SA
Dokumentation

Analyse Intranet Content

Semester: Herbst 2024

Version: 0.1
Date: 2024-12-20
Git Version: 3b23011

Projekt Team: David Stäheli
Cedric Christen

Projektbetreuer: Frank Koch

Fachbereich Informatik
OST – Ostschweizer Fachhochschule

Inhaltsverzeichnis

Teil I

Einleitung

Abstract

Im Rahmen unserer Studienarbeit haben wir ein Tool entwickelt, das Unternehmen dabei unterstützt, Trends und Muster in ihren Daten zu erkennen, relevante Informationen zu vernetzen und datenbasierte Entscheidungen für ein effektives Wissensmanagement und eine fundierte strategische Analyse zu treffen. Solche Analysen sind ohne entsprechende automatische Aufbereitung äusserst zeitaufwändig. In vielen Organisationen sammeln sich über die Zeit Dokumente, oder allgemeiner, Informationen an. Allerdings bleibt oft unklar, welche Themen, Trends oder Schlüsselpersonen über einen bestimmten Zeitraum hinweg relevant sind.
Unser Tool adressiert dieses Problem, indem es Inhalte ausliest, analysiert und visuell aufbereitet. Daten von festgelegten Webseiten wie internen Blogs, Geschäftsberichten oder firmeninternen sozialen Plattformen werden regelmässig extrahiert, geparst und mit Large Language Models (LLMs) analysiert. Die Originaltexte und Analysen werden in einer Datenbank gespeichert, welche als Grundlage für weiterführende Analysen und Auswertungen dient. Die Analyseergebnisse ermöglichen die Auswertung der Entwicklung bestimmter Themenbereiche, etwa die Feststellung, dass in den letzten sechs Monaten die Anzahl der Dokumente im Bereich Künstliche Intelligenz gestiegen ist.
Der Zugriff auf die aufbereiteten Daten erfolgt über eine benutzerfreundliche Webanwendung mit umfangreichen Visualisierungsmöglichkeiten. Nutzer haben die Möglichkeit, mithilfe von Filtern gezielt Themen und Trends zwischen den Dokumenten und Plattformen zu erkunden. Die unterschiedlichen Abfragemöglichkeiten und die grafische Aufbereitung der Daten erlauben es, ein umfassendes Verständnis der Inhalte zu entwickeln, Muster zu erkennen und datenbasierte Entscheidungen zu treffen.

Management Summary

Ausgangslage

Die Aufgabenstellung für das Projekt ist es, ein Tool zu entwickeln, welches die Analyse von Intranet-Inhalten ermöglicht. Um dies im Rahmen der Studienarbeit zu gewährleisten, wurde der Fokus auf den Inhalt von HTML-Seiten gelegt. Das Tool soll es ermöglichen, dass HTML-Seiten von verschiedenen Quellen heruntergeladen, geparst und analysiert werden können. Ein wichtiger Aspekt der Aufgabenstellung ist auch, dass die Inhalte mittels eines LLM analysiert werden sollen und die Klassifizierung der Inhalte in verschiedenen Kategorien erfolgen soll.

Vorgehen

Zur Umsetzung wurde eine modulare Softwarelösung entwickelt, die auf modernen Technologien basiert.

Scraping / Parsing

Mit einer vordefinierten Liste von Plattformen (Webseiten), welche in der MongoDB gespeichert sind, werden die Informationen von den Webseiten durchsucht, herausgelesen, gefiltert und dann in der Datenbank gespeichert.

Analyse

Geparste Daten werden mit dem Analyzer dann über einen API-Endpunkt bei Infomaniak[2] mit dem ausgewählten LLM-Modell geschickt. Dieses analysiert und klassifiziert den Inhalt und das Ergebnis wird in der Datenbank gespeichert, um im nächsten Schritt diese visualisieren zu können.

Visualisierung

Die analysierten Daten werden im Frontend über eine Streamlit-Webapplikation mit verschiedenen Diagrammen in einer Dashboard-Ansicht dargestellt. Standardmässig werden die Daten über einen Zeitraum von sieben Tagen angezeigt und keine Filter gesetzt. Bei einzelnen Diagrammen kann interaktiv die Anzahl Kategorien angepasst werden, wie bei der Abbildung: 1 ersichtlich ist.

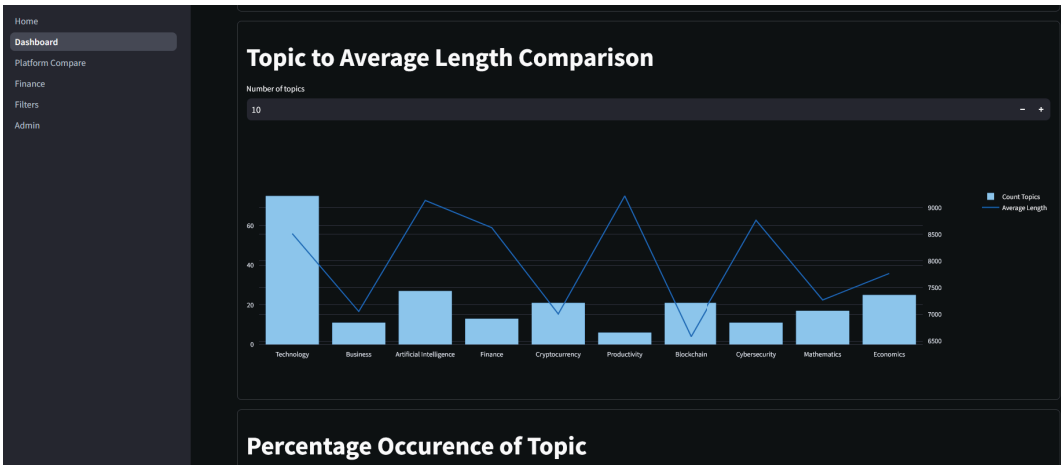


Abbildung 1: Dashboard-Ansicht: Textlängenvergleich nach Topic

Filterung

Verschiedene Filter (Abbildung: 2) können angewendet werden, um gezielte Informationen anzeigen zu lassen. Der Zeitraum, die Kategorie und die Quelle können aktuell gefiltert werden.

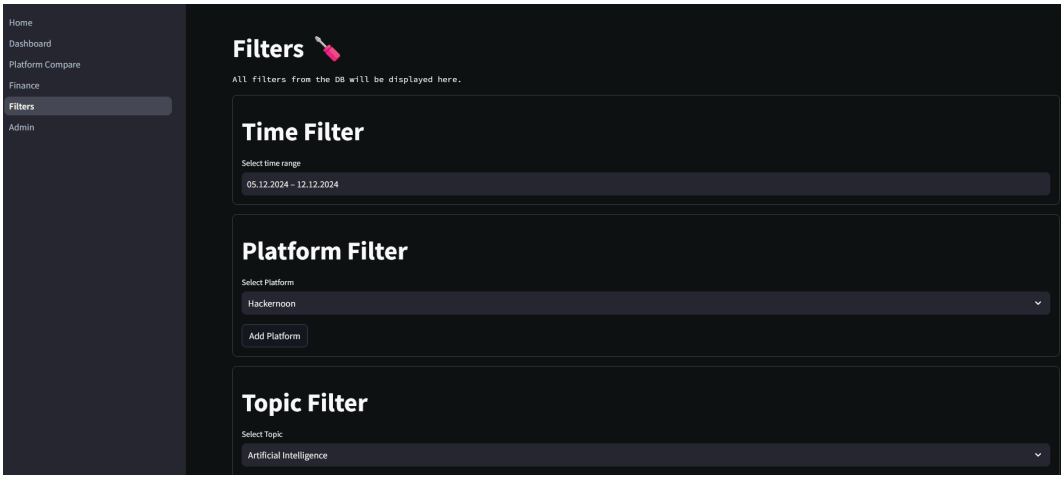


Abbildung 2: Filter Seite

Weitere Tools wie Traefik, Mongo-Express und Portainer wurden für Routing und Monitoring genutzt.

Ergebnisse

Die entwickelte Lösung umfasst ein Dashboard, das Nutzern die Analyseergebnisse interaktiv bereitstellt. Mit Filteroptionen können Inhalte nach Zeitraum, Kategorien oder spezifischen Quellen angezeigt werden. Das Tool ermöglicht es, grosse Datenmengen effizient zu verarbeiten und Entwicklungen wie die Zunahme bestimmter Themenbereiche über Zeit zu erkennen. Für den Einsatz in diesem Fall wurde nach einer detaillierten Evaluierung das LLM-Modell Llama3 ausgewählt, da es die spezifischen Anforderungen der Textanalyse am besten erfüllt. Die Verwendung eines datenbankbasierten NoSQL-Modells ermöglicht eine problemlose Erweiterung der Plattform. Schwachstellen, wie Ladezeiten bei sehr grossen Datenmengen, wurden erkannt und dokumentiert, um in zukünftigen Iterationen adressiert zu werden.

Ausblick

Die aktuelle Lösung bietet ein solides Fundament für zukünftige Erweiterungen. Geplant ist die Unterstützung zusätzlicher Dateiformate wie PDFs und die Möglichkeit, lokale Dateien einzulesen und zu analysieren. Die Einführung eines Benutzerverwaltungssystems könnte die Anwendung auf Mehrbenutzerszenarien erweitern. Zusätzlich könnten die Visualisierungen weiterentwickelt werden, um neue Perspektiven auf die analysierten Inhalte zu ermöglichen. Eine Optimierung der Analysemethodik durch erweiterte Prompts und eine tiefere Integration von Datenquellen könnten den Nutzen des Tools weiter steigern. Langfristig könnte die Anwendung so ausgebaut werden, dass sie nicht nur Daten analysiert, sondern auch Empfehlungen für konkrete Handlungsoptionen gibt.

Aufgabenstellung

Im Rahmen der Studienarbeit, kurz SA, für die Ostschweizer Fachhochschule (OST) wurde die folgende Aufgabenstellung übertragen.
Heutzutage werden sehr viele Informationen im Intranet von Unternehmungen gespeichert. Jedoch ein tieferes Verständnis für diese Informationen ist eher selten der Fall. Beispielsweise, dass in den letzten 6 Monaten mehr Einträge zum Thema AI etc. erstellt wurden.
Dafür soll eine Anwendung entwickelt werden, welche den Inhalt einer Webseite ausliest und mit Hilfe eines LLM kategorisiert und mit anderen Einträgen in Verbindung setzt. Die kategorisierten Einträge sollen wiederum in einem Dashboard dargestellt werden und Filterfunktionen für das individuelle Anzeigen von Einträgen bieten.
Die komplette Aufgabenstellung ist im Anhang13 zu finden.

Teil II

Produkt Dokumentation

Kapitel 1

Anforderungen

1.1 Einführung

Nach der Klärung der Aufgabenstellung beginnt nun die detaillierte Auseinandersetzung mit den zentralen Aspekten des Projekts. Im Fokus stehen die Definition der Anforderungen, die Analyse der relevanten Domäne sowie die Ausarbeitung der technischen Architektur.
Zunächst werden die funktionalen und nicht-funktionalen Anforderungen spezifiziert, um den Rahmen und die Ziele klar zu definieren. Daraufhin folgt die Domainanalyse, die essenzielle Modelle und Prozesse beschreibt, welche die Grundlage für die Softwarearchitektur bilden. Anschliessend wird die geplante Architektur dokumentiert, einschliesslich der verwendeten Technologien, der modularen Aufteilung und der Schnittstellen zwischen den Komponenten.

1.2 Funktionelle Anforderungen

1.2.1 Use Case Diagram

Akteure

- Benutzer:
  - Ziel: Startet Analyseprozess, zeigt Analyseergebnis als Graph an, filtert die anzuzeigenden Daten, sucht nach bestimmten Inhalten
- Infomaniak[2]:
  - Ziel: Analysiert erhaltene Daten gemäss den Prompts. Sendet Resultat zurück.
- Hackernoon[8]:
  - Ziel: Zeigt neueste News Artikel an.

Diagramm

Die Struktur der Use Cases ist in Abbildung 1.1 dargestellt.

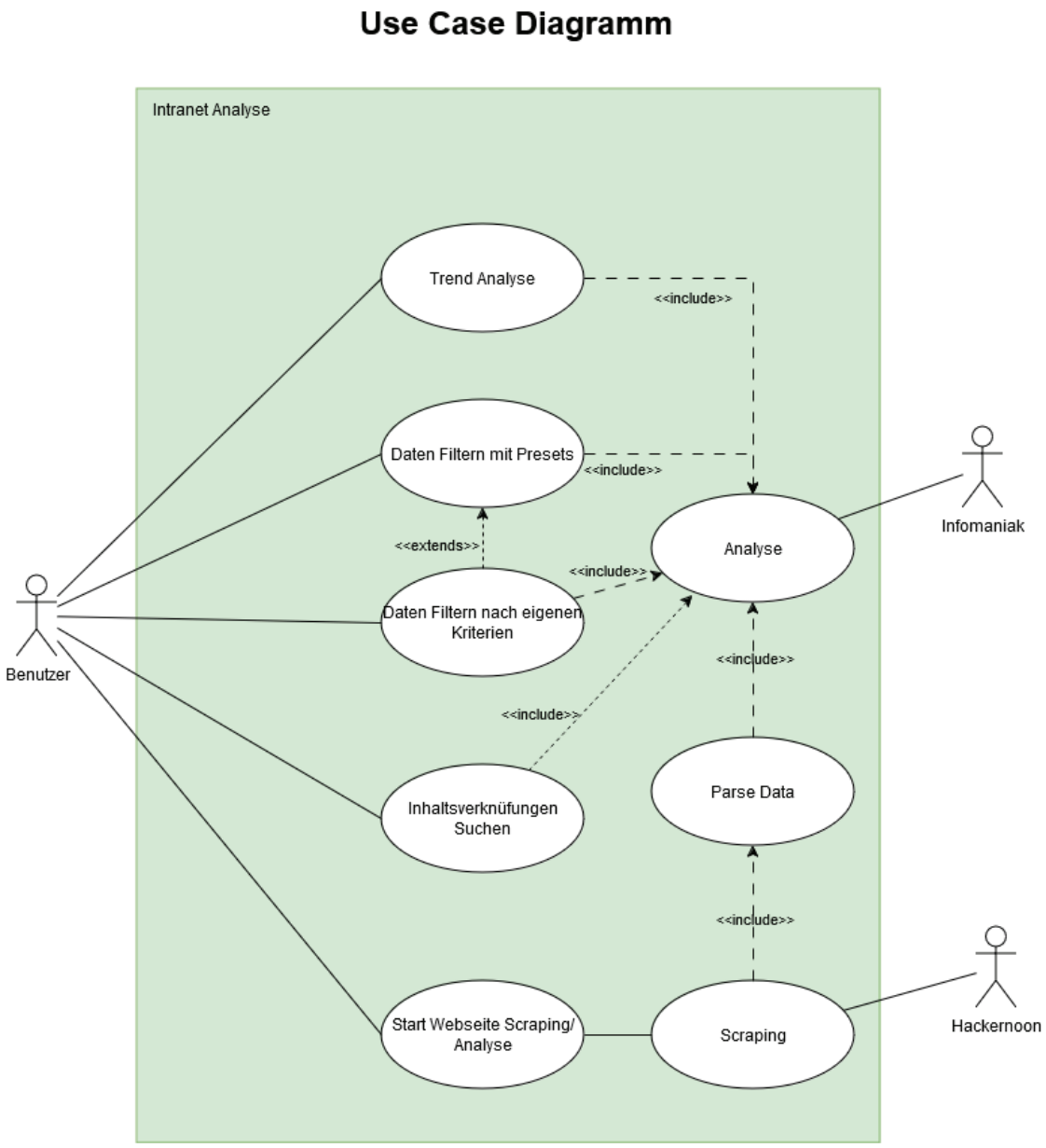


Abbildung 1.1: Use Case Diagramm

1.2.2 Use Case Beschreibung

Der wichtigste Use Case wird in Fully Dressed geschrieben und alle anderen im Casual Format.

Casual Use Cases

- **UC-1 Trend Analyse:** Der Benutzer kann die Trend Daten auf dem Frontend anschauen.
- **UC-2 Daten filtern mit Presets:** Der Benutzer kann die Daten im Frontend mit vorbereiteten Filtern filtern.
- **UC-3 Daten filtern nach eigenen Kriterien:** Der Benutzer kann die Daten im Frontend mit eigenen Kriterien filtern.
- **UC-4 Inhaltsverknüpfungen suchen:** Der Benutzer kann überschneidende Inhalte miteinander verknüpfen und diese analysieren lassen.
- **UC-6 Datenbank anpassen via GUI:** Der Benutzer kann im Admin Bereich mittels GUI Änderungen an den Plattform-Konfigurationen vornehmen.

Fully Dressed Use Cases

ID	UC-5
Name	Start Webseite Scraping/Analyse
Akteur	Benutzer
Beschreibung	Benutzer startet manuell einen Scraping- oder Analyseablauf
Voraussetzung	• Eine neue Webseite wurde hinzugefügt oder Analyseschritt wurde geändert • Benutzer hat Zugriff auf das Backend
Standard-Ablauf	• Benutzer führt Befehl aus, damit alle Scraper ablaufen
Alternativer-Ablauf	Benutzer lässt nur die Analyse laufen
Nachbedingung	Neue Webseite oder alte Webseiten mit neuer Analyse wurden erfolgreich analysiert
Resultat	Neu analysierte Daten sind in der Datenbank gespeichert

Tabelle 1.1: UC-5

Use Case Priorisierung

Die genannten Use Cases sind in der folgenden Reihenfolge priorisiert:

1. UC-5 Start Webseite Scraping/Analyse in der Tabelle: UC-5
2. UC-1 Trend Analyse
3. UC-2 Daten filtern mit Presets
4. UC-3 Daten filtern nach eigenen Kriterien
5. UC-4 Inhaltsverknüpfungen suchen
6. UC-6 Datenbank anpassen via GUI

1.3 Nicht-Funktionale Anforderungen

Die nicht-funktionalen Anforderungen des Projekts sind in den Tabellen 1.2, 1.3, 1.4, 1.5, 1.6, 1.7 und 1.8 definiert und ergeben sich aus der Aufgabenstellung.

ID	NFR-1
Kategorie	Performance Efficiency
Anforderung	Das Backend sollte 1000 Requests pro Minute handeln können.
Messung	Stresstest
Messweg	Stresstest ausführen
Priorisierung	Mittel
Resultat	Durch einen Architekturentscheid ist diese Metrik nicht mehr gültig.

Tabelle 1.2: NFR-1

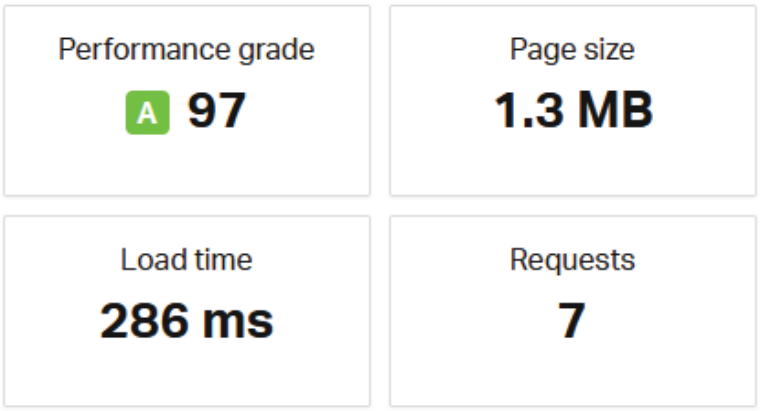
ID	NFR-2
Kategorie	Performance Efficiency
Anforderung	Jede Seite sollte nicht länger als 200ms für das Laden benötigen
Messung	Testing
Messweg	Mit Webtool Zeit testen.
Priorisierung	Mittel
Resultat	 Abbildung 1.2: Screenshot von Pingdom[18] Site Performance Test. Die Ladezeit ist höher als erwünscht, was darauf zurückzuführen ist, dass die einzelnen Diagramme mehr Zeit für das aufbereiten der Daten benötigen. Die Initialladezeit bis zur ersten Anzeige im Browser beträgt unter 200 ms, basierend auf Messungen mit den Chrome-Entwicklertools.

Tabelle 1.3: NFR-2

ID	NFR-3
Kategorie	Usability
Anforderung	Die Seite soll sowohl auf dem Tablet als auch auf dem Desktop gut aussehen
Messung	Auf beiden Geräten manuell testen.
Messweg	Webseite auf Desktop und Tablet anschauen und bewerten.
Priorisierung	Hoch
Resultat	Durch Streamlit sieht die Webseite sowohl auf dem Desktop als auch auf dem Tablet ansprechend aus.

Tabelle 1.4: NFR-3

ID	NFR-4
Kategorie	Interaction Capability
Anforderung	Die Web-Applikation sollte auf Firefox, Chrome und Safari laufen
Messung	Keine unerwarteten Fehler auf den genannten Browsern auftreten.
Messweg	Manuelle Tests auf allen genannten Browsern.
Priorisierung	Hoch
Resultat	Das Frontend funktioniert einwandfrei in Firefox und Chrome, wie durch manuelle Tests evaluiert wurde. In Abstimmung mit dem Industriepartner wurde auf das Testen mit Safari verzichtet.

Tabelle 1.5: NFR-4

ID	NFR-5
Kategorie	Usability
Anforderung	Drei von vier TestUsern sollten das UI (Kategorien: Layout, Responsiveness, Farbe, Inhalt) der Applikation mit einem Tablet mit einer Note von mindestens 8 von 10 bewerten, wobei 10 das Beste ist.
Messung	Mindestens 4 Benutzer testen das UI.
Messweg	Es werden Testbenutzer befragt, welche das UI bewerten sollen.
Priorisierung	Mittel
Resultat	Die Testpersonen bewerteten die Benutzerfreundlichkeit der Anwendung mit einem durchschnittlichen Wert von 8.34, wie der Usability-Test in der Tabelle: 14.1 im Anhang zeigt. Damit gilt die entsprechende Anforderung als erfüllt.

Tabelle 1.6: NFR-5

ID	NFR-6
Kategorie	Performance Efficiency
Anforderung	Die Datenbank soll bis zu 100’000 Seiten managen können.
Messung	Datenbank mit 100’000 Seitendaten füllen.
Messweg	Überprüfen, ob die Datenbank mit der Anzahl Seiten gut klarkommt.
Priorisierung	Hoch
Resultat	Die Seite konnte die 100’000 Seiten ohne Probleme managen. Wir haben einen Load Test durchgeführt, bei dem wir 100’000 analysierte Seiten simuliert haben. Es wurden erheblich längere Ladezeiten festgestellt, jedoch war die Funktionalität nicht eingeschränkt oder defekt.

Tabelle 1.7: NFR-6

ID	NFR-7
Kategorie	Maintainability
Anforderung	Clean Code
Messung	Funktionsgleiche Methoden sollen im Code nicht existieren.
Messweg	Sonar benutzen, um die Code Metrics zu überprüfen
Priorisierung	Mittel
Resultat	Der Ruff[10] Checker überprüft bei jedem Push oder Pull-Request mittels einer GitHub Action den Code, wodurch durchgehend Clean Code eingehalten wird.

Tabelle 1.8: NFR-7

1.3.1 Verifikation von Nicht-Funktionalen Anforderungen

Die meisten nicht-funktionalen Anforderungen (NFAs) werden durchgehend während dem Projekt automatisch überprüft. Die restlichen werden in Woche 40 überprüft.

1.3.2 Resultat

Alle NFAs wurden entweder erfüllt oder nachvollziehbar begründet, warum ihre Umsetzung nicht möglich oder nicht zumutbar ist.

Kapitel 2

Domainanalyse

2.1 Datenbankmodell

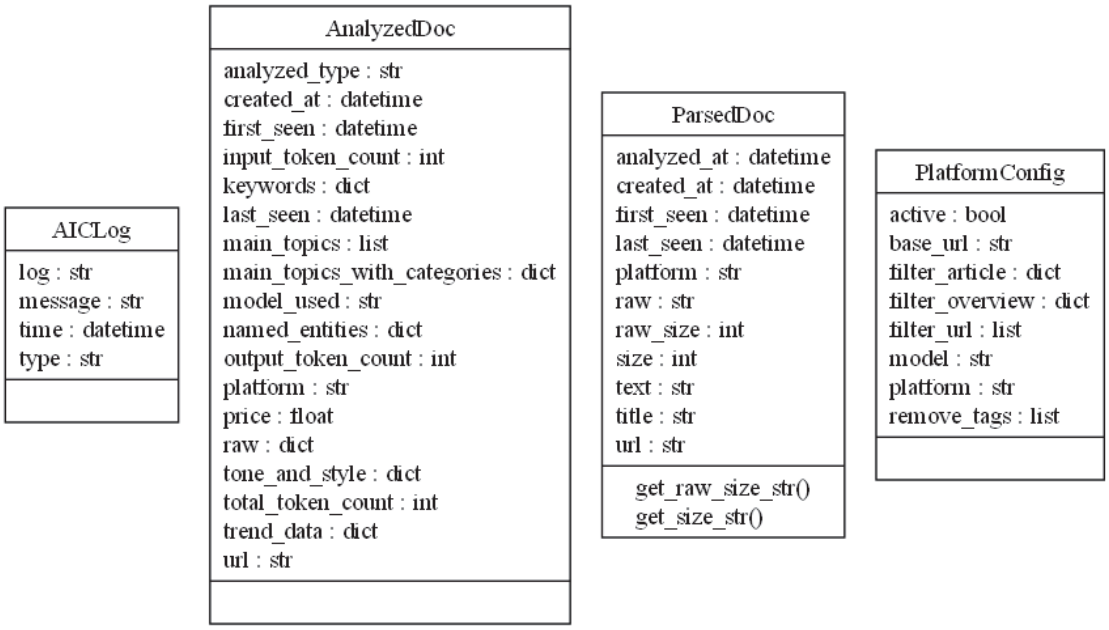


Abbildung 2.1: Datenbankmodell

Die Datenbank, die im Kapitel Datenbank 3.3.4 ausführlich beschrieben wird, und deren Modelle (Abbildung 2.1) sind einfach aufgebaut. Sie besteht aus zwei Datenfiles, einer Konfigurationsdatei und einer Logdatei. Dank des NoSQL-Ansatzes lässt sich das Modell problemlos erweitern.

- ParsedDoc: Hier werden die geparsten Informationen abgespeichert, welche dann vom Analyser ausgelesen werden.

- AnalyzedDoc: Die analysierten Daten befinden sich hier. Das Frontend erstellt aus diesen Informationen hauptsächlich die Grafiken für das Dashboard.
- PlatformConfig: Die Konfigurationen für eine Plattform werden hier gespeichert.
- AICLog: Das Logging, welches vom Frontend angezeigt werden kann.

2.2 Programmfluss

Der Standard-Programmablauf (Abbildung 2.2) umfasst vier Hauptkomponenten sowie den Scheduler. Die einzelnen Elemente des Flussdiagramms werden im Folgenden detailliert beschrieben.

2.2.1 Scheduler

Startet Scrape Jobs zu einem vordefinierten Zeitpunkt.

2.2.2 Scraper

Der Scraper geht über die Startseite jeder Plattform, welche aktiv in der PlatformConfig ist. Dort holt er sich alle URLs, welche nicht ausgefiltert werden und übergibt diese an den Parser.

2.2.3 Parser

Der Parser parst diese URLs und speichert den geparsten Text in das ParsedDoc. Bei einem Fehler wird dieser ins AICLog geschrieben.

2.2.4 Analyser

Der Analyser übernimmt die Daten im ParsedDoc, schickt diese zum LLM, welches von Infomaniak gehostet wird und speichert dann die Ergebnisse in AnalyzedDoc. Bei Fehlern wird ins AICLog geschrieben.

2.2.5 Streamlit

Streamlit[12] oder auch einfach das Frontend liest die Daten aus dem AnalyzedDoc und kreiert daraus Graphen, um diese dann auf der Webseite anzuzeigen.

Flussdiagramm

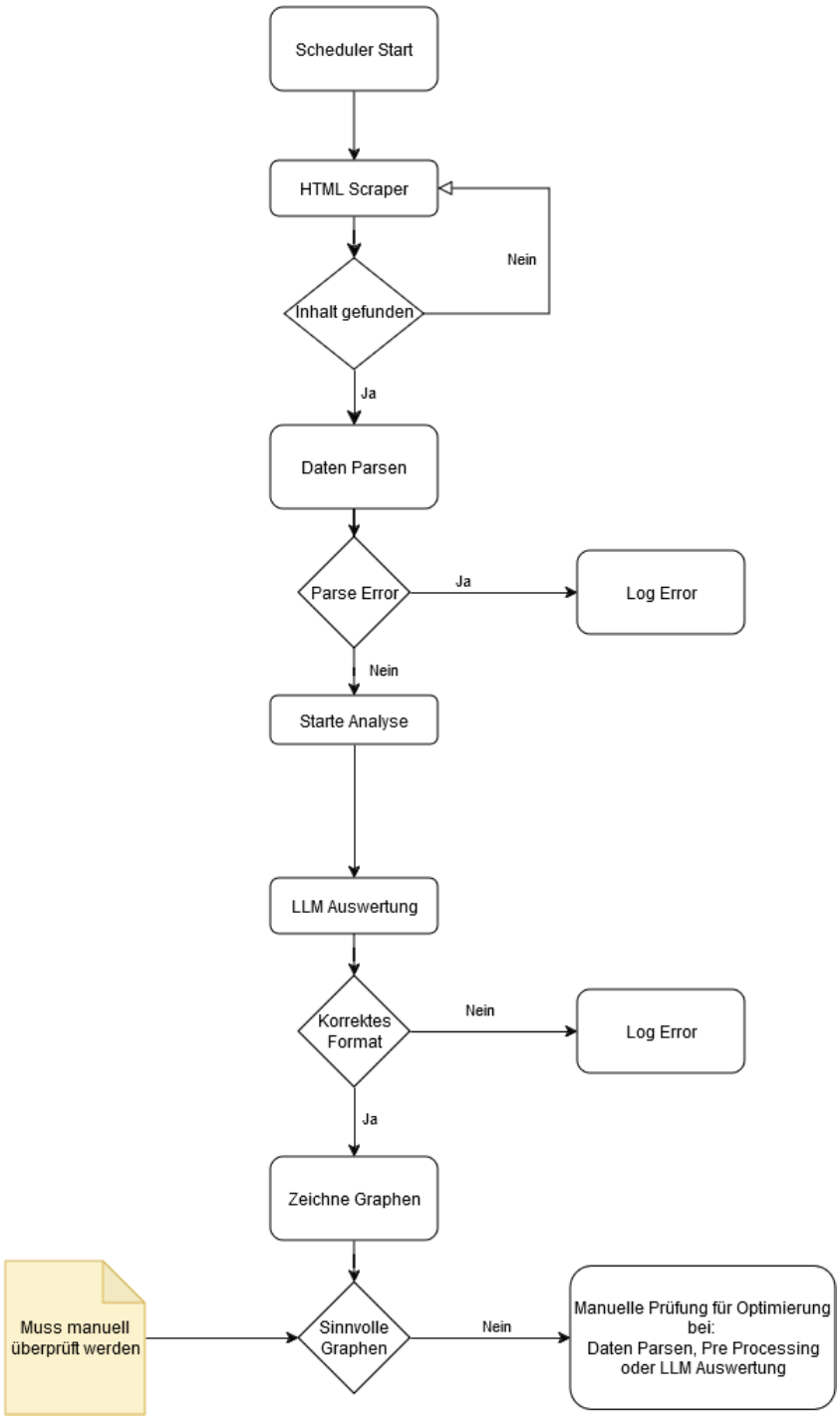


Abbildung 2.2: Programmflussdiagramm

Kapitel 3

Architektur

3.1 arc42 Modell

In diesem Abschnitt dokumentieren wir die Architektur des Systems mit Bezug zum arc42 Modell. Das arc42 Modell ist eines der leitenden Software Architektur Dokumentations-Templates. Es beinhaltet ein umfangreiches Framework zur detaillierten Beschreibung der Architektur eines Software-Systems. Darin inbegriffen sind die Strukturentscheidungen, Qualitätsanforderungen und Laufverhalten.

3.2 Stakeholders

Die Stakeholders des Projektes sind unser Betreuer (Frank Koch), der Industriepartner AdaptIT GmbH (Michael Güntensperger) und das Projektteam. Das Projektteam ist in Unterabschnitt 5.1.3 definiert.

3.3 Architekturentscheidungen

Die Architektur ist grundsätzlich als Drei-Schichten-Architektur aufgebaut, mit Frontend, Backend und der Datenbank. Das Ganze wird mittels Container-Image in Docker betrieben. Dies ermöglicht eine Vereinheitlichung der Entwicklungs- und Produktionsumgebung.

3.3.1 Produktionsumgebung

In der Produktionsumgebung haben wir zusätzlich Applikationen verwendet, für die Bereitstellung und Überwachung der Applikation.

Traefik

Traefik[13] ist ein Reverse Proxy und Loadbalancer, welcher es ermöglicht, die Applikation über das Internet zu erreichen. Mittels Service-Label auf dem Docker-Container

kann Traefik konfiguriert werden, wie der Container erreichbar ist. Dazu gehören Einstellungen wie Subdomain, Prefix, Port und SSL-Zertifikate, um nur einige zu nennen. Dies hat die nötige Flexibilität gegeben, um jederzeit einen Docker-Container verfügbar zu machen, falls dies benötigt wird.

Portainer

Portainer[16] bietet eine grafische Benutzeroberfläche zur Überwachung und Konfiguration von Docker-Containern.

Watchtower

Watchtower[4] ist ein Service, welcher die Docker-Container überwacht und bei einem Update automatisch die Container neu startet. Mittels Service-Label kann Watchtower konfiguriert werden, welche Container überwacht werden sollen.

Mongo-Express

Mongo-express[14] ist eine webbasierte MongoDB-Admin-Oberfläche, geschrieben in Node.js. Diese Oberfläche erlaubt das einfache Bearbeiten der Dokumente von MongoDB. Auch das Speichern, bzw. das Backup von Dokumenten ist einfach via diese Oberfläche möglich. Es hilft für die Entwicklung, Wartung und Überprüfung der gespeicherten Daten. Mongo-Express ist optional und für den Live-Betrieb nicht erforderlich.

3.3.2 Frontend

Python Streamlit

Für das Frontend fiel die Wahl auf Streamlit, da es eine einfache und schnelle Möglichkeit zur Visualisierung von Daten bietet. Des Weiteren ist die Grundlage Python keine neue Programmiersprache wie im Backend, was das Arbeiten und Verstehen des Codes für alle Teammitglieder erleichtert. Auch können Datenklassen direkt in Streamlit verwendet werden, da es auf Python basiert.

3.3.3 Backend

Python

Python ist eine sehr simple und direkte Programmiersprache, welche sehr geeignet ist, um verschiedene Komponenten zu verbinden. Für unseren Fall bedeutet das, die Verbindung von der Datenbank und der einzelnen Tools. (Webscraper, Datenbank, Streamlit)

Mongoengine

Das Mongoengine-Framework vereinfacht die Verbindung von Python-Code zu einer MongoDB mit ihrem ODM-Layout. Dies ermöglicht trotz einer dateibasierten Datenbank eine Struktur einfacher zu definieren.

3.3.4 Datenbank

MongoDB

Die Datenbank MongoDB[15] speichert die Daten in einem dokumentbasierten Format. Dies ist sehr geeignet für dieses Projekt, da viele Daten als JSON gespeichert und verarbeitet werden. Dank des NoSQL-Ansatzes lässt sich das Datenbankmodell problemlos erweitern und anpassen. Diese Flexibilität ist besonders vorteilhaft für zukünftige Anpassungen der Analyse. Ebenso werden viele Funktionalitäten bezüglich der Datensuche wie Filter, hybride Suche (Schlüsselwort- und Vektorsuchergebnis), Volltextsuche etc. von MongoDB unterstützt. Solche Funktionalitäten sind in diesem Fall eine ideale Grundlage für die effiziente Verarbeitung von Daten. Ein weiterer Vorteil ist die geringe Einstiegshürde bei der Nutzung von MongoDB sowie die bereits vorhandene Erfahrung mit der Datenbank im Team.
Nach einem Vektor-Datenbank-Vergleich[19] hat sich gezeigt, dass MongoDB alle der geforderten Funktionalitäten unterstützt.

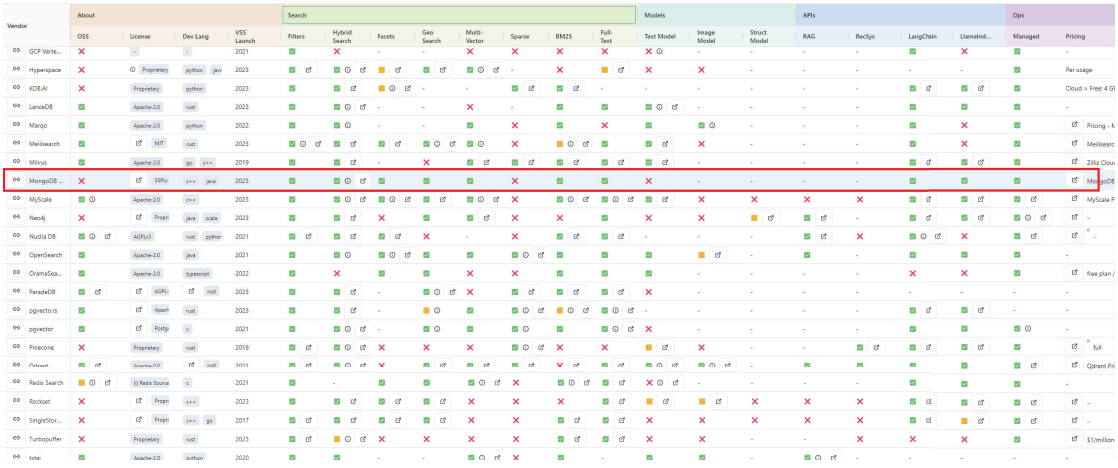


Abbildung 3.1: DB(Vector)-Vergleich

3.4 Software Struktur

Um den Techstack möglichst homogen zu gestalten, wurde Python 3 sowohl für das Backend als auch für das Frontend gewählt. Für die Entwicklung wurde VS Code mit

den Erweiterungen für Python und Devcontainer eingesetzt. Dies ist jedoch kein muss und die Wahl des Editiors ist frei möglich.

3.4.1 Modular

Die Softwarekomponenten werden modular gestaltet, um sowohl die Testbarkeit als auch die Wiederverwendbarkeit des Codes zu fördern. Ein modularer Aufbau erleichtert die Pflege und Erweiterung der Software, da einzelne Module unabhängig voneinander entwickelt, getestet und aktualisiert werden können.
Dies ist für die Erweiterbarkeit des Parser-Moduls erforderlich, weshalb in diesem Fall das Adapter-Pattern verwendet wurde.

3.4.2 Backend

Die Ordner im **src** Ordner des Backends teilen dieses in seine wichtigsten Module auf. Jedes Modul hat seine eigene Logik, wobei das main.py und core Modul diese verschiedenen Programm-Logiken zusammenbringen.
Das Parser Modul ist zusätzlich als Adapter-Pattern aufgebaut, was eine einfache Erweiterung von neuen Parsern ermöglicht.

Ordnerstruktur

```
backend/
├── src/
│   ├── main.py
│   ├── server.py
│   ├── start_uvicorn.py
│   ├── analyzer/
│   │   ├── api/
│   │   └── prompts/
│   ├── cli/
│   ├── core/
│   │   └── scheduler.py
│   ├── exceptions/
│   ├── migration/
│   ├── model/
│   ├── parser/
│   │   └── parsers/
│   ├── pdo/
│   ├── preprocess/
│   ├── scraper/
│   │   └── scrapers/
├── tests/
│   ├── analyzer/
```

```
├── core/
├── data/
├── parser/
├── preprocess/
└── scraper/
```

- **main.py**: Ursprünglicher Einstiegspunkt in das Projekt, führt die eigentliche Programm-Logik aus. (Ohne Scheduler und API)
- **server.py**: Definiert die API-Endpunkte und ruft das main.py im API-Modus auf und initialisiert den Scheduler.
- **start_uvicorn.py**: Startet die Applikation via den uvicorn Service.

3.4.3 Frontend

Das Frontend ist in drei Teile aufgeteilt:

- model: Die Datenbank Models, welche die Daten enthalten.
- pages: Die Views zum Anzeigen.
- services: Der Controller zwischen den Models und den Views.

```
frontend/
├── frontend/
│   ├── Home.py
│   ├── model/
│   ├── pages/
│   └── services/
└── tests/
    └── data/
```

- **Home.py**: Die Startseite, mit welcher das gesamte Frontend gestartet wird.

3.5 Schnittstellen

Die Applikation beinhaltet zwei verschiedene externe Interfaces:

- **Infomaniak**: Via API-Aufruf wird ein LLM von Infomaniak benutzt für die Datenanalyse.
- **Webseiten**: Verschiedene Webseiten werden vom Scraper geparst via URL-Requests.

3.6 Bausteinsicht

3.6.1 Whitebox Overall System

Das Front- und Backend haben die Möglichkeit, auf die Datenbank zuzugreifen.

3.6.2 Bausteine

Die Bausteine sind folgend aufgebaut: 3.2

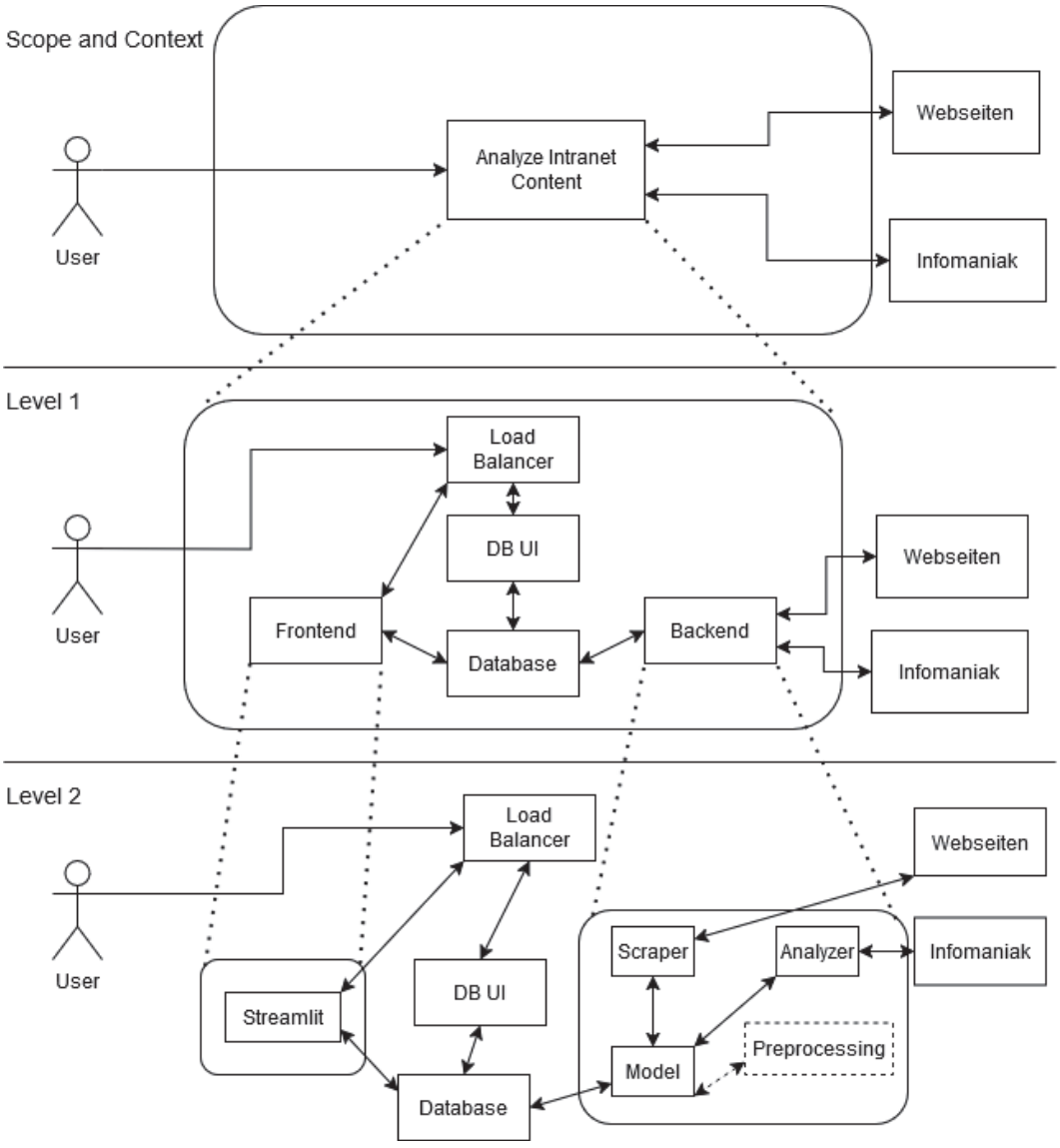


Abbildung 3.2: Bausteine

3.6.3 Kontext und Abgrenzung

Die Applikation verwendet Infomaniaks LLM für die Analyse der Daten. Es werden externe Webseiten nach Informationen geparst.

3.6.4 Level 1

Jeglicher Zugriff erfolgt über den Loadbalancer, welcher zusätzlich als Reverse Proxy funktioniert. Die Datenbank steht im Mittelpunkt der Applikation und alle Dienste haben Zugriff darauf.
Definition Level 1 in der Tabelle 3.1

Name	Verantwortung
Frontend	Zeigt Daten in einem Dashboard an
Backend	Sammelt und analysiert die externen Daten
DB UI	Administrationsansicht auf die Datenbank für Konfiguration
Database	Speichert Daten und stellt diese zur Verfügung
Loadbalancer	Übernimmt die Kommunikation zwischen dem Benutzer und der Applikation
Webseiten	Enthält Informationen, welche geparst werden
Infomaniak	Analysiert Daten mittels LLM

Tabelle 3.1: Kontext und Abgrenzung Level 1

3.6.5 Level 2

Definition Level 2 in der Tabelle 3.2

Name	Verantwortung
Streamlit	Zeigt analysierte Informationen im Dashboard an
Model	Gewährleistet Zugriff auf die Datenbank
Analyzer	Analysiert die geparsten Daten via Infomaniak
Scraper	Sammelt externe Daten
Preprocessing	Verbessert die geparsten Daten vor der Analyse

Tabelle 3.2: Kontext und Abgrenzung Level 2

3.7 Verteilungssicht

Die Verteilungssicht des Projekts ist in Abbildung 3.3 dargestellt. Grundsätzlich soll die Applikation möglichst einfach aufgebaut werden, damit sie lokal und in der Cloud betrieben werden kann. Die Applikation besteht aus drei Hauptkomponenten, dem Frontend, dem Backend und der Datenbank.
Das Frontend sowie auch das Backend sind mehrheitlich voneinander unabhängig. Einzige Ausnahme bilden Funktionsaufrufe vom Frontend auf das Backend via API bei einer Anpassung der Datenbank. Das Frontend bezieht seine Daten direkt von der MongoDB.

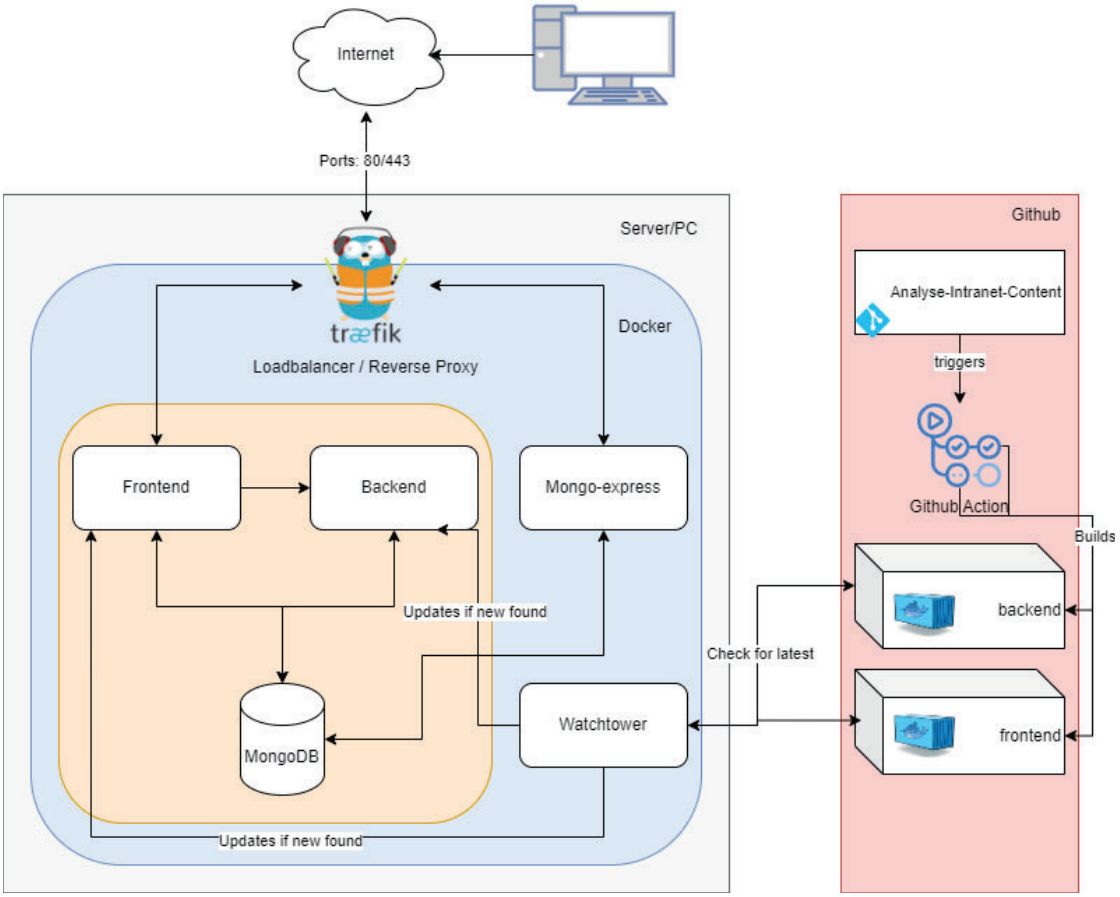


Abbildung 3.3: Verteilungssicht

3.7.1 Produktionsumgebung

Die Produktionsumgebung ist ein Linux Server, welcher Docker benötigt, um die Applikation zu betreiben. Da der Linux Server via Port 80 und 443 von überall erreichbar ist, ermöglicht es uns, via Reverse Proxy die Applikation über das Internet zu erreichen. Für die Funktionalität des Reverse Proxy, Routing und Loadbalancing wird Traefik verwendet. Dieser Service ermöglicht es mittels Docker-Labels, unser Frontend via Port 80/443 zur Verfügung zu stellen. Somit können wir das Dashboard von überall aufrufen und anzeigen lassen.

Kapitel 4

Qualitätsmessung

4.1 Definition of Done

Die folgenden Kriterien beschreiben unsere Definition, wann eine Arbeit als fertiggestellt definiert wird.

- Erwartete Funktionalitäten sind vorhanden.
- Alle automatisierten Tests sind erfolgreich.
- Alle Qualitätssicherungen sind durchlaufen.
- Dokumentation, Projektplan und Zeiterfassung sind auf dem neuesten Stand.

4.2 Code Qualität

Mittels linting und testing wird die Qualität des Codes sichergestellt. Dies wird durch die CI/CD Pipeline automatisiert durchgeführt, welche bei jedem Push, Pull Request oder Tag ausgeführt wird.

4.2.1 Qualitätssicherungen

Alle Qualitätssicherungen sind durchlaufen und die folgenden Kriterien wurden erfüllt:

Frontend

- Weniger als 5% duplizierter Code.
- Ruff Codequalitätstest bestanden.
- Pytest: alle Tests bestanden.
- Pylint Score von mindestens 9.5.

Backend

- Weniger als 5% duplizierter Code.
- Ruff Codequalitätstest bestanden.
- Pytest: alle Tests bestanden.
- Pylint Score von mindestens 9.5.

4.3 CI/CD Pipeline

Die CI/CD Pipeline besteht aus zwei verschiedenen Dateien, welche zwei verschiedene Pipelines definieren. Die "test.yml"Datei wird bei jedem Push, Pull Request oder Tag ausgeführt. Die "deploy.yml"Datei wird nur ausgeführt, wenn ein Tag erstellt wurde.

- **test.yml**: Dies ist die Hauptpipeline, welche bei jedem Push, Pull Request oder Tag ausgeführt wird.
  - **test**: Der Code wird mittels Pytest[9] getestet. Dieser Schritt wird immer ausgeführt, egal ob PR, Branch oder Tag.
  - **lint**: Der Code wird mittels Ruff auf Codequalität geprüft.
  - **build**: Der Code wird gebaut, mittels Docker, um sicherzustellen, dass der Code auch wirklich gebaut werden kann.
- **deploy.yml**: Dies ist die Deploy Pipeline, welche nur ausgeführt wird, wenn ein Tag erstellt wurde.
  - **build**: Der Code wird gebaut, mittels Docker, um sicherzustellen, dass der Code auch wirklich gebaut werden kann.
  - **image**: Das Docker Image, welches beim Build erstellt wurde, wird mit dem entsprechenden Tag versehen und in das Gitlab Registry hochgeladen.
  - **deploy**: Via Webhook wird das Deployment auf dem Server ausgelöst. Dies passiert jedoch nur, falls ein Versionstag gemacht wurde.

4.3.1 Test

Die Tests werden mittels Pytest ausgeführt. Um allfällige Integrationen und Schnittstellen zu testen, verwenden wir Testcontainers. Mit Testcontainers können wir Docker Container starten und stoppen, um die Integration zu testen. Dies ermöglicht uns, die Tests in einer isolierten Umgebung auszuführen und die meisten Testfälle abzudecken. Dieser Schritt wird immer ausgeführt, egal ob PR, Branch oder Tag.

4.3.2 Lint

Der Code wird mittels Ruff auf Codequalität geprüft. So können wir sicherstellen, dass der Code sauber und lesbar ist und den definierten Code-Standards entspricht. Dieser Schritt wird immer ausgeführt, egal ob PR, Branch oder Tag.

4.3.3 Build

Mit dem Docker Build wird der Code gebaut. Dieser Schritt wird immer ausgeführt, egal ob PR, Branch oder Tag.

4.3.4 Image

Der Image Schritt wird nur ausgeführt, wenn ein Tag erstellt wurde. Hier wird dem Docker Image ein Tag hinzugefügt und in das Github Registry hochgeladen.

4.3.5 Deploy

Der Deploy Schritt wird nur ausgeführt, wenn ein Tag erstellt wurde. Hier wird das Deployment auf dem Server ausgelöst, welcher via Webhook angesteuert wird. Somit können wir sehr schnell und einfach neue Versionen deployen und live schalten.

4.4 Test Konzept

Das Testkonzept beschreibt die Teststrategie, Testtechniken, Testumgebung, Testergebnisse, Testplan, Testrollen und Testartefakte.

4.4.1 Funktionalitäten zum Testen

- Scraping
- Parsing
- Analyse
- Visualisierung der Daten

4.4.2 Test Strategie

Die Strategie für das Testen beinhaltet folgende Punkte:

- Identifizierung der Testfälle.
- Ausführung der Tests. Die Tests werden automatisiert und manuell durchgeführt.
- Reporting von Fehlern und Testergebnissen.
- Wiederholung der Tests bis alle Fehler behoben sind.

4.4.3 Test Techniken

Folgende Testtechniken werden in unserem Projekt verwendet:

- Automatisch
  - Unit-Tests
  - Integrationstests
  - Verfügbarkeitstests
- Manuell
  - Systemtests

4.4.4 TestUmgebung

4.4.5 Server

- OS: Ubuntu 24.04 LTS
- Docker: Version 27.3.1

- Docker-Compose: Version 2.25.0
- Python: Version 3.12
- MongoDB: Version 8.0.0

Kunde

- OS: Windows 11
- Browser: Chrome, Firefox höher als Version 90

4.4.6 Test-Ergebnisse

Die Testergebnisse sind:

- Testartefakte

4.4.7 Test Plan

Das Testing wird mit den Veröffentlichungen der Meilensteine (Beta, Release) gemacht. Nur die bereits verfügbaren Test-Cases werden überprüft. Die Test-Cases sind in den folgenden Tabellen 4.1, 4.2, 4.3, 4.5, 4.6, 4.7, 4.8, 4.9 und 4.4 definiert.

Grober Plan

- Woche 47: Test Beta
- Woche 50: Test Release

Detaillierter Plan

Frontend
Beim Frontend wird Folgendes getestet:

- Korrektes Anzeigen der Daten
- Zeitliche Entwicklung verfolgen
- Inputs korrekt verarbeiten

ID	TC-1
Leiter	Cedric
Titel	Korrektes Anzeigen der Daten
Voraussetzung	Daten erhalten und erfolgreich ausgewertet.
Beschreibung	• Die Graphen zeigen die korrekten Daten an und sind korrekt skaliert dargestellt. • Die Daten werden korrekt aktualisiert.
Erwartetes Resultat	Daten Graphen in der korrekten Skala welche aktualisiert werden können.

Tabelle 4.1: TC-1

ID	TC-2
Leiter	Cedric
Titel	Zeitliche Entwicklung Verfolgen
Voraussetzung	Daten erhalten und erfolgreich ausgewertet über einen Zeitraum von mindestens zwei Tagen.
Beschreibung	• Daten können über zeitraum dargestellt werden.
Erwartetes Resultat	Zeitstrahl Diagramm wird dargestellt

Tabelle 4.2: TC-2

ID	TC-3
Leiter	Cedric
Titel	Inputs korrekt verarbeiten
Voraussetzung	Inputs erhalten und korrekt verarbeitet.
Beschreibung	• Inputs werden korrekt verarbeitet. • Inputs werden korrekt dargestellt.
Erwartetes Resultat	Inputs werden korrekt verarbeitet und dargestellt.

Tabelle 4.3: TC-3

ID	TC-9
Leiter	Cedric
Titel	Invalide Daten
Voraussetzung	Invalide Daten erhalten.
Beschreibung	• System soll bei Invaliden Daten nicht abstürzen. • Eine Fehlermeldung wird angezeigt.
Erwartetes Resultat	Fehler wird angezeigt.

Tabelle 4.4: TC-9

Backend
Beim Backend werden wir folgende Testfälle durchführen:

- HTML scrapen
- Fehlerverhalten bei Parser Fehler
- LLM Analyse Kategorisierung
- Fehlerverhalten bei LLM Analyse
- Lasttest

ID	TC-4
Leiter	Cedric
Titel	Scrape HTML
Voraussetzung	Die zu scrapende Webseite ist verfügbar.
Beschreibung	• Html wird korrekt gescraped. • Die Daten werden korrekt verarbeitet. • Die Daten werden korrekt gespeichert.
Erwartetes Resultat	HTML konnte gescraped, verarbeitet und erfolgreich gespeichert werden.

Tabelle 4.5: TC-4

ID	TC-5
Leiter	Cedric
Titel	Fehlerverhalten bei Parser Fehler
Voraussetzung	Die zu scrapende Webseite ist verfügbar.
Beschreibung	• Html wird korrekt gescraped. • Die Daten werden nicht korrekt verarbeitet. • Fehler Protokoll wird geschrieben.
Erwartetes Resultat	HTML konnte gescraped, jedoch nicht korrekt verarbeitet werden. Fehlerprotokoll wird geschrieben

Tabelle 4.6: TC-5

ID	TC-6
Leiter	Cedric
Titel	LLM Analyse Kategorisierung
Voraussetzung	Webseite konnte gescraped werden und Daten sind gespeichert.
Beschreibung	• Daten werden gelesen und vorverarbeitet. • Entsprechender Prompt wird mit den Daten an API gesendet. • Rückgabe wird verarbeitet und gespeichert.
Erwartetes Resultat	Analysierte Daten von der API, welche abgespeichert werden

Tabelle 4.7: TC-6

ID	TC-7
Leiter	Cedric
Titel	Fehlerverhalten bei LLM Analyse
Voraussetzung	Webseite konnte gescraped werden und Daten sind gespeichert.
Beschreibung	• Daten werden gelesen und vorverarbeitet. • Entsprechender Prompt wird mit den Daten an API gesendet. • Rückgabe kann nicht verarbeitet werden. • Fehler Protokoll wird geschrieben.
Erwartetes Resultat	Analysierte Daten von der API können nicht verwertet werden. Fehlerprotokoll geschrieben.

Tabelle 4.8: TC-7

ID	TC-8
Leiter	Cedric
Titel	Lasttest
Voraussetzung	Die ganze Applikation ist lauffähig.
Beschreibung	• Scrapen von 1 Artikel. • Danach Analyse des Artikels. • Scrapen von 10 Artikeln. • Analyse wird parallel gestartet, sobald 1. Artikel gescraped wurde. • Scrapen von 100 Artikeln. • Analyse wird parallel gestartet, sobald 1. Artikel gescraped wurde.
Erwartetes Resultat	Die App bleibt während des ganzen Tests resposive und stürzt nicht ab.

Tabelle 4.9: TC-8

4.4.8 Test Rollen

- Leiter: Verantwortlich für das Erstellen, Ausführen und Berichten.

- Tester: Verantwortlich für die Testausführung, Bug Reporting.

4.4.9 Test Artefakte

Durch die Änderung der Analyse fällt TC-9 aus, da es nicht mehr möglich ist, invalide Daten zu erhalten. Die Testartefakte sind im Anhang9 unter dem Kapitel Test Artifakte zu finden.

Teil III

Projekt Dokumentation

Kapitel 5

Projektplan

5.1 Ablauf

Wir entschieden uns für die Benutzung von Scrum+. Dabei handelt es sich um eine Kombination von Scrum mit dem Rational Unified Process (RUP). Grund dafür ist die Möglichkeit, eine Langzeitplanung des RUP mit der Kurzzeitplanung von Scrum kombinieren zu können. Das ermöglicht uns, einen groben Zeitplan zu erstellen, aber immer noch genügend Freiraum für allfällige Änderungen zu haben.

5.1.1 Iteration

Im Rahmen unseres Projekts wurden zu Beginn wöchentliche Iterationen durchgeführt. Dies führte zu schnellen Fortschritten, regelmässigen Feedbacks und liess uns flexibel auf kurzfristige Änderungen reagieren. Diese kurzen Iterationen waren besonders hilfreich in der frühen Projektphase, da wir uns schnell auf das Produktkonzept und die Anforderungen der Stakeholder einstellen konnten.
Nach einer Evaluierungsphase wurde jedoch festgestellt, dass der wöchentliche Rhythmus zu häufigen Übergaben und erhöhter Planungszeit führte, was den Arbeitsfluss teilweise beeinträchtigte. Aufgrund dieser Erkenntnisse wurde beschlossen, die Iterationsdauer auf einen zweiwöchigen Zyklus anzupassen. Dieser Wechsel ermöglichte es, uns tiefergehend mit den Aufgaben zu befassen und grössere Inkremente fertigzustellen, ohne die Agilität des Projekts zu beeinträchtigen.
Durch die Anpassung der Iterationslänge konnten wir die Balance zwischen Flexibilität und Produktivität optimieren und die Effizienz unseres Entwicklungsprozesses verbessern.

5.1.2 Zeiterfassung und Tickets

Die Zeiterfassung und Tickets werden über Jira gemanaged. Die Zeiten werden auf 15 Minuten gerundet.

5.1.3 Rollen

Scrum Master

Überblick über die Aufgaben des Scrum Masters:

- Organisiert Scrums und Sprints
- Leitet Scrum Meetings
- Überprüft das Meetingprotokoll
- Unterstützt den Product Owner mit der Verwaltung des Backlogs

Diese Rolle wird übernommen von: Cedric Christen

Product Owner

Überblick über die Aufgaben des Product Owners:

- Verwaltung des Backlogs mit Priorisierungen
- Meetings organisieren
- Kommuniziert mit dem Berater
- Übergibt die Dokumente zeitgerecht

Diese Rolle wird übernommen von: David Stäheli

Project Manager

Überblick über die Aufgaben des Project Managers:

- Überprüft, dass die Leitfäden eingehalten werden
- Überprüft, dass die Pipelines funktionieren
- Überprüft, dass die Qualität zufriedenstellend ist

Diese Rolle ist aufgeteilt in 4 Unterrollen:

- UI-Manager ist Cedric Christen
- AI-Manager ist David Stäheli
- Scraper-Manager ist Cedric Christen
- Documentation-Manager ist Cedric Christen
- Infrastructure-Manager ist David Stäheli

Entwickler

Überblick über die Aufgaben des Entwicklers:

- Arbeitet an Arbeitspunkten

Diese Rolle wird übernommen von: Beiden

5.2 Leitfaden für den Code

Backend und Frontend sind beide in Python geschrieben, weswegen der Leitfaden für beide Module der gleiche ist. Der Leitfaden bezieht sich auf PEP 8 - Style Guide for Python Code.[21]
Die wichtigsten Regeln sind:

- Snake case für Methoden, Dateinamen und Variablen.
- Camel case für Klassen.
- Einrückung von 4 Leerschlägen.

Dies wird überprüft mit dem ruff[10] Linter Check in der Pipeline.

- Konfigurationsdateien werden von PEP 8 ignoriert.
- Zeilenlänge wird auf 88 Zeichen gesetzt.

Die genaue Konfiguration kann in pyproject.toml gefunden werden.

5.3 Leitfaden für die Dokumentation

Der Leitfaden bezieht sich auf WRITE THE DOCS.[20]
Dies bedeutet, dass zum Beispiel Hyperlinks direkt eingefügt werden sollen und nicht als “Click here“ oder ähnliches. Die Sprache sollte ausserdem einheitlich sein in der Dokumentation.

5.4 Phasen

Die Phasen orientieren sich grundsätzlich nach den Meilensteinen.

1. Inception 16.09.2024 - 29.09.2024
2. Elaboration 30.09.2024 - 27.10.2024
3. Construction 28.10.2024 - 15.12.2024
4. Transition 16.12.2024 - 20.12.2024

5.5 Meilensteine

Meilensteine sind jeweils auf den Donnerstag gesetzt.

M1 Projekt Einrichtung

Datum: 29.09.2024
Ziel:

1. Aufgabenstellung definieren
2. Grober Zeitplan erstellen

M2 Anforderungen/Scraper Einrichten

Datum: 13.10.2024
Ziel:

1. Scraper Prototyp eingerichtet und angefangen zu crawlen
2. Grober Zeitplan abgeschlossen
3. Risiken definiert
4. Use Cases definiert

M3 MVP

Datum: 27.10.2024
Ziel:

1. MVP erstellt
2. Architektur dokumentiert

M4 Beta

Datum: 17.11.2024
Ziel:

1. Beta erstellt
2. Qualitätssicherungen vorbereitet

M5 Final Submission

Datum: 22.12.2024
Ziel:

1. Dokumentation abgegeben
2. Code deployed
3. Qualität gesichert

5.6 Geplante Releases

Minimum Viable Product (MVP)

Release Date: 27.10.2024
Der MVP beinhaltet folgende Funktionalitäten:

- Scraping von einer Webseite
- AI Analyse von den HTML Seiten

Beta

Release Date: 17.11.2024
Die Beta beinhaltet folgende Funktionalitäten:

- Scraping von einer weiteren Webseite
- UI Tool mit einfachen Filtern
- Optimierung der AI

Final Release

Release Date: 20.12.2024
Der Final Release beinhaltet neben den vorherigen Releases folgende Funktionalitäten:

- Scraping von weiteren Webseiten
- Optimierung der AI
- UI Tool mit Konfigurationsmöglichkeiten

5.7 Langzeitplan / Roadmap

Das Projekt hat am 16.09.2024 angefangen und wird am 20.12.2024 mit der Abgabe der Dokumentation abgeschlossen. Der Plan ist in Abbildung 5.1 grafisch dargestellt.

Project Plan

			W38 16.09 - 22.09	W39 23.09 - 29.09	W40 30.09 - 06.10	W41 07.10 - 13.10	W42 14.10 - 20.10	W43 21.10 - 27.10	W44 28.10 - 03.11	W45 4.11 - 10.11	W46 11.11 - 17.11	W47 18.11 - 24.11	W48 25.11 - 01.12	W49 02.12 - 08.12	W50 9.12 - 15.12	W51 16.12 - 22.12
Start Date	16.09.2024															
End Date	20.12.2024			M1		M2		M3			M4					M5
1	Business Modelling															
1.1	Rough Project Plan (RUD)															
2	Requirements															
2.1	Non Functional Requirements															
2.2	Functional Requirements															
2.3	Domain Model / Additional Artifacts															
3	Architecture															
3.1	Define Architecture															
3.2	Prototype															
4	Implementation															
4.1	MVP implementation															
4.2	Beta															
4.3	Final product implementation															
4.4	NFR implementation															
5	Test															
5.1	Testing of FR & NFR															
6	Project Management															
6.1	Planning															
6.2	Risk Management															
6.3	Documentation															
7	Environment															
7.1	Tooling setup															
7.2	Repository setup															
7.3	CI/CD setup															
7.4	Test environment setup															

Abbildung 5.1: Roadmap

5.8 Risikomanagement

Das Risikomanagement wurde strukturiert in den Tabellen 5.1 bis 5.5 dokumentiert. Diese umfassen Risiken wie die Verfügbarkeit von AI-Providern, unerwartete Kosten, Änderungen bei zu scrapenden Webseiten und die Qualität der AI-Kategorisierung.

5.8.1 Risiko

ID	**R1**
ID	1
Risiko	AI Provider ist nicht mehr verfügbar.
Kommentar	AI Provider ist nicht mehr verfügbar. (Bankrott, Service-Wechsel, etc.)
Vorbeugung	Die AI API dynamisch gestalten, so dass diese einfach mit einem neuen Provider verbunden werden kann.
Korrigierung	Wechsel von AI Provider mit neuem oder lokalem.
Schweregrad	Mittel
Wahrscheinlichkeit	Sehr tief

Tabelle 5.1: Risiko 1

ID	**R2**
ID	2
Risiko	AI Provider Kosten unterschätzt
Kommentar	AI Provider Kosten stellen sich als höher heraus, als zuerst gedacht.
Vorbeugung	Kostenschätzung machen und nach Start überprüfen, ob diese stimmt.
Korrigierung	NLP vor der Analyse benutzen, um die Anzahl Tokens zu verkleinern.
Schweregrad	Mittel
Wahrscheinlichkeit	Tief

Tabelle 5.2: Risiko 2

ID	**R3**
ID	3
Risiko	Scraping Webseite nicht mehr verfügbar
Kommentar	Eine Webseite, welche gescraped werden sollte, ist nicht mehr verfügbar.
Vorbeugung	Webseiten zum Scrapen sind konfigurierbar und können entfernt oder hinzugefügt werden.
Korrigierung	Webseite aus der Liste nehmen und gegebenenfalls mit anderer ersetzen.
Schweregrad	Sehr tief
Wahrscheinlichkeit	Hoch

Tabelle 5.3: Risiko 3

ID	**R4**
ID	4
Risiko	Scraping Webseite ändert Struktur
Kommentar	Eine Webseite, welche gescraped wird, kriegt ein Update und ändert die Struktur.
Vorbeugung	Scraper allgemein halten mit konfigurierbaren Filtern, welche ausgewechselt werden können.
Korrigierung	Filter erneuern oder zu Standardwerten zurückgreifen. Scraper wechselt automatisch zu Standard, falls nichts mehr gefunden wird.
Schweregrad	Sehr tief
Wahrscheinlichkeit	Hoch

Tabelle 5.4: Risiko 4

ID	**R5**
ID	5
Risiko	AI Kategorisierung nicht zufriedenstellend
Kommentar	Die Kategorisierung der AI ist nicht zufriedenstellend für eine Analyse.
Vorbeugung	Frühzeitig testen, ob die AI gute Ergebnisse liefern kann mit einem Prototypen.
Korrigierung	AI Kategorisierung verbessern und Daten vorzeitig besser aufbereiten.
Schweregrad	Sehr hoch
Wahrscheinlichkeit	Mittel

Tabelle 5.5: Risiko 5

5.8.2 Risikomap

Die Risikomap (Abbildung 5.2) ist wie folgt definiert:

Wahrscheinlichkeit

Die Wahrscheinlichkeit zeigt, wie sehr erwartet wird, dass etwas auftritt. Es wird in Prozent angegeben.

- Sehr hoch: 75%-100%
- Hoch: 50%-75%
- Mittel: 30%-50%
- Tief: 10%-30%
- Sehr tief: 1%-10%

Schweregrad

Der Schweregrad zeigt, wie lange es dauert, den Schaden eines Risikos zu reparieren, falls es eintritt.

- Sehr hoch: 16h+
- Hoch: 8h-16h
- Mittel: 4h-8h
- Tief: 2h-4h
- Sehr tief: 0h-2h

Sehr Tief	Tief	Mittel	Hoch	Sehr Hoch	Wahrscheinlichkeit / Schweregrad
		5			**Sehr Hoch**
					Hoch
1	2				**Mittel**
					Tief
			3,4		**Sehr Tief**

Abbildung 5.2: Riskmap

Kapitel 6

Projekt

6.1 AI

Ein wesentlicher Bestandteil des Projekts ist die Integration von AI für die Auswertung der Daten, welche aus verschiedenen Quellen kommen können. Dafür muss ein passender Prompt (Eingabeaufforderung) erstellt werden und auch die verschiedenen verfügbaren Modelle gegeneinander verglichen werden. Auch sind die entstehenden Kosten ein Faktor für die Auswahl eines geeigneten Prompts und Modells. Als API wurde die Infomaniak API vorgegeben, welche auf der OpenAI API basiert, jedoch andere Modelle und Preise anbietet. Um die Kosten am Anfang gering zu halten und mit den Prompts und Modellen zu experimentieren, fiel die Wahl zu Beginn auf das günstigste Modell Mixtral 8x7b.

6.1.1 Prompt und API Aufruf

Eine initiale Version des Prompts wurde relativ schnell erstellt, welche mithilfe von ChatGPT aufgearbeitet worden ist. Diese Version entsprach jedoch nicht allen Anforderungen, die wir an die Auswertung hatten. Wir erhielten 95% der Antworten korrekt. Manchmal wurde jedoch die Antwort nicht wie angefordert als "JSON", sondern als normaler Text zurückgegeben. Dies führte dazu, dass die Antwort nicht korrekt verarbeitet werden konnte. Auch wurden gewisse Aufzählungen nicht korrekt gezählt, was direkt keine Fehler verursachte, jedoch in der Auswertung die Graphen verfälschte.
Nach einigen weiteren Iterationen mit der Infomaniak API und dem Prompt kamen wir auf eine Version, welche mehrheitlich ein brauchbares Resultat erzielte. Grund dafür war, dass wir noch keine Erfahrung hatten, strukturierte Daten der AI APIs zu erhalten.
Bei einer späteren Iteration und nochmaliger Überarbeitung des Prompts, weil ursprünglich zusätzliche Informationen ausgewertet werden sollten, optimierten wir auch den Aufruf der API, sodass der Prompt nicht mit dem eigentlichen Inhalt mitgegeben wurde, sondern vorgelagert als System Argument, welches die Antwort der API konsistenter machte. Somit hatten wir gleich zwei Probleme gelöst: Die Antwort war

konsistenter, was zur Folge hatte, dass die Parser-Fehler nicht mehr auftraten, und wir konnten auch die zusätzlichen Informationen auswerten.

6.1.2 Modelle Vergleich

Die Infomaniak API bietet zum Zeitpunkt dieser Studienarbeit 3 verschiedene Modelle[1] über ihre API an. Dies sind Llama3, Mixtral 8x7b und Mixtral 8x22b, welche unterschiedliche Preise, Genauigkeiten sowie Stärken haben. Das preisgünstigste Modell ist Mixtral 8x7b, welches auch die geringste Genauigkeit hat. Danach folgt Mixtral 8x22b und Llama3, welche beide gleich teuer sind, jedoch Llama3 eine höhere Genauigkeit hat für den Anwendungsfall dieses Projekts.
Wenn die Analyse der Modelle miteinander verglichen wird, ist zu erkennen, dass beim Auszählen der Anzahl Schlüsselwörter, Llama3 die exaktesten Ergebnisse aller Modelle liefert. Dies liegt zum einen daran, dass der geparste Artikel nicht nur den "reinen" Text enthält, sondern auch noch zusätzliche Informationen wie zum Beispiel den Autor, Hash-Tags, etc. Diese Informationen werden von Llama3 korrekterweise beim Zählen der Schlüsselwörter ignoriert, was zu einer höheren Genauigkeit führt und somit auch zu einer besseren Auswertung.
Hier zu sehen im Beispiel: Abbildung 6.1.
Der ausgewertete Text enthält unter anderem das Schlüsselwort: AI. Dieses kommt im Text genau 14 Mal vor. Jedoch im gesamten Artikel, mit allen Hash-Tags, "Teilwörtern", Links etc., wird es 35 Mal gezählt.

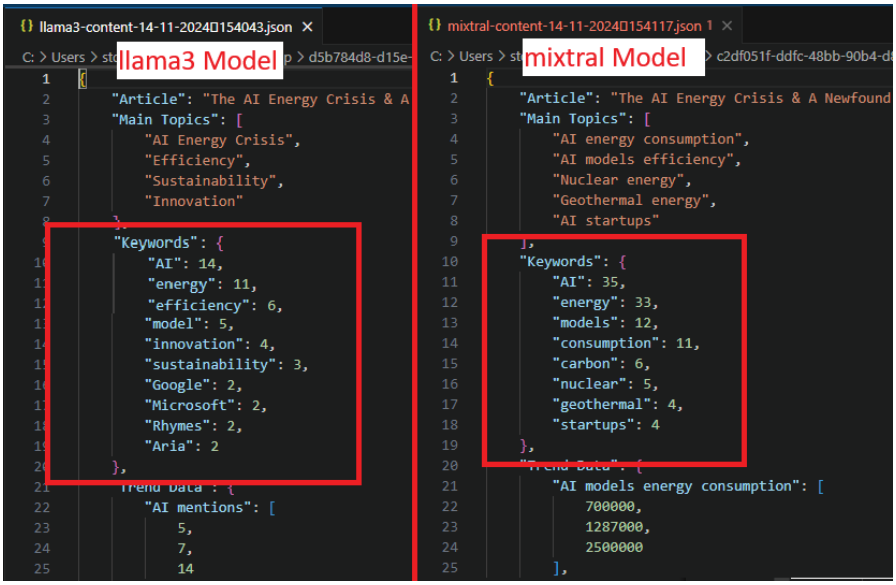


Abbildung 6.1: Vergleich von Keywords der Modelle Llama3 und Mixtral

6.2 Frontend

Für das Frontend mussten wir zwischen zwei Möglichkeiten entscheiden. Entweder Streamlit[12] oder Plotly[11]. Da das Frontend nicht im Fokus stand, wurde Streamlit gewählt, da es einfach einzurichten und zu bedienen ist. Ausserdem unterstützt Streamlit auch Plotly Diagramme, wodurch wir die gleichen Diagramme wie bei Plotly zur Verfügung hatten.
Die anfängliche Einrichtung und das Vertrautmachen mit den Besonderheiten von Streamlit führten zunächst zu Verzögerungen. Nachdem jedoch die Funktionsweise des Tools verstanden war, konnten schnelle Fortschritte erzielt und das Frontend dynamisch aufgebaut werden. Erst gegen Ende spürten wir die technischen Limiten von Streamlit. Formulare zu erstellen war sehr simpel, aber dynamische Felder zu erzeugen war nicht intuitiv. Ausserdem gab es für die Authentifizierung keine Abhilfe, weswegen wir diese nur bei der sicherheitskritischen Admin-Seite angewendet haben.

6.3 Scraper und Parser

Die Informationsgewinnung wird von den zwei Modulen Scraper und Parser übernommen. Zuerst sucht der Scraper die richtigen URLs, welche dann an den Parser weitergegeben werden, wo dann die eigentlichen Informationen herausgelesen werden. Der Scraper und Parser sind sehr einfach aufgebaut und suchen die Startseite einer Plattform nach allen URLs ab. Schnell entschieden wir uns, den Scraper mit Filtern zu erweitern, damit wir nicht die Navigation oder User Profile mitzogen.
Der Parser blieb während der ganzen Entwicklung beinahe unangetastet, bis auf die Erweiterung von einem Klassenfilter.

6.4 Scheduler

Zu Beginn entschieden wir uns, den Scheduler mit einem Cronjob[5] zu implementieren. Dieser würde dann jeden Tag um die gleiche Zeit den Scraper starten, welcher dann die Dokumente an die anderen Module weitergab. Jedoch bereitete uns dieser Weg viele Probleme. Das Erstellen des Cronjobs war sehr simpel, jedoch war es uns nicht möglich, ein Python Skript direkt auszuführen. Deswegen erstellten wir eine Shelldatei zwischen dem Cronjob und dem Python Skript. Doch auch dies funktionierte nur, wenn wir den Docker mit dem Root Benutzer ausführen. Zusätzlich muss Cronjob manuell neugestartet werden, falls dieser aus irgendeinem Grund unterbrochen wurde.
Nach einer Recherche zu den Cronjob Problemen mit Docker Containern entschieden wir, den Scheduler mit dem Python Modul Schedule[3] zu implementieren. Dies ist eine sehr schlanke und einfache Lösung, welche es erlaubt, periodisch Funktionen auszuführen, ohne eine Cron Konfiguration zu erstellen und mit den dazugehörigen Schwierigkeiten bei Docker Containern. Die Einstiegshürde in das Modul war sehr gering und wir konnten schnell den Scheduler implementieren, testen und in Betrieb nehmen.

Es stellte sich jedoch heraus, dass diese minimale Lösung auch ihre Nachteile hat, was mit der Implementierung einer API Schnittstelle im Backend ersichtlich wurde. Da der Scheduler im Main Thread läuft, konnten wir nicht ohne Weiteres eine API Schnittstelle implementieren, da die API in einem eigenen Thread laufen muss, damit der Scheduler nicht blockiert wird.
Damit diese Thread Implementierung nicht komplet neu gemacht werden mussten, wurde entschieden, die Bibliothek für den Scheduler nochmals zu wechseln, was uns zum APScheduler (Advanced Python Scheduler)[7] führte. Diese Bibliothek erlaubt es, den Scheduler in einem eigenen Hintergrund Thread zu starten, während die API im Main-Thread läuft. Dies ermöglichte, die API Schnittstelle zu implementieren und den Scheduler weiterhin zu verwenden, wobei nur das Scheduler Modul auf die neue Bibliothek anpassen mussten. Dies ist auf den modularen Aufbau der Softwarestruktur zurückzuführen.
Rückblickend hätte es uns viel Zeit erspart, wenn wir die Entscheidung einer API Schnittstelle früher getroffen hätten. Somit wären wir viel früher auf die Probleme mit dem Scheduler gestossen und nicht erst im letzten Drittel der Studienarbeit. Jedoch war es auch eine gute Erfahrung, die verschiedenen Implementierungen, wie Aufgaben zeitgesteuert ausgeführt werden können, zu lernen und zu verstehen. Das Einrichten eines Cronjobs ist eine einfache Lösung, für wiederkehrende Aufgaben im Linux System, jedoch nicht eine optimale Lösung für einen Docker Container, welcher idealerweise nur einen einzigen Prozess ausführt. Das Ausführen eines Cron Daemons widerspricht jedoch diesem Prinzip.

6.5 API

Der ursprüngliche Plan des Architektur Designs war, dass das Frontend direkt auf die Datenbank zugreift, was auch in der Entwicklung so umgesetzt wurde und problemlos funktioniert. Jedoch stellte sich gegen Ende des Projektes heraus, dass wir keine zufriedenstellende Lösung für das Ausführen von manuellen Aufgaben via das Frontend für den Benutzer bereitstellen können. Deswegen wurde entschieden, eine API Schnittstelle zu implementieren, welche nur die minimalsten Funktionen bereitstellt, um z.B. den Scheduler neu zu starten oder die Analyse manuell zu starten, wenn z.B. das Modell geändert wurde.
Für diese API Schnittstelle verwenden wir das Python-Modul FastAPI[17]. Dies ist ein einfaches, modernes und asynchrones Web-Framework, welches Python Type Hints verwendet, um die API zu definieren und zu dokumentieren. Dies bietet den Vorteil, dass die API automatisch dokumentiert wird und diese Dokumentation über den Webbrowser aufgerufen werden kann unter dem API-Endpunkt /docs. Somit können wir die API einfach testen und zusätzlich ist klar ersichtlich, welche Endpunkte und Funktionen zur Verfügung stehen.

6.6 Schlussfolgerung

Die Evaluation der Technologien und Werkzeuge für die Studienarbeit hat gezeigt, dass Llama3 sich als zuverlässigste Lösung für Textanalysen erwiesen hat. Es hat die präzisesten Ergebnisse geliefert, und die ursprünglich befürchteten Kostenüberschreitungen blieben aus, da die tatsächliche Nutzung die Schätzungen unterschritten hat.
Die Nutzung von Cron in Verbindung mit Docker stellte eine Herausforderung dar, insbesondere im Kontext der Prozessausführung und Stabilität. Eine schrittweise Migration zu Python-basierten Scheduling-Lösungen wie APScheduler hat sich letztlich als stabil erwiesen und ermöglicht eine nahtlose Integration mit anderen Backend-Komponenten.
Für das Frontend wurde Streamlit gewählt, da es eine einfache Visualisierung von Daten bietet. Dennoch zeigte sich, dass die Plattform bei spezialisierten Anforderungen wie der dynamischen Erzeugung komplexer Benutzeroberflächen an ihre Grenzen stösst. Die Eignung von Streamlit für Prototypen und kleinere Projekte wurde jedoch bestätigt.
Insgesamt konnte das Projekt durch iterative Verbesserungen und die Anpassung der eingesetzten Technologien erfolgreich abgeschlossen werden. Die gewählten Werkzeuge erwiesen sich, mit Ausnahme der genannten Einschränkungen, als geeignet, die gesteckten Ziele zu erreichen.

Kapitel 7

Ausblick

Die Grundbausteine für ein umfassendes Analyse-Projekt sind gelegt. Erweiterungen in verschiedensten Bereichen der Applikation sind möglich, um deren Funktionalität und Anwendungsbereich zu erweitern. Hier sind einige Ideen, die in Zukunft umgesetzt werden könnten:

- Das Hinzufügen der Funktionalität für das Parsen von lokalen und geteilten Ordnern und deren Analyse würde vielen Unternehmen helfen, ihre eigenen Inhalte besser zu verstehen.
- Ein User-Management-System würde es ermöglichen, benutzerdefinierte Analysen auszuführen und den Service remote zur Verfügung zu stellen.
- Einige der geparsten Informationen werden zurzeit noch nicht in einem Diagramm angezeigt. Beispielsweise die Information über Schlüsselpersonen. Diese Information ermöglicht es, Personen mit unterschiedlichen Tätigkeiten miteinander zu verbinden.
- Der Prompt, welcher für die Analyse verwendet wird, kann optimiert oder erweitert werden, um zusätzliche Informationen zu extrahieren und auszuwerten.

Teil IV

Verzeichnisse

Literaturverzeichnis

[1] Infomaniak Network AG. Infomaniak llm models, 14-12-2024. URL: https://www.infomaniak.com/en/hosting/ai-tools/open-source-models#llm.

[2] Infomaniak Network AG. Infomaniak, 18-12-2024. URL: https://www.infomaniak.com/de.

[3] Daniel bader. schedule, 14-12-2024. URL: https://schedule.readthedocs.io/en/stable/.

[4] containrrr.dev/ Watchtower. Watchtower, 12-12-2024. URL: https://containrrr.dev/watchtower/.

[5] cron job.org. Cronjob, 12-12-2024. URL: https://docs.cron-job.org/.

[6] Figma GmbH. Figma, 4-12-2024. URL: https://www.figma.com/.

[7] Alex Grönholm. Advanced python scheduler, 14-12-2024. URL: https://apscheduler.readthedocs.io/en/stable/.

[8] HackerNoon. Hackernoon, 15-12-2024. URL: https://hackernoon.com.

[9] holger krekel and pytest-dev team. Pytest, 14-12-2024. URL: https://docs.pytest.org/en/stable/.

[10] Astral Software Inc. Ruff, 4-12-2024. URL: https://astral.sh/ruff.

[11] Plotly Technologies Inc. Plotly, 12-12-2024. URL: https://plotly.com/.

[12] Snowflake Inc. Streamlit, 9-12-2024. URL: https://streamlit.io/.

[13] Traefik Labs. Traefik, 12-12-2024. URL: https://doc.traefik.io/traefik/.

[14] mongo express. Mongo-express, 19-12-2024. URL: https://github.com/mongo-express/mongo-express.

[15] Inc MongoDB. Mongodb, 19-12-2024. URL: https://www.mongodb.com/de-de.

[16] Portainer. Portainer, 12-12-2024. URL: https://www.portainer.io.

[17] Sebastián Ramírez. Fastapi, 14-12-2024. URL: https://fastapi.tiangolo.com/.

[18] LLC SolarWinds Worldwide. Pingdom, 11-12-2024. URL: https://tools.pingdom.com.

[19] Superlinked. Vector db comparison, 10-10-2024. URL: https://superlinked.com/vector-db-comparison.

[20] Write the Docs. Documentation principles, 10-10-2024. URL: https://www.writethedocs.org/guide/writing/docs-principles/.

[21] Guido van Rossum, Barry Warsaw, and Alyssa Coghlan. Pep 8 – style guide for python code, 10-10-2024. URL: https://peps.python.org/pep-0008/.

Abbildungsverzeichnis

Tabellenverzeichnis

Glossar

CI/CD (Continuous Integration/Continuous Deployment) beschreibt Prozesse und Praktiken in der Softwareentwicklung, die eine schnelle und automatisierte Bereitstellung von Anwendungen ermöglichen.. 27, 29

LLM Large language model. 3, 6

NFAs nicht-funktionalen Anforderungen. 14

OS Operating System, auf deutsch Betriebssystem. 31, 32

Prompt Eine Eingabe oder Anfrage, die an eine KI oder ein System gestellt wird, um eine spezifische Antwort oder Aktion auszulösen.. 49

Python Type Hints Syntax zur Angabe von Datentypen in Python, die den Code verständlicher machen und Fehler frühzeitig durch statische Analysewerkzeuge erkennen lassen, ohne die Laufzeit zu beeinflussen.. 52