Alle Container in eigene Klassen strukturiert	4,00
10.01.2025	094	Implementierung Endpunkte für Dokumente, Mails und Rechnungen	4,00
11.01.2025	095	Implementierung Endpunkte für Chat und Claimcoverage	3,00
11.01.2025	096	Implementierung: API-Call tracking im "GlobalContext"	3,00
11.01.2025	097	Bugfix: API Calls	4,00
12.01.2025	098	Recherche ReactPortal	2,00

Datum	Task	Beschreibung	Dauer in h
12.01.2025	099	Refactoring: Hilfsklassen exif und formatUtils	2,00
12.01.2025	100	Implementierung Neue Buttons für CoverageCheck	2,00
12.01.2025	101	Style-Optimierung: Section4.css   1,0	1,00
17.01.2025	102	Implementierung React.reducer für Dokumente	4,00
17.01.2025	103	G.u.M- Konklusion Exifreader und Tabelle	1,00
18.01.2025	104	G.u.M- Realisierung und Implementierung Section1 und Section2	2,00
18.01.2025	105	G.u.M- Realisierung und Implementierung Section3 und Section4	2,00
18.01.2025	106	G.u.M- Routing und Zugriffsschutz und Globale Context	1,00
25.01.2025	107	G.u.M- React Hooks	4,00
26.01.2025	108	G.u.M- React Memo	4,00
31.01.2025	109	G.u.M- Realisierung Exifreader JSON	4,00
01.02.2025	110	G.u.M- React Context	4,00
02.02.2025	111	G.u.M- Erstellung von Diagrammen	4,00
14.02.2025	112	G.u.M- Formatierung überarbeiten	4,00
15.02.2025	113	G.u.M- Verzeichnisse aktualisieren	2,00
16.02.2025	114	G.u.M- Küberarbeiten der Diplomarbeit	4,00
28.02.2025	115	G.u.M- Kapitel Projektmanagement ausarbeiten	4,00
09.03.2025	116	G.u.M- Finale lesen der Diplomarbeit	3,00
<b>Gesamtarbeitszeit (Ist)</b>			<b>389,00</b>
Soll			160,00
Offen (Ist-Soll)			+229,00

Tabelle 6.4 Arbeitszeitnachweis von Gordon Telsnig

## 7. Literaturverzeichnis

- [1] G. Cullen, „Ankündigung der Pinecone-Integration in Microsoft Azure, mit Leistungs- und Compliance-Vorteilen,“ Pinecone.io, 13 Juli 2023. [Online]. Available: <https://www.pinecone.io/blog/azure/>. [Zugriff am 15 Februar 2025].
- [2] E. Avidon, „Pinecone launches serverless vector database on Azure, GCP,“ Techtarget, 27 August 2024. [Online]. Available: <https://www.techtarget.com/searchdatamanagement/news/366609193/Pinecone-launches-serverless-vector-database-on-Azure-GCP>. [Zugriff am 15 Februar 2025].
- [3] PeerSpot, „Microsoft Azure Cosmos DB vs Pinecone comparison,“ PeerSpot, Jänner 2025. [Online]. Available: [https://www.peerspot.com/products/comparisons/microsoft-azure-cosmos-db\\_vs\\_pinecone](https://www.peerspot.com/products/comparisons/microsoft-azure-cosmos-db_vs_pinecone). [Zugriff am 15 Februar 2025].
- [4] V. Gomez, „Sixfold's Transformation of Insurance Underwriting with Pinecone,“ Pinecone.io, 5 April 2024. [Online]. Available: <https://www.pinecone.io/blog/sixfold/>. [Zugriff am 15 Februar 2025].
- [5] Educational\_Cup9809, „Azure Search vs. Pinecone?,“ Reddit.com, 2024. [Online]. Available: [https://www.reddit.com/r/LangChain/comments/17mxe9t/azure\\_search\\_vs\\_pinecone/](https://www.reddit.com/r/LangChain/comments/17mxe9t/azure_search_vs_pinecone/). [Zugriff am 15 Februar 2025].
- [6] „macht-der-proteine-fortschritte-in-der-proteinfaltung-durch-ki,“ 6 3 2025. [Online]. Available: <https://e-fundresearch.com/newscenter/106-pictet-asset-management/artikel/48727-macht-der-proteine-fortschritte-in-der-proteinfaltung-durch-ki>.
- [7] Soxes, „kuenstliche-intelligenz-beispiele,“ 6 3 2025. [Online]. Available: <https://soxes.ch/kuenstliche-intelligenz-beispiele/>.
- [8] OpenAI, „OpenAI,“ OpenAI, [Online]. Available: <https://platform.openai.com/docs/models/gpt-4o>. [Zugriff am 8 3 2025].
- [9] A. PBC, „claude-3-5-sonnet,“ 8 3 2025. [Online]. Available: <https://www.anthropic.com/news/claude-3-5-sonnet>.
- [10] A. PBC, 8 3 2025. [Online]. Available: [claude-3-5-sonnet](#).

- 
- [11] Meta, „meta-llama-3-1,“ 8 3 2025. [Online]. Available: <https://ai.meta.com/blog/meta-llama-3-1/>.
- [12] TIGER-AI-Lab, „TIGER-AI-Lab,“ 4 3 2025. [Online]. Available: <https://github.com/TIGER-AI-Lab/MMLU-Pro>.
- [13] A. Analysis, „methodology/intelligence-benchmarking,“ 5 3 2025.. [Online]. Available: <https://artificialanalysis.ai/methodology/intelligence-benchmarking>.
- [14] ArtificialAnalysis.ai, 5 3 2025. [Online]. Available: <https://artificialanalysis.ai/models?models=gpt-4o%2Cgpt-4o-mini%2Cllama-3-1-instruct-70b%2Cclaude-35-sonnet>.
- [15] Help-OpenAI, „4936856-what-are-tokens-and-how-to-count-them,“ 8 3 2025. [Online]. Available: <https://help.openai.com/en/articles/4936856-what-are-tokens-and-how-to-count-them>.
- [16] Wikipedia, „Inference\_engine,“ 6 3 2025. [Online]. Available: [https://en.wikipedia.org/wiki/Inference\\_engine](https://en.wikipedia.org/wiki/Inference_engine).
- [17] S. M. Computer, „Success\_Story\_xAI\_Colossus\_Cluster.pdf,“ 3 3 2025. [Online]. Available: [https://www.supermicro.com/CaseStudies/Success\\_Story\\_xAI\\_Colossus\\_Cluster.pdf](https://www.supermicro.com/CaseStudies/Success_Story_xAI_Colossus_Cluster.pdf).
- [18] Microsoft, 8 3 2025. [Online]. Available: <https://learn.microsoft.com/de-de/legal/cognitive-services/openai/data-privacy?context=%2Fazure%2Fai-services%2Fopenai%2Fcontext%2Fcontext&tabs=azure-portal>.
- [19] P. Informatik, „Was ist ein Framework?,“ [Online]. Available: <https://www.pi-informatik.berlin/pi-lexikon/softwareentwicklung/was-ist-ein-framework/>. [Zugriff am September 2024].
- [20] J. Patel, „List of 10 Best Front end Frameworks to Use For Web Development,“ [Online]. Available: <https://www.monocubed.com/blog/best-front-end-frameworks/>. [Zugriff am September 2024].
- [21] I. Redaktion, „Webframeworks – ein Überblick,“ Oktober 2022. [Online]. Available: <https://www.ionos.at/digitalguide/websites/web-entwicklung/webframeworks-ein-ueberblick/>. [Zugriff am September 2024].
- [22] React, „React Official Documentation – Getting Started,“ [Online]. Available: <https://reactjs.org/docs/getting-started.html>. [Zugriff am 13 Februar 2025].

- [23] Angular, „Angular Official Documentation – Angular Architecture,“ [Online]. Available: <https://angular.io/guide/architecture>. [Zugriff am 14 Februar 2024].
- [24] Vue.js, „Vue.js Official Documentation – Components,“ [Online]. Available: <https://vuejs.org/v2/guide/components.html>. [Zugriff am 14 Februar 2025].
- [25] S. D. Lead, „Comparison: Top Frontend Frameworks For Web Development in 2024,“ [Online]. Available: <https://softdevlead.com/comparison-top-frontend-frameworks-for-web-development-in-2024>. [Zugriff am September 2024].
- [26] dotnetpro, „React, Angular und Vue.js: Eine Gegenüberstellung von Frontend-Frameworks,“ [Online]. Available: <https://www.dotnetpro.de/frontend/user-interface/react-angular-vuejs-gegenueberstellung-frontend-frameworks-2910346.html>. [Zugriff am September 2024].
- [27] S. Maintainers, „Introduction to Svelte.“ Svelte Official Documentation,“ [Online]. Available: <https://svelte.dev/docs>. [Zugriff am 15 September 2024].
- [28] E. C. Team, „Guides and Tutorials.“ Ember Official Documentation,“ [Online]. Available: <https://guides.emberjs.com/release/>. [Zugriff am 14 September 2024].
- [29] S. Marnach, „PyExifTool - PyPI,“ Python Software Foundation, [Online]. Available: <https://pypi.org/project/PyExifTool/>. [Zugriff am September 2024].
- [30] I. Redaktion, „Was sind EXIF-Daten?,“ [Online]. Available: <https://www.ionos.de/digitalguide/websites/webdesign/was-sind-exif-daten/>. [Zugriff am 2024 September].
- [31] Wikipedia-Autoren, „Exif,“ [Online]. Available: <https://en.wikipedia.org/wiki/Exif>. [Zugriff am September 2024].
- [32] Foto-Tipps.com, „EXIF-Daten der Kamera auswerten,“ [Online]. Available: <https://www.foto-tipps.com/grundlagen/exif-daten/>. [Zugriff am September 2024].
- [33] J. E. a. I. T. I. Association, „www.kodak.com,“ [Online]. Available: <https://www.kodak.com/global/plugins/acrobat/en/service/digCam/exifStandard2.pdf>. [Zugriff am 5 März 2025].
- [34] M. Haltenhof, „Foto Metadaten auslesen (EXIF-Daten),“ Matthias Haltenhof Fotografie Blog, [Online]. Available: [https://www.matthiashaltenhof.de/blog/foto-metadaten-auslesen/#Standardisierte\\_Metadaten\\_EXIF\\_IPTC\\_XMP](https://www.matthiashaltenhof.de/blog/foto-metadaten-auslesen/#Standardisierte_Metadaten_EXIF_IPTC_XMP). [Zugriff am September 2024].

- [35] T. P. Contributors, „pillow.readthedocs,“ o. J.. [Online]. Available: <https://pillow.readthedocs.io>. [Zugriff am 25.9.2025].
- [36] M. Kovařík, „Exifr,“ Mutiny.cz, [Online]. Available: <https://mutiny.cz/exifr/>. [Zugriff am September 2024].
- [37] A. Ingelheim, „Datenschutz bei Versicherungen: Was sieht die DSGVO bei Versicherungen vor?,“ Datenschutzexperte.de, 2. Juli 2021. [Online]. Available: <https://www.datenschutzexperte.de/blog/datenschutz-bei-versicherungen-was-sieht-die-dsgvo-bei-versicherungen-vor>. [Zugriff am 15. Februar 2025].
- [38] J. Craft, „Anonymized Test Data a Must for Database Migrations,“ Privacydynamics.io, 3. Jänner 2024. [Online]. Available: <https://www.privacydynamics.io/post/anonymized-test-data-a-must-for-database-migrations/>. [Zugriff am 15. Februar 2025].
- [39] Deutsche Gesellschaft für Datenschutz, „Auswirkungen der DSGVO auf die Versicherungsbranche,“ [Online]. Available: [https://dg-datenschutz.de/auswirkungen\\_der\\_dsgvo\\_auf\\_die\\_versicherungsbranche](https://dg-datenschutz.de/auswirkungen_der_dsgvo_auf_die_versicherungsbranche). [Zugriff am 15. Februar 2025].
- [40] J. Niestroj, „Bitkom.org,“ 2022. [Online]. Available: <https://www.bitkom.org/sites/main/files/2022-12/2202212BitkomTrendreportInsurance.pdf>. [Zugriff am 15. Februar 2025].
- [41] R. Karg, „Schadenmanagement: Zwei Trends unter der Lupe,“ Convista, 3. Juni 2024. [Online]. Available: <https://www.convista.com/impulse/branche/versicherungswirtschaft/schadenmanagement-trends-unter-der-lupe/>. [Zugriff am 15. Februar 2025].
- [42] sprout.ai, „sprout.ai,“ sprout.ai, 2025. [Online]. Available: <https://sprout.ai/>. [Zugriff am 15. Februar 2025].
- [43] capgemini.com, „capgemini.com,“ 2025. [Online]. Available: <https://www.capgemini.com/de-de/wp-content/uploads/sites/5/2021/06/Broschuere-Schadenabwicklung-mit-KI-gestuetzer-Automatisierung-effektiv-und-effizient-gestalten-.pdf>. [Zugriff am 15. Februar 2025].
- [44] R. Schwaber-Cohen, „What is a Vector Database & How Does it Work? Use Cases + Examples,“ Pinecone.io, 3. Mai 2023. [Online]. Available: <https://www.pinecone.io/learn/vector-database/>. [Zugriff am 15. Februar 2025].

- [45] Microsoft, 5 3 2025. [Online]. Available: <https://learn.microsoft.com/en-gb/azure/update-manager/overview?tabs=azure-vms>.
- [46] G. D. DL, „comparison-of-flask-django-and-fastapi-advantages-disadvantages-and-use-cases-63e7c692382a,“ 7 3 2025. [Online]. Available: <https://medium.com/@tubelwj/comparison-of-flask-django-and-fastapi-advantages-disadvantages-and-use-cases-63e7c692382a>.
- [47] FastAPI, „fastapi.tiangolo.com,“ 5 3 2025. [Online]. Available: <https://fastapi.tiangolo.com/tutorial/cors/>.
- [48] brightonpanelworks, „wp-content/uploads/2019/10/auto-3734396\_1280.jpg,“ 5 3 2025. [Online]. Available: [https://brightonpanelworks.com.au/wp-content/uploads/2019/10/auto-3734396\\_1280.jpg](https://brightonpanelworks.com.au/wp-content/uploads/2019/10/auto-3734396_1280.jpg).
- [49] React, „React-Router,“ [Online]. Available: <https://reactrouter.com/start/framework/routing#configuring-routes>. [Zugriff am 14 Februar 2025].
- [50] R. Team, „React – A JavaScript library for building user interfaces,“ [Online]. Available: <https://reactjs.org/docs/context.html>. [Zugriff am 11 Jänner 2025].
- [51] R. Team, „React-Hooks,“ [Online]. Available: <https://legacy.reactjs.org/docs/hooks-intro.html>. [Zugriff am 11 Jänner 2025].
- [52] React, „React-Reducer,“ [Online]. Available: <https://react.dev/reference/react/useReducer>. [Zugriff am 25 3 3].
- [53] R. Team, „React-Memo,“ [Online]. Available: <https://legacy.reactjs.org/docs/react-api.html#reactmemo>. [Zugriff am 11 Jänner 2025].
- [54] Microsoft, „TypeScript,“ [Online]. Available: <https://www.typescriptlang.org/docs/handbook/2/narrowing.html>.
- [55] Matoba, „Piexif Dokumentation,“ Read the Docs, 2018. [Online]. Available: <https://piexif.readthedocs.io/en/latest/about.html#license>. [Zugriff am 9 September 2024].
- [56] Flatirons, „FastAPI vs Flask: Choosing the Best Python Web Framework,“ Flatirons, 9 Juni 2024. [Online]. Available: <https://flatirons.com/blog/fastapi-vs-flask/>. [Zugriff am 15 Februar 2025].

- [57] Microsoft.com, „What is Azure OpenAI Service?“, Microsoft.com, 30 Jänner 2025. [Online]. Available: <https://learn.microsoft.com/en-us/azure/ai-services/openai/overview>. [Zugriff am 15 Februar 2025].
- [58] Microsoft, „Data, privacy, and security for Azure OpenAI Service“, microsoft.com, 18 Dezember 2024. [Online]. Available: <https://learn.microsoft.com/en-us/legal/cognitive-services/openai/data-privacy?tabs=azure-portal>. [Zugriff am 15 Februar 2025].
- [59] D. Noakes, „MetadataExtractor Javadoc“, Javadoc.io, [Online]. Available: <https://javadoc.io/doc/com.drewnoakes/metadata-extractor/2.16.0/com/drew/imaging/ImageMetadataReader.html>. [Zugriff am September 2024].
- [60] X. BV, „Exiftool - GitHub“, GitHub, [Online]. Available: <https://github.com/xillio/exiftool>. [Zugriff am September 2024].
- [61] GeoSolutions, „ImageIO-EXT - GitHub“, GitHub, [Online]. Available: <https://github.com/geosolutions-it/imageio-ext>. [Zugriff am September 2024].
- [62] O. Ozcitak, „ExifLibrary for .NET“, [Online]. Available: <https://www.codeproject.com/Articles/43665/ExifLibrary-for-NET>. [Zugriff am September 2024].
- [63] I. Cloud, „Top 10 best front end frameworks in 2025“, [Online]. Available: <https://www.imaginarycloud.com/blog/best-frontend-frameworks>. [Zugriff am September 2024].
- [64] H. RegioConcept, „Was ist ein Framework? Einfach erklärt“, [Online]. Available: <https://www.heise-regioconcept.de/glossar/framework>. [Zugriff am September 2024].
- [65] O. Foundation, „Cross Site Scripting Prevention Cheat Sheet“, [Online]. Available: <https://owasp.org/www-community/attacks/xss/>. [Zugriff am 30.09.2024].
- [66] Meta, „dangerouslySetInnerHTML – React Documentation.“ React Official Documentation, [Online]. Available: <https://react.dev/reference/react-dom/components/common#dangerously-setting-the-inner-html>. [Zugriff am 30. September 2024].

## 8. Abbildungsverzeichnis

Abbildung 1.1 Systemstruktur .....	10
Abbildung 2.1 Intelligenz Benchmark [14] .....	17
Abbildung 2.2 Kontextlänge [14].....	18
Abbildung 2.3 Ausgabegeschwindigkeit [14] .....	19
Abbildung 2.4 Intelligenz vs. Preis [14] .....	19
Abbildung 3.1 Datenfluss der Deckungsprüfung .....	35
Abbildung 3.2 UML-Diagramm viva_regelwerke .....	38
Abbildung 3.3 Frontend zur Wartung der viva_regelwerke Datenbank.....	39
Abbildung 3.4 Wichtige API-Endpunkte .....	41
Abbildung 3.5 Sequenzdiagramm des Endpunktes „/claimimage/{claim_id}“ .....	45
Abbildung 4.1 Klassendiagramm Bildanalyse.....	62
Abbildung 4.2 Sequenzdiagramm Bildanalyse .....	67
Abbildung 4.3 Beispielbild [55] .....	68
Abbildung 5.1 Flussdiagramm-Frontend.....	73
Abbildung 5.2 Welcomescreen .....	74
Abbildung 5.3 Section1-Übersicht.....	75
Abbildung 5.4 Section2-Exifanalyse .....	75
Abbildung 5.5 Section3-KiAnalyse.....	78
Abbildung 5.6 Section4-Deckungsprüfung.....	78
Abbildung 5.8 useReducerHook(); .....	84
Abbildung 5.9 React.Memo.....	86
Abbildung 6.1 Epics und Issues Griesser .....	93
Abbildung 6.2 Epics und Issues Di Bernardo 1 .....	94
Abbildung 6.3 Epics und Issues Di Bernardo 2 .....	95
Abbildung 6.4 Epics und Issues Di Bernardo 3 .....	96
Abbildung 6.5 Epics und Issues Telsnig 1 .....	97
Abbildung 6.6 Epics und Issues Telsnig 2 .....	98
Abbildung 6.7 Epics und Issues Telsnig 3 .....	99
Abbildung 6.8 monatliche Kosten der App (Entwicklung) .....	103
Abbildung 6.9 Besprechungsprotokoll 14.10.2024 Seite 1 .....	104
Abbildung 6.10 Besprechungsprotokoll 14.10.2024 Seite 2 .....	105
Abbildung 6.11 Besprechungsprotokoll 18.11.2024 Seite 1 .....	106
Abbildung 6.12 Besprechungsprotokoll 18.11.2024 Seite 2 .....	107
Abbildung 6.13 Besprechungsprotokoll 17.02.2025 Seite 1 .....	108
Abbildung 6.14 Besprechungsprotokoll 17.02.2025 Seite 2 .....	109
Abbildung 6.15 Besprechungsprotokoll 10.03.2025 Seite 1 .....	110
Abbildung 6.16 Besprechungsprotokoll 10.03.2025 Seite 2 .....	111

## 9. Tabellenverzeichnis

Tabelle 2.1 Frameworks .....	29
Tabelle 6.1 Kostenaufstellung .....	103
Tabelle 6.2 Arbeitszeitnachweis von Alexander Di Bernardo .....	114
Tabelle 6.3 Arbeitszeitnachweis von Michael Griesser .....	116
Tabelle 6.4 Arbeitszeitnachweis von Gordon Telsnig .....	119

## 10. Listings

Listing 3.1 JSON-Response der Deckungsprüfung.....	44
Listing 4.1 API-Key Validierung.....	60
Listing 4.2 Status Endpunkt mit API Key.....	61
Listing 4.3 Base64 Codierung .....	63
Listing 4.4 Klasse Bildanalyse Konstruktor .....	64
Listing 4.5 OpenAI SDK mit Bild URL .....	65
Listing 4.6 Parallele Bildanalyse .....	66
Listing 4.7 Prompt JSON Template .....	66
Listing 5.1 JSON-EXifdaten .....	77
Listing 5.2 Validierung .....	79
Listing 5.3 Beispiel ProtectedRoute .....	80
Listing 5.4 Beispiel Routes .....	80
Listing 5.5 Beispiel useEffect();.....	83
Listing 5.6 useStateHook();.....	83