Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Leon Marlo König

Konzeption und Umsetzung eines KI-gestützten Web Scrapers zur Extraktion von Nachrichtenartikeln

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Leon Marlo König

Konzeption und Umsetzung eines KI-gestützten Web Scrapers zur Extraktion von Nachrichtenartikeln

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Wirtschaftsinformatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Stefan Sarstedt
Zweitgutachter: Prof. Dr. Olaf Zukunft

Eingereicht am: 5. Oktober 2023

Leon Marlo König

Thema der Arbeit
Konzeption und Umsetzung eines KI-gestützten Web Scrapers zur Extraktion von Nachrichtenartikeln

Stichworte
Web Scraping, Nachrichtenartikel, künstliche Intelligenz, Python

Kurzzusammenfassung
Diese Bachelorarbeit befasst sich mit der Konzeption und Umsetzung einer Anwendung, die durch den Einsatz von Web Scrapern Nachrichtenartikel aus verschiedenen Quellen extrahiert und aufbereitet. Die Aufbereitung findet hierbei zum Teil durch KI-gestützte Tools statt, durch die eine Zusammenfassung und Übersetzung der Artikeltexte erstellt wird. Das Ziel dieser Anwendung ist es, den Zugriff auf seriöse Nachrichten zu erleichtern und die Hürde für potentielle Leser durch verschiedene Aufbereitungsschritte zu senken.

Leon Marlo König

Title of the paper
Design and implementation of an AI-supported web scraper for the extraction of news articles

Keywords
Web Scraping, News Articles, Artificial Intelligence, Python

Abstract
This bachelor thesis deals with the conception and implementation of an application that extracts and processes news articles from various sources using web scrapers. The processing is partly done through AI-assisted tools, which create summaries and translations of the article texts. The goal of this application is to facilitate access to credible news sources and reduce the barrier for potential readers through various processing steps.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1. Einleitung

Der Großteil der Menschen in Deutschland informiert sich heutzutage über das Internet Pawlik (2023). Dies hat zur Folge, dass Nachrichtenwebseiten einen großen Einfluss auf das Meinungsbild der Bevölkerung haben. Durch die große Nachfrage an Informationen ist die Auswahl an Nachrichten im Internet vielfältig, wobei es starke Unterschiede zwischen den Angeboten gibt. Um die Aufmerksamkeit möglichst vieler Leser zu gewinnen, schrecken viele Nachrichtenwebseiten nicht vor reißerischen Titeln und stark verkürzten Texten zurück, welche möglicherweise zu einem Wachstum an Lesern verhelfen, jedoch nicht die Möglichkeit bieten, komplexe Themen differenziert zu behandeln. Andere, seriöse Nachrichtenwebseiten, die eine solche Berichterstattung anstreben, tun sich vermehrt schwer, in der heutigen Medienlandschaft konkurrenzfähig zu bleiben. Um vermehrt Leser für seriöse, etablierte Medien zu gewinnen, ist es für diese wichtig, gesellschaftliche Trends zu erkennen und dementsprechend zu handeln. Ein solcher Trend ist die Schnelllebigkeit von Informationen und der damit verbundene konstante Informationsfluss, welcher zu Folge hat, dass Leser innerhalb einer kurzen Zeit entscheiden, ob sie einen Artikel lesen oder nicht. Eine gebündelte Ansicht konkreter Informationen zu Artikeln vieler Nachrichtenwebseiten ist hierbei ein möglicher Lösungsansatz.

1.1. Aufgabenstellung und Vorgehen

Das Ziel dieser Arbeit ist die Konzeption und Umsetzung eines KI-gestützten Web Scrapers zur Extraktion von Nachrichtenartikeln. Der Fokus liegt hierbei in gleichen Teilen auf der Entwicklung des eigentlichen Web Scrapers und der Aufbereitung der Nachrichtenartikel durch verschiedene KI-Tools. Die Anwendung soll einen vereinfachten, gebündelten Zugriff auf seriöse, etablierte Medien ermöglichen und zusätzlich die Entscheidungsfindung eines Lesers simpler gestalten. Dies geschieht über die automatische Generierung von Übersetzungen, Schlüsselwörtern und Zusammenfassungen der jeweiligen Artikel. Außerdem wird durch die private Speicherung der Nachrichtenartikel eine Archivierungsfunktion erfüllt, wodurch ein Zugriff auf bereits gelöschte oder veränderte Artikel ermöglicht wird.

Um die Aufgabenstellung zu erfüllen, werden zunächst die Grundlagen zu Web Scrapern erläutert. Hierbei liegt ein besonderer Augenmerk auf der Funktionsweise und den legalen Aspekten. Anschließend werden die technischen und qualitativen Eigenschaften der Datenquellen analysiert und erläutert, um so erste Schlüsse zum Aufbau der Web Scraper zu erhalten. Im selben Kapitel werden ebenfalls die funktionalen Anforderungen und die gewollten Qualitätsmerkmale der späteren Anwendung entwickelt und festgehalten. Im nächsten Schritt wird die grundlegende Struktur der Anwendung anhand von Spezifikationen und Architekturentwürfen festgehalten. Anschließend wird die Implementierung der Spezifikationen erläutert. Hierbei liegt der Fokus auf den verwendeten Tools sowie der konkreten Umsetzung der Web Scraper. Um Aussagen über die Qualität treffen zu können, findet danach eine Evaluation statt, bei der geprüft wird, ob die gestellten Anforderungen befriedigend erfüllt wurden. Zum Schluss wird die Arbeit durch ein Fazit retrospektiv betrachtet.

2. Grundlagen

In diesem Kapitel werden die Grundlagen des Web Scrapings vorgestellt und erläutert. Dies dient dem grundlegendem Verständnis, welches für spätere Kapitel der Arbeit von äußerster Bedeutung ist. Im Folgenden liegt ein besonderer Augenmerk auf der Funktionsweise von Web Scrapern, sowie den rechtlichen und ethischen und Aspekten, die bei der Entwicklung beachtet werden müssen.

2.1. Web Scraping

2.1.1. Definition

Web Scraping ist eine automatisierte Methode, um Daten von Webseiten zu extrahieren. Es bezieht sich auf den Prozess, bei dem ein Computerprogramm oder ein Skript die Informationen von Webseiten sammelt und zur weiteren Analyse vorbereitet beziehungsweise speichert. Die Extraktion der Daten erfolgt entweder über eine direkte Eingabe eines Benutzers oder einen automatischen Bot Zhao (2017). Ein manueller Zugriff durch einen Menschen sowie ein Aufruf durch ein Application Programming Interface (API) zählen hierbei nicht mehr zum Bereich des Web Scraping Mitchell (2018).

Neben Begriffen wie *screen scraping*, *data mining* oder *web harvesting* wird Web Scraping häufig mit *Web Crawling* verglichen bzw. verwechselt. Web Crawler sind automatisierte Bots, die Webseiten analysieren und für Suchmaschinen wie z. B. Google klassifizieren. Diese Klassifikation findet entweder durch Hyperlinks oder eine, vom Web Crawler erstellte, Menge an Schlüsselwörtern statt. Im Gegensatz zum Web Crawling liegt der Fokus beim Web Scraping alleine auf der Extraktion der auf den Webseiten enthaltenen Daten und weniger auf der effizienten Klassifikation dieser. Um die bestmögliche Qualität der extrahierten Daten zu ermöglichen, sind Web Scraper genauestens an die zu analysierende Webseite angepasst und unterliegen somit häufigen Änderungen Mehta u. a. (2020).

2.1.2. Funktionsweise

Der grundlegende Ablauf des Web Scraping ist in den meisten Fällen nahezu identisch und kann in zwei aufeinander folgende Schritte unterteilt werden: das Abrufen von Webressourcen und das Extrahieren der gewünschten Informationen aus den abgerufenen Daten. Zunächst startet ein Web Scraping Programm, indem es eine HTTP-Anfrage erstellt, um Ressourcen von einer bestimmten Website zu erhalten. Die Ressourcen können in verschiedenen Formaten vorliegen, wie z. B. Webseiten im HTML-Format, Datenfeeds im XML- oder JSON-Format oder multimediale Daten wie Bilder, Audio- oder Videodateien. Nachdem die Webdaten heruntergeladen wurden, erfolgt der Extraktionsprozess, um die Daten zu analysieren, neu zu formatieren und strukturiert zu organisieren Zhao (2017). Ein grundlegender Ablauf ist in Abbildung 1 zu sehen. Nach der Extraktion ist eine Vorverarbeitung sowie eine Bereinigung der erhaltenen Daten notwendig, um diese effektiv nutzen zu können. Ein mögliches Vorgehen wird in der Arbeit von Tarannum (2019) diskutiert.

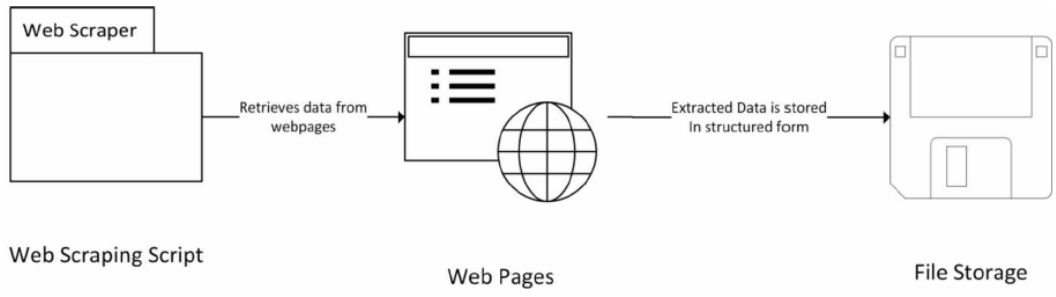

Abbildung 1.: Grundlegender Ablauf: Scraping-Prozess Karthikeyan u. a. (2019)

Die konkrete Funktionsweise eines Web Scrapers wird stark durch das gewählte Design der jeweiligen Webseite beeinflusst, die als Quelle dient. Hierbei lassen sich die meisten Webseiten entweder statisch oder dynamisch geladen unterteilen.
Das Scrapen von statisch geladenen Webseiten stellt in den meisten Anwendungen keine große Herausforderung dar. Statisch geladene Webseiten sind reine HTML- bzw. XML-Dokumente, die nach der initialen Anfrage durch einen Klienten nicht mehr durch die Ausführung von Scripten oder Ähnlichem verändert werden können. Dies bedeutet, dass ein statischer Web Scraper eine simple GET-Anfrage an den Webserver stellen muss, um an alle benötigten Informationen auf der Webseite zu gelangen. Dieses Vorgehen ist effizient und erfordert wenig Rechenleistung um eine große Menge an Daten zu extrahieren.

Das Scrapen von dynamisch geladenen Webseiten hingegen ist in den meisten Fällen weitaus komplizierter. Anders als bei statisch geladenen Webseiten, enthält das von dem Webservern erhaltene HTML- bzw. XML-Dokument noch auszuführende Skripts, die meistens in der Skriptsprache JavaScript geschrieben sind. Die Skripts dienen dem dynamischen Laden von individuellen Inhalten und werden erst auf der Clientseite ausgeführt Bühler u. a. (2018). Dies hat zur Folge, dass die benötigten Daten nicht vollständig auf dem initial erhaltenen Dokument zu finden sind. Ein Scraper für dynamisch geladene Webseiten muss somit ebenfalls in der Lage sein, enthaltene Skripts zu erkennen und auszuführen. Neben einer komplizierteren Architektur bedeutet dies eine niedrige Effizienz und einen, im Vergleich zu statischen Web Scrapern, größeren Verbrauch an Rechenleistung Mehta u. a. (2020).

2.1.3. Techniken

Bei der Entwicklung von Web Scrapern werden viele verschiedene Techniken angewandt. Die Auswahl der verwendeten Techniken hängt stark von den technischen Anforderungen sowie den fachlichen Vorgaben ab. In den meisten Web Scrapern werden mehrere Techniken miteinander kombiniert, was das Extrahieren von Daten auf fast jeder Webseite ermöglicht. Im Folgenden werden die wichtigsten Techniken vorgestellt und kurz erläutert.

Manuelles Kopieren und Einfügen

Bei dieser Technik werden die Daten einer Webseite von einem Benutzer manuell kopiert und in eine Datenbank eingefügt. Für kleinere Datensätze ist dies pragmatisch und häufig die effizienteste Lösung, da eine automatisierte Lösung einen größeren Arbeitsaufwand im Verhältnis zu den gewonnenen Daten bedeuten würde. Wenn es sich jedoch um eine große Menge an Daten handelt, wird diese Methode mühsam und ineffizient Khder (2021).

Reguläre Ausdrücke

Bei der Analyse durch reguläre Ausdrücke wird der gesamte Quelltext einer Webseite durch einen regulären Ausdruck gefiltert, welcher eine Menge an zugelassenen Zeichenfolgen definiert. Dies ist eine vielfältig angewandte und einfache Methode, um Daten zu extrahieren bzw. zu filtern. Das Filtern des Quelltextes basiert intern auf der Verwendung des grep-Befehls von UNIX-Systemen und ist in viele Programmiersprachen integriert Sirisuriya u. a. (2015).

HTML-Analyse

Die HTML-Analyse ist eine häufig verwendete Technik bei der Analyse von statisch geladenen Webseiten. Es werden die benötigten HTML-Tags gefiltert und der gewünschte Inhalt extrahiert. Dies kann durch JavaScript erfolgen, welches lineare HTML-Seiten anspricht und die HTML-Tags von Webseiten somit schneller erkennt. Beim Scrapen von dynamisch geladenen Elementen einer Webseite hingegen ist eine reine HTML-Analyse nicht zielführend und führt möglicherweise dazu, dass unbemerkt ein Teil der Daten nicht extrahiert wird.

Document Object Model Analyse

Bei einer Document Object Model Analyse (DOM-Analyse) benutzt ein Scraper das DOM einer Webseite, welches das Verhalten, den Inhalt, die Struktur und den Stil eines XML-Dokuments widerspiegelt. Durch den Zugriff auf das DOM erhält der Scraper Einblick in die interne Funktionalität einer Webseite. DOM-Analysewerkzeuge in Verbindung mit XPath-gestützten Knoten ermöglichen und, im Vergleich zu anderen Techniken, vereinfachen das Extrahieren von Inhalten. Sogar dynamische Webseiten können von DOM-Analysatoren effizient verarbeitet werden.

XML Path Language

Bei der XML Path Language (XPath) handelt es sich um eine Abfragesprache, mit der einzelnen Elemente eines XML-Dokuments adressiert und ausgewertet werden können, was durch die hierarchische, baumartige Struktur von XML-Dokumenten ermöglicht wird. Ein Web Scraper macht sich dies zu Nutze, indem dieser die Daten gezielt aus den benötigten Knoten extrahiert. In Kombination mit der DOM-Analyse bietet XPath somit ein mächtiges Tool für die Entwickelung evon automatisierten Web Scrapern. Als Erweiterung zu XPath entwickelten Grasso u. a. (2013) OXPath, welches die etablierten Standards um mehrere Funktionen, wie z.B. das Filtern nach visuellen Merkmalen durch den Benutzer, erweitert.

2.1.4. Anwendungsgebiete

Web Scraping findet umfassende Anwendung in verschiedenen Bereichen der Informatik, insbesondere in Feldern wie Business Intelligence, Künstliche Intelligenz, Data Science oder Big Data Nadee und Prutsachainimmit (2018). Neben der kommerziellen Nutzung ist Web Scraping besonders in wissenschaftlichen Anwendungen sehr beliebt, da es die Extraktion großer, personalisierter Datensätze zu minimalen Kosten ermöglicht. Außerdem werden zunehmend veraltete Methoden der Datensammlung fortschreitend durch Einbindung von Web Scrapern ersetzt Khder (2021).

Im Bereich Business Intelligence wird Web Scraping häufig für Preisverfolgung, Marktforschung und Sentiment-Analyse eingesetzt. Vor allem bei sich häufig ändernden Konkurrenzpreisen im E-Commerce erweist sich Web Scraping als wertvoll, da es die automatische Extraktion von Preisinformationen von Konkurrenten ermöglicht und aktuelle Daten in einem einzigen Dokument oder einer Datenbank zugänglich macht. Die Alternative hierzu wäre eine manuelle Überwachung der Preise, welche einen erheblichen Mehraufwand bedeuten würde.

Die durch einen Web Scraper generierten Daten könnten für kommerzielle Operationen einschließlich Marktpreisgestaltung, Trendanalyse, Optimierung des Markteintritts, Forschung und Entwicklung sowie Wettbewerbsüberwachung verwendet werden. Darüber hinaus erleichtert Web Scraping die Analyse des Aktienmarktes durch die Visualisierung von Preisänderungen im Laufe der Zeit und sammelt Kommentare aus sozialen Medien, um die öffentliche Meinung durch z. B. eine Sentiment-Analyse zu verstehen. Die gewonnenen personalisierten Nutzerdaten werden ebenfalls für die Entwicklung verschiedenster Empfehlungssysteme genutzt. Diese wiederum finden Anwendung in personalisierter Werbung und kollaborativer Filterung Diouf u. a. (2019).

Neben der Nutzung für kommerzielle Operationen finden Web Scraper ebenfalls Anwendung in der Meinungsforschung für Politiker in sozialen Medien. Die erhobenen Daten helfen bei der Vorhersage von Wahlergebnissen, da Sentiment-Erkennungsalgorithmen Muster erkennen, die über den Namen des Kandidaten in Tweets und Beiträgen hinausgehen. Im Vergleich zu aggregierten Beiträgen, die z. B. auf der Basis von Twitter-Hashtags erstellt werden, ermöglichen Daten, die durch Web Scraping gesammelt wurden, eine genauere Analyse Nadee und Prutsachainimmit (2018).

Auch im Gesundheitssektor finden Web Scraper mittlerweile Einsatz. Die exportierten Gesundheitsdaten dienen als Grundlage für die Analyse von verschiedenen Methoden, wie z.B. die Anreicherung einer Gen-Gruppe Diouf u. a. (2019).

2.1.5. Legale Aspekte

Die Entwicklung eines Web Scrapers ist häufig umstritten und kann eine Vielzahl an rechtlichen Fragen nach sich ziehen. Außerdem ist zu erwähnen, dass laut Snell und Menaldo (2016) die Rechtmäßigkeit von Web Scraping immer noch zu einem juristischen Graubereich gehört und somit einige Fragen nicht pauschal beantwortet werden können. Im Folgenden werden einige Aspekte und Konsequenzen erläutert, die zu beachten sind. Hierbei wird hauptsächlich auf die Arbeit von Krotov und Silva (2018) Bezug genommen.

Nutzungsbedingungen

Beim automatisierten Zugriff auf Webseiten sind die geltenden Nutzungsbedingungen ein wichtiger Faktor für die Entwicklung von Web Scrapern. In rechtlichen Kreisen herrscht häufig die Meinung, dass der Besitzer einer Webseite die Möglichkeit hat, einen automatisierten Zugriff durch eine Klausel in den Nutzungsbedingungen zu unterbinden. Wenn ein Dritter gegen diese Klausel verstößt, kann dies als ein Vertragsbruch gesehen werden und rechtliche Folgen in Form einer Klage nach sich ziehen. Um dies gewährleisten zu können, muss der Betreiber eines Webscrapers eine ausdrückliche Vereinbarung zur Einhaltung der Nutzungsbedingungen mit dem Betreiber der Webseite eingehen. Aus der Sicht des Webseitenbesitzers ist dies zwar ein valides Mittel, um den automatisierten Zugriff einzudämmen, aber möglicherweise nicht, um jegliche Art zu unterbinden. Es ist z. B. denkbar, dass der Betreiber eines Web Scrapers argumentiert, dass dieser den Nutzungsbedingungen nie offiziell zugestimmt hat.

Urheberrechtlich geschütztes Material

Beim Scrapen von Webseiten kann es dazu kommen, dass Daten oder Informationen, die durch den Inhaber urheberrechtlich geschützt sind, gespeichert und unter Umständen sogar veröffentlicht werden. Dies ist klar als Urheberrechtsverstoß zu bewerten und kann zu einer Klage führen. Hierbei ist es jedoch wichtig zu beachten, dass eine Webseite nicht zwangsläufig alle angezeigten Daten auch besitzt. Ein Beispiel hierfür sind Webseiten, die nutzergenerierte Inhalte anzeigen. Der Betreiber hat in diesem Fall keinen urheberrechtlichen Anspruch auf die von den Benutzern erstellten Daten. Darüber hinaus sind vage Ideen nicht vom Urheberrecht geschützt, lediglich die konkrete Form oder Darstellung dieser kann geschützt sein. Daraus folgt, dass urheberrechtlich geschützte Daten verwendet werden dürfen, um z. B. eine Zusammenfassung im Rahmen einer Analyse dieser zu erstellen. Außerdem ist es möglich, urheberrechtlich geschützte Elemente unter dem Grundsatz der angemessenen Verwendung im

begrenztem Umfang zu nutzen. Dies ist jedoch im Kontext des Web Scrapens eher als schwierig einzuordnen, da meist alle verfügbaren Daten vollständig zwischengespeichert werden müssen.

Wirtschaftlicher Schaden für die Betreiber

Im Rahmen von Web Scraping kann es dazu kommen, dass der Betreiber einer Webseite einen wirtschaftlichen Schaden erfährt. Ein möglicher Grund hierfür ist die unrechtmäßige oder betrügerische Verwendung von extrahierten Daten, welche durch mehrere Gesetze verboten ist. Dies ist der Fall, wenn wissentlich auf kostenpflichtige Inhalte zugegriffen wird, um diese zu verkaufen oder weiterhin über einen nicht autorisierten Kanal abzurufen. Neben einem Einstellungs- und Unterlassungsschreiben kann dies eine straftrechliche Verfolgung nach sich ziehen.
Ein weiterer möglicher Grund für einen wirtschaftlichen Schaden ist die Überlastung eines Webservers durch die vielen Anfragen eines Web Scrapers. Eine zu große Menge an Anfragen kann dazu führen, dass der Server die Anfragen anderer Benutzer langsamer oder, im Extremfall, gar nicht mehr verarbeiten kann. Dies ist funktional ähnlich zu einem Denial-of-Service-Angriff Zhao (2017). Sollte es nachweislich dazu kommen, hat der Eigentümer des Webservers Anspruch auf eine Entschädigung in Höhe der verursachten Kosten. Um einen solchen Schaden zu vermeiden, ist es für den Betrieb eines Web Scrapers zu empfehlen, die Anfragen pro Zeiteinheit zu limitieren. Dieses Limit sollte durch den Betreiber des Webservers festgelegt sein und dessen Ressourcen nicht im besonderen Maße in Anspruch nehmen.
Des Weiteren führen Web Scraping Applikationen häufig dazu, dass weniger Benutzer den Anzeigen auf der Webseite ausgesetzt werden. Dadurch kommt es für die Betreiber zu einer Beeinträchtigung der Monetarisierung von Inhalten. Darüber hinaus ist es denkbar, dass die durch einen Web Scraper extrahierten und z. B. zusammengefassten Inhalte in direkter oder indirekter Konkurrenz zu den Inhalten der gescrapten Webseiten stehen. Dies kann auch ohne eine Verletzung des Urheberrechts geschehen.

2.1.6. Ethische Aspekte

Neben rechtlichen Fragen sind bei der Entwicklung eines Web Scrapers häufig auch ethische Aspekte zu beachten. Diese werden im Folgenden kurz vorgestellt.

Individuelle Privatsphäre

Bei der Verwendung von Webseiten mit personenbezogenen Daten kann es dazu kommen, dass die Privatsphäre der Benutzer unbeabsichtigt beeinträchtigt wird. Ein Abgleich von verschie-

denen Daten kann z. B. dazu führen, dass die Identität eines vermeintlich anonymen Nutzers enthüllt wird. Aber auch ohne eine solche, direkte Verletzung der individuellen Privatsphäre ist die Verwendung und Speicherung von Nutzerdaten durch Dritte als ethisch fraglich einzuschätzen. Neben den möglichen Konsequenzen für die Betreiber von Web Scrapern, können solche Verletzungen der Privatsphäre ebenfalls schwerwiegende Folgen für die Inhaber der Webseiten nach sich ziehen. Diese Problematik wird durch die gestiegenen Bedenken hinsichtlich personenbezogener Daten im Internet nochmals verschärft.

Geschäftsgeheimnisse

Auch Organisationen und Webseitenbetreiber haben das Recht, bestimmte Strategien, Daten oder Geschäftsabläufe vertraulich zu behandeln. Im Rahmen des automatisierten Web Scraping kann es jedoch dazu kommen, dass bestimmte Geschäftsgeheimnisse oder vertrauliche Informationen unabsichtlich bekannt gemacht werden. So kann es dazu kommen, dass durch eine Zusammenfassung von Daten auf Webseiten für Stellenanzeigen die Marktanteile dieser veröffentlicht werden. Außerdem ist es möglich herauszufinden, wie effizient die Daten auf einer Webseite gespeichert werden. Die Veröffentlichung solcher, scheinbar harmloser Informationen kann den Ruf des Unternehmens, welches die Webseite betreibt, nachhaltig schädigen, ohne rechtliche Konsequenzen nach sich zu ziehen Krotov und Silva (2018).

2.1.7. Abwehrmechanismen gegen Web Scraper

Wie bereits in Abschnitt 2.1.5 erwähnt, ist der Einsatz von Web Scrapern, vor allem bei Betreibern von Webseiten, umstritten und meist nicht erwünscht. Aus diesem Grund sind mehrere Mechanismen entwickelt worden, die den Einsatz von Web Scrapen erschweren oder sogar gänzlich verhindern. Die verwendeten Mechanismen können in Offline- und Real-Time-Techniken unterteilt werden, welche im Folgenden kurz erläutert werden.

Offline-Techniken

Bei Offline-Techniken wird erst nach dem eigentlichen Zugriff analysiert, ob es sich tatsächlich um einen unerwünschten, automatisierten Zugriff handelt. Somit kann es unter Umständen einige Zeit dauern, bis ein Web Scraper als ein solcher erkannt wird Doran und Gokhale (2011). Ein Beispiel hierfür ist das Fingerprinting eines Nutzers. Beim Fingerprinting versucht eine Webseite festzustellen, ob es sich um einen menschlichen Nutzer oder einen automatisierten Web Scraper handelt. Eine Ausprägung ist das HTML-Fingerprinting, welches anhand des erhaltenen HTML-Headers versucht dies zu erkennen. Durch Fingerprinting können Be-

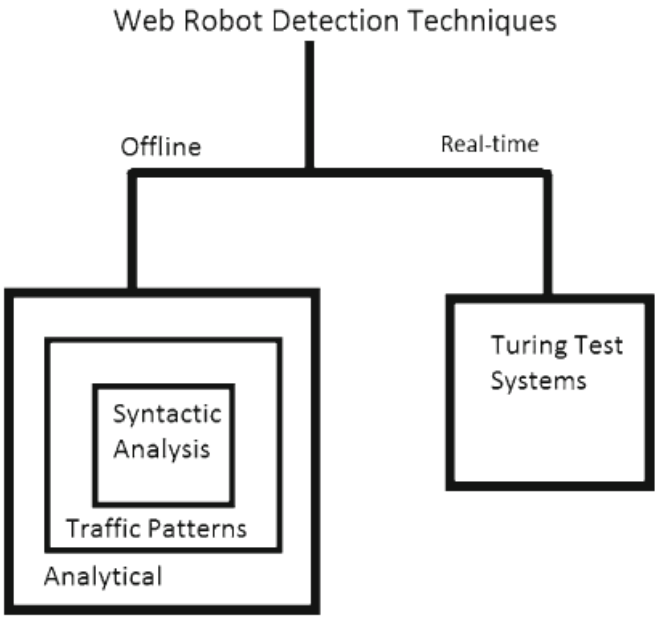

Abbildung 2.: Techniken zur Erkennung von automatisierten Zugriffen Doran und Gokhale (2011)

treiber ebenfalls Datenbanken von Nutzern erstellen und deren Zugriffe aufzeichnen und gegebenenfalls analysieren. Bei auffälligen Verhaltensmustern oder einem abnormal hohen Anfragevolumen können Nutzer limitiert oder geblockt werden Acar u. a. (2013). Neben simplen Metriken und Entscheidungsbäumen finden auch künstliche neuronale Netze immer häufiger Anwendung bei der Analyse von Anfragemustern Doran und Gokhale (2011).

Real-Time-Techniken

Bei Real-Time-Techniken forcieren Webseitenbetreiber jeden Nutzer, eine bestimmte Aufgabe zu absolvieren, bevor dieser auf den eigentlichen Inhalt der Webseite zugreifen kann. Anhand des Ergebnisses dieser Aufgabe soll zwischen menschlichen und automatisierten Anfragen unterschieden werden. Bei den Aufgaben handelt es sich meistens um einen Turing-Test, welcher in der Theorie klar beantworten kann, ob es sich um einen Menschen oder eine Maschine handelt. In der Praxis hingegen ist es für Web Scraper, vor allem durch den Einsatz von künstlicher Intelligenz, möglich geworden, die gestellten Aufgaben zu beantworten, welches jedoch mit einem erheblichen Mehraufwand an Ressourcen verbunden ist. Ein Beispiel für ein häufig angewandtes Turing-Test-System auf Webseiten ist der 2003 vorgestellte *Completely Automated Pulic Turing Test To Tell Computers And Humans Apart*, kurz CAPTCHA. Der

CAPTCHA erhält einen vom Server generierten, simplen Test, welcher durch den Nutzer beantwortet werden muss, um Zugriff auf den gewünschten Inhalt zu erlangen. Eine typische CAPTCHA-Abfrage namens *optical character recognition* (OCR) fordert den Nutzer dazu auf, einen Text aus einem generierten Bild zu kopieren oder den gesprochenen Text aus einer Audiodatei wiederzugeben. Hierbei sind die generierten Bilder oder Audiodateien so konzipiert, dass diese möglichst schlecht von automatisierten Programmen erkannt werden können Doran und Gokhale (2011). Nach der Veröffentlichung des auf dem OCR basierenden Konzepts wurden mehrere Alternativen vorgeschlagen. Darunter auch bildbasierte CAPTCHAs, welche den Nutzer auffordern, Gemeinsamkeiten oder bestimmte Objekte auf den Bildern zu erkennen. Eine weitaus sicherere Alternative bieten interaktive CAPTCHAs. Diese erfordern mehrere komplexe Eingaben eines Nutzers, wie z.B. Drag and Drop, um die gestellten Aufgaben zu lösen. Hierbei ist jedoch zu erwähnen, dass keine CAPTCHA-Alternative einen hunderprozentigen Schutz liefern kann. Es wird lediglich versucht, den Aufwand für die Betreiber eines Web Scrapers in soweit zu erhöhen, sodass sich das Beantworten einer CAPTCHA-Anfrage wirtschaftlich nicht rentieren würde Moradi und Keyvanpour (2015).

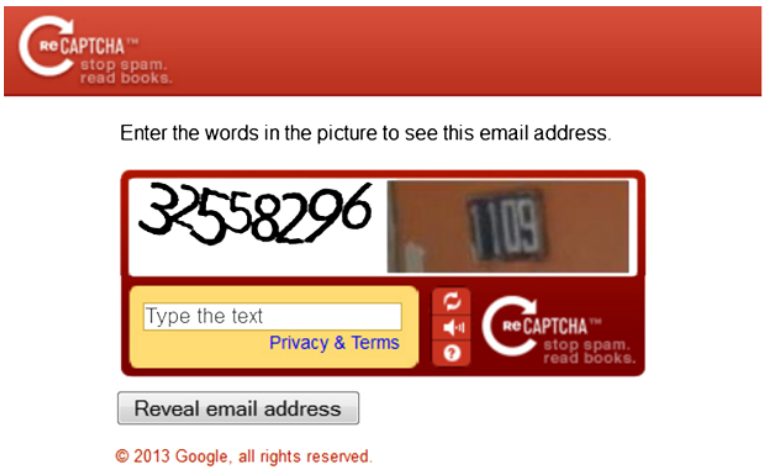

Abbildung 3.: Beispiel für eine OCR-basierte CAPTCHA-Abfrage Moradi und Keyvanpour (2015)

Eine weitere Methode zur Abwehr von Crawlern oder Scrapern ist die Integration des Robots Exclusion Protocol (REP), welches versucht die Aktivität von Bots auf einer Webseite zu regulieren. Das REP ist ein freiwilliges, nicht bindendes Protokoll, welches die Entwicklung von ethischen Scrapern ermöglicht, jedoch böswilligen Zugriff nicht verhindert kann. Das REP wird meistens durch eine Datei namens *robots.txt* implementiert. Die *robots.txt*-Datei befindet sich bei vielen Webseiten im Stammverzeichnis und beschreibt Regeln für den Zugriff von Crawlern,

Scrapern und Bots. Ein Webseitenbetreiber kann dort Pfade festlegen, auf die verschiedene User-Agents nicht zugreifen sollen, um so bestimmte Inhalte vor einem automatisierten Zugriff zu schützen. Die Regeln folgen einer einheitlichen Syntax, damit diese von Bots automatisch gelesen und beachtet werden können. Ein Beispiel für eine simple *robots.txt* ist in Abbildung 4 zu sehen. Da es sich beim REP um eine freiwillige Maßnahme handelt, ist ein vollständiger Schutz vor Zugriffen nicht zu gewährleisten. In wissenschaftlichen und kommerziellen Scrapern sind die Regeln des REP jedoch akzeptiert und werden zunehmend geachtet Giles u. a. (2010).

```
User-agent: *
Disallow: /ardimport/
Disallow: /*~player.html

User-agent: webzip
Disallow: /

User-agent: AhrefsBot
Disallow: /

User-agent: webwhacker
Disallow: /

User-agent: websauger
Disallow: /

User-agent: webcapture
Disallow: /

User-agent: teleport
Disallow: /

User-agent: sitesnagger
Disallow: /

User-agent: net attache
Disallow: /

User-agent: httrack
Disallow: /
```

Abbildung 4.: robots.txt von tagesschau.de

3. Analyse

In diesem Kapitel wird zunächst die Wahl der Datenquellen begründet. Danach werden die Datenquellen auf verschiedene Aspekte analysiert, wobei auf die zuvor behandelten Grundlagen Bezug genommen wird. Anschließend werden die Anforderungen an die zu entwickelnde Anwendung analysiert und festgehalten. Hierbei findet eine Trennung in funktionale Anforderungen und Qualitätsmerkmale statt.

3.1. Analyse der Datenquellen

3.1.1. Enthaltene Informationen

Die Verwendung von Nachrichtenartikeln als Datenquelle ist detailreich und wirft viele mögliche Fragen auf. Darunter auch die Frage nach dem grundlegenden Aufbau sowie der Struktur von verschiedenen Quellen. Um diese analysieren zu können, ist eine Abstraktion über die enthaltenen Informationen notwendig. Die in einem Nachrichtenartikel enthaltenen Informationen gehen meist über den bloßen Inhalt hinaus und lassen sich in viele denkbare Aspekte unterteilen. Diese lassen sich wie folgt zusammenfassen:

- Überschriften, Absatzüberschriften
- textueller Inhalt
- visueller Inhalt
- Datum und Uhrzeit der Veröffentlichung
- Datum und Uhrzeit der letzten Aktualisierung
- Autoren
- Quellen

Eine Abstraktion anhand dieser Informationen ist für die spätere Entwicklung notwendig und ermöglicht eine effiziente Extraktion. Hierbei ist zu beachten, dass der konkrete Aufbau

der verschiedenen Webseiten im höchsten gerade unterschiedlich ist. Somit ist eine individuelle Analyse jeder Einzelnen notwendig. Daraus folgt, dass auch der jeweilig benutzte Scraper eine Individuallösung darstellt und als solche bewertet werden muss. Des Weiteren zieht eine Veränderung des Layouts einer Webseite in den meisten Fällen auch eine Anpassung des jeweiligen Scrapers nach sich. Aus diesem Grund ist ein individueller Scraper ohne Anpassung fast nie für einen längeren Zeitraum anwendbar.

3.1.2. Kriterien zur Auswahl von Datenquellen

Die Auswahl von möglichen Datenquellen erfordert eine genaue Analyse von verschiedenen Gegebenheiten, die im Folgenden kurz erläutert werden. Hierbei ist zu bemerken, dass die rechtlichen und ethischen Richtlinien zumeist die limitierenden Faktoren darstellen. Bei der technischen Umsetzbarkeit und der journalistischen Qualität ist in den meisten Fällen ein gewisser Spielraum vorhanden.

Technische Umsetzbarkeit

Die konkreten, oben genannten, Informationen eines Nachrichtenartikels sind in den meisten Fällen direkt in das HTML-Dokument einer Webseite integriert. Dies ist jedoch nur bei statisch geladenen Webseiten der Fall und trifft bei dynamisch geladenen Webseiten nur unter Vorbehalt zu. Außerdem ist es zu beachten, dass neben den gewollten Informationen des Artikels viele weitere, für die Analyse unwichtige Elemente und Textabschnitte auf dem HTML-Dokument zu finden sind. Um eine zielgerichtete Extraktion von Informationen zu ermöglichen, ist es somit notwendig, das vollständige HTML-Dokument zu laden, um anschließend die gewollten und ungewollten Abschnitte technisch zu unterscheiden und im Folgeschritt zu filtern. Dies kann sich, je nach Struktur und Logik der Webseite, als eine mögliche Herausforderung darstellen.

Rechliche und ethische Richtlinien

Neben den technischen Aspekten, die bei der Extraktion von Nachrichtenartikeln zu beachten sind, spielen legale und ethische Richtilinien ebenfalls eine große Rolle. Wie bereits in Abschnitt 2.1.5 grundlegend erläutert wurde, ist die Entwicklung eines legalen Web Scrapers zwangsläufig mit einer genauen Analyse dieser verbunden. Im Kontext der Extraktion von Nachrichtenartikeln stellt dies, aufgrund vieler verschiedener legaler Aspekte, eine komplexe Aufgabe dar. So ist es zwangsläufig notwendig, die *robots.txt* sowie die Nutzungsbedingungen der einzelnen Webseiten vollständig zu lesen. Sollten diese den Zugriff auf benötigte Ressour-

cen, oder sogar Web Scraping als Ganzes, verbieten, ist es demnach nicht mehr möglich, diese als Datenquelle zu benutzen.

Journalistische Qualität und politische Ausrichtung

Ein weiterer wichtiger und nicht zu vernachlässigender Punkt ist die fachliche Korrektheit bzw. die journalistische Qualität des konkreten Textes. Ohne die Berücksichtigung dieses Punktes ist das endgültige Produkt zwar technisch, rechtlich und ethisch korrekt, jedoch inhaltlich mangelhaft und in der Praxis unbrauchbar. Eine Einschätzung der journalistischen Qualität sowie der politischen Ausrichtung ist hierbei zum Teil subjektiv und bietet Raum für Interpretation.

Verwendete Datenquellen

Unter der Berücksichtigung der oben genannten Aspekte werden im Rahmen dieser Arbeit die Webseiten *tagesschau.de*, *euronews.com* und *politico.eu* als Datenquellen verwendet.

3.1.3. Technische Aspekte

Wie bereits erwähnt ist die technische Umsetzung der einzelnen Datenquellen sehr individuell und unterscheidet sich teilweise grundlegend. Trotzdem ist eine generelle Abstraktion möglich. Der Zugriff auf eine der oben genannten Webseiten kann in zwei technische Schritte unterteilt werden.

Aufruf der Suchfunktion

Bei dem Abruf der Suchfunktion werden die Artikel anhand von Kriterien wie eines Suchbegriffs oder eines Datums gefiltert und sortiert. Die technische Umsetzung der jeweiligen Suchfunktion ist, wie bereits erwähnt, höchst unterschiedlich und erfordert eine individuelle Lösung. Eine Suchfunktion kann entweder statisch oder dynamisch realisiert werden. Bei statischen Suchfunktionen wird meist eine festgelegte maximale Anzahl an Artikeln pro HTML-Dokument geladen. Eine konkrete Implementierung bietet häufig eine Seitennummerierung, die alle Elemente auf der Seite neu lädt. Bei dynamischen Suchfunktionen werden bei Bedarf mehr Artikel in das bereits vorhandene HTML-Dokument geladen. Häufig wird dies durch eine „Mehr Laden“-Funktion gelöst. Hierbei ist eine Kombination aus beiden Ansätzen denkbar. Neben der konkreten technischen Umsetzung der jeweiligen Suchfunktion sind weitere Aspekte zu beachten. Hierunter fallen unter anderem Pop-up-Fenster, Sprache oder mögliche

interne API-Aufrufe. Diese beeinflussen im späteren Verlauf die konkrete Implementation und sind somit ebenfalls von Bedeutung.

Aufruf der Artikel

Die durch die Suchfunktion gefundenen Artikel beinhalten zumeist die eigentlich benötigten Daten. Diese liegen innerhalb des jeweiligen HTML-Dokuments vor und müssen gefiltert werden. Hierbei ist es ebenfalls notwendig, die grundlegende Struktur jeder Datenquelle genauestens zu analysieren. Genau wie bei der Suchfunktion können die Webseiten der Artikel entweder statisch oder dynamisch geladen werden, wobei eine Mischform ebenfalls denkbar ist.

Gegebenheiten der Datenquellen

Eine Analyse der Gegebenheiten der drei Datenquellen ergibt folgendes Ergebnis:

	Tagesschau	EuroNews	Politico
Suchfunktion	Statisch	Statisch	Dynamisch
Artikelseite	Statisch	Statisch	Statisch
Sprache	Deutsch	Englisch	Englisch
Cookies	Nein	Ja, akzeptieren nicht notwendig	Ja, akzeptieren bei Suchfunktion
Interner API-Aufruf	Ja	Nein	Nein

Tabelle 3.1.: Technische Details der Datenquellen

Den Ergebnissen nach zu urteilen, sind die technischen Gegebenheiten der Datenquellen Tagesschau und EuroNews als ähnlich zu bewerten. Bei der Datenquelle Politico sieht dies anders aus. Aufgrund der dynamisch geladenen Suchfunktion wird ein erdachter Scraper zusätzlich in der Lage sein müssen, dynamische Elemente mit Hilfe von JavaScript zu laden. Außerdem ist es wichtig zu erwähnen, dass sich die Einteilung in statisch oder dynamisch ausschließlich auf die benötigten Informationen bezieht. Somit werden Elemente, die im Rahmen dieser Arbeit nicht von Bedeutung sind, bei der Einteilung nicht berücksichtigt. Beispiele hierfür sind Bilder, Videos oder Werbeanzeigen.

3.2. Anforderungsanalyse

Nach der vorhergegangenen Analyse der zu verwendenden Datenquellen werden nun auf Basis dieser die Anforderungen an das zu entwickelnde System konzipiert. Bei der Anforderungsanalyse werden sowohl die gewollten Eigenschaften eines Systems als auch die Ziele und Wünsche von Benutzern in strukturierter Form festgehalten. Diese lassen sich in funktionale Anforderungen oder Qualitätsanforderungen, welche häufig auch als nichtfunktionale Anforderungen bezeichnet werden, unterteilen Pohl (2010).

3.2.1. Funktionale Anforderungen

Im Folgenden werden die funktionalen Anforderungen an das System genannt und kurz erläutert. Unter funktionalen Anforderungen versteht man die Beschreibung der geplanten Funktionalitäten, welche das geplante System zur Verfügung stellen soll. Die unten genannten funktionalen Anforderungen wurden in geschachtelter Form präsentiert, welches eine Abhängigkeit oder eine genauere Spezifikation darsterstellt. Pohl und Rupp (2021)

F1: Suchfunktion

Das System soll dem Nutzer die Möglichkeit geben, durch die Übergabe von Attributen, die Extraktion und Ausgabe der Nachrichtenartikel zu limitieren bzw. zu filtern. Die auswählbaren Attribute sind:

- Suchbegriff
- Nachrichtenquellen
- Sprache
- Anpassbare Zeitspanne
- Vorbestimmte, relative Zeitspanne

F1.1: Validierung der Eingaben

Das System soll die durch den Nutzer getätigten Eingaben auf Vollständigkeit und Korrektheit überprüfen. Kommt es zu einem Fehler, muss dies dem Nutzer erkenntlich gemacht werden.

F2: Extraktion von Nachrichtenartikeln

Das System soll auf Basis der gewählten Attribute die Informationen der gefundenen Nachrichtenartikel automatisch extrahieren.

F2.1: Persistente Speicherung

Das System soll alle extrahierten Informationen persistent speichern und eine Überprüfung durchführen, ob diese bereits gespeichert wurden.

F2.2: Übersetzung der extrahierten Informationen

Das System soll alle extrahierten Informationen in die von Benutzern spezifizierte Sprache übersetzen.

F2.3: Erstellung von Schlüsselwörtern

Das System soll für jeden Artikel eine Liste von Schlüsselwörtern erstellen.

F2.4: Erstellung einer Zusammenfassung

Das System soll für jeden Artikel eine Zusammenfassung in der vom Nutzer gewünschten Sprache erstellen.

F2.5: Verarbeitungsstatus anzeigen

Das System soll den Nutzer über den aktuellen Status der Verarbeitung in Kenntnis setzen.

F3: Ausgabe der Ergebnisse

Das System soll dem Nutzer die extrahierten Artikel in tabellarischer, sortierter Form anzeigen.

F3.1: Detailansicht eines Ergebnisses

Der Nutzer kann durch die Auswahl eines Artikels eine Detailansicht öffnen, welche die Zusammenfassung sowie den übersetzten Text enthält.

F4: Nebenläufigkeit

Das System soll gleichzeitig auf die verschiedenen Quellen zugreifen und extrahierte Informationen nebenläufig verarbeiten.

3.2.2. Qualitätsanforderungen

Die Qualitätsanforderungen beschreiben die gewollten, qualitativen Merkmale des zu entwickelnden Systems. Hierbei haben diese einen großen Einfluss auf die Gestalt der späteren Architektur. Der Einfluss übersteigt meist auch den der funktionalen Anforderungen und ist somit, wie es häufig der Fall ist, nicht zu unterschätzen Pohl und Rupp (2021).

Die Qualität eines Systems festzustellen ist meist eine Aufgabe, welche stark durch Subjektivität beeinflusst wird. Dennoch lassen sich verchiedene Qualitätsmerkmale in Kategorien unterteilen. Eine mögliche Unterteilung ist durch die Norm ISO/IEC 25010:2011 gegeben. Diese bietet acht mögliche Kategorien, welche wiederum in kleiner Untergruppen unterteilt werden können. Die acht konzipierten Kategorien sind in Abbildung 5 zu sehen. Bei der Konzeption von Qualitätsanforderungen wird es empfohlen, drei bis fünf Kategorien als Architekturziele festzulegen und diese genauer zu konkretisieren. Diese Einschränkung trägt dazu bei, dass die Ziele des zu entwickelnden Systems genauer definiert werden können ISO/IEC25010:2011 (2011).

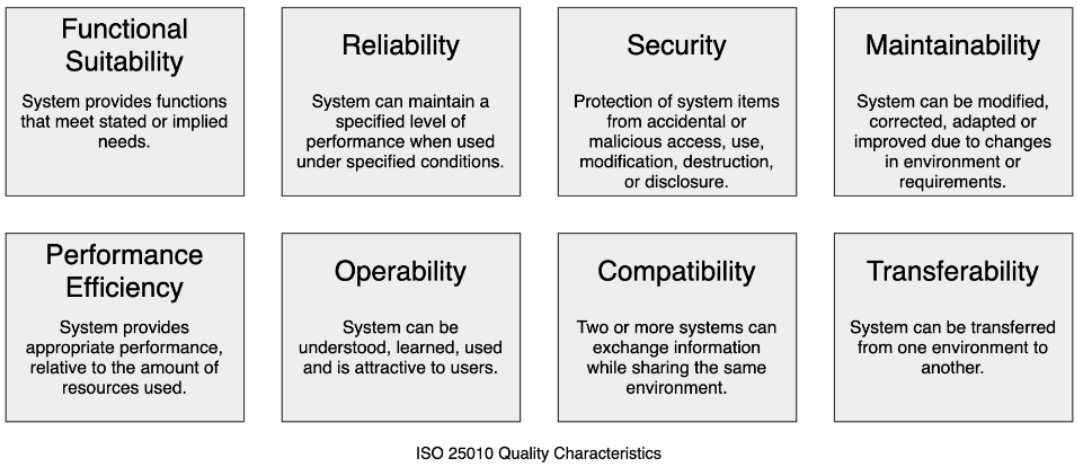

Abbildung 5.: Qualitätsmerkmale laut ISO/IEC 25010:2011 Starke und Hruschka (2023)

Q1: Bedienbarkeit

Die Bedienbarkeit soll bei der Entwicklung des Systems besonders im Fokus stehen, da ein Großteil der Anwender keine tiefgründigen IT-Kenntnisse besitzen werden. Um eine zufriedenstellende und intuitive Bedienbarkeit zu gewährleisten, erfolgt die Steuerung des Systems durch eine grafische Benutzeroberfläche. Ein strukturierter Aufbau, der nur relevante Informationen enthält, soll dafür Sorgen, dass ein potentieller Nutzer die Anwendung in weniger als einer Minute bedienen kann.

Q2: Effiziente Leistung

Das System soll, relativ zur erbrachten Leistung, eine effiziente Nutzung der vorhandenen Ressourcen vorweisen. Dies ermöglicht es, die Anwendung auch auf Systemen mit wenig Rechenleistung zu verwenden. Das System sollte bei der Benutzung maximal 2 GB Arbeitsspeicher verbrauchen.

Q3: Wartbarkeit

Das System soll eine simple Wartung erlauben. Dies ist besonders mit Hinblick auf den sich ständig ändernden Aufbau von Webseiten zu betrachten. Außerdem soll die Anwendung es ermöglichen, Erweiterungen, wie z. B. neue Scraper, schnell und effizient hinzuzufügen. Ein Entwickler soll nicht länger als 30 Minuten brauchen, um einen bereits entwickelten Scraper in die funktionierende Anwendung hinzuzufügen.

4. Konzeption

Nach der Analyse der Datenquellen und der Beschreibung der Anforderungen, werden in diesem Kapitel die getroffenen Entwurfsentscheidungen vorgestellt. Die Konzeption wird hierbei in Spezifikation und Architektur getrennt und auf Basis der zuvor erstellten funktionalen Anforderungen, sowie der gewählten Qualitätsmerkmale entwickelt.

4.1. Spezifikation

Für ein generelles Verständnis für die Funktionsweise der erdachten Anwendung ist es von großer Bedeutung, die fachlichen und visuellen Aspekte dieser genauer zu beleuchten. Erst wenn diese verstanden und entworfen wurden, ist es sinnvoll, die konkrete, technische Sicht auf die Anwendung zu konzipieren, da diese die späteren Architekturentscheidungen im großen Maße prägen.

4.1.1. Fachliches Datenmodell

Die fachlichen Komponenten der Anwendung werden durch ein fachliches Datenmodell genauer beschrieben. Das in Abbildung 6 gezeigte Modell enthält keine technischen Klassen, sondern dient lediglich zur Visualisierung der fachlichen Aspekte, welche in den Anforderungen in Kapitel 3.2 festgehalten wurden. Bei der Betrachtung des Modells ist deutlich zu erkennen, dass das Scraper-Modul eine zentrale Rolle spielt, da diese die Artikel erstellt. Hierbei ist zu erwähnen, dass es maximal drei Scraper-Objekte geben wird. Dies ist mit der Anzahl an Quellen zu erklären. Das Gleiche gilt ebenfalls für die Objekte der Suchwebseiten. Die Klasse Suchbeschreibung ist dafür angedacht, die Eingaben eines Benutzers zu modellieren. Außerdem ist bereits eine Teilung in Such- und Extraktionsfunktionen klar zu erkennen, wobei die Klassen rund um den URLGetter zur Suchfunktion und der Rest, rund um den Scraper, zur Extraktionfunktion gehören.

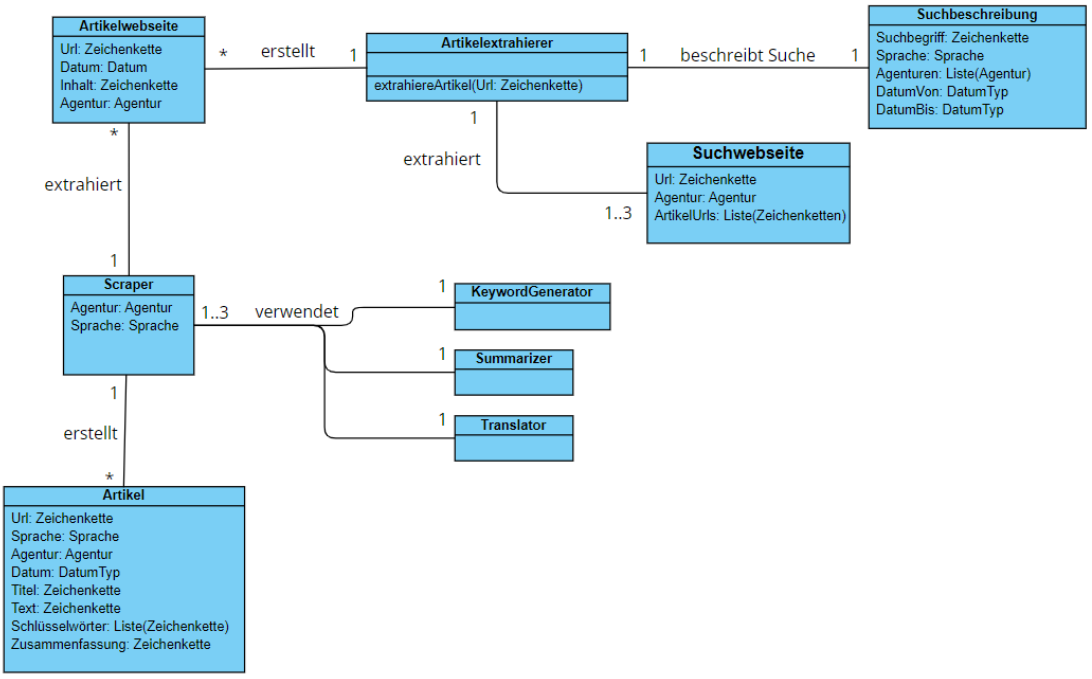

Abbildung 6.: Fachliches Datenmodell der Anwendung

4.1.2. Datenbankentwurf

Der Datenbankentwurf spielt bei der Konzeption eine wichtige, aber denkbar simple Rolle. Ein ER-Modell der angedachten Datenbanktabellen ist in Abbildung 7 zu sehen. Da keine Login-Funktion oder Ähnliches angedacht wird, dient die Datenbank lediglich der Speicherung von extrahierten Artikeln. Dies resultiert in einer geringen Anzahl an Tabellen, die jedoch einen möglichst effizienten Zugriff erlauben müssen. Aus diesem Grund wird die in Abbildung 6 erdachte Artikel-Klasse in die zwei Tabellen *Artikel* und *Artikel_Data* unterteilt. Die *Artikel*-Tabelle enthält die Meta-Daten und die *Artikel_Data*-Tabelle enthält, neben der Sprache, die eigentlichen Informationen eines Artikels. Diese Aufteilung verhindert die redundante Speicherung und Verarbeitung von Artikeln, die in mehreren Sprachen vorhanden sind. Folglich hat ein Artikel, der in drei Sprachen übersetzt wurde, einen Eintrag in der *Artikel*-Tabelle und drei Einträge in der *Artikel_Data*-Tabelle, die auf diesen verweisen. Dies geschieht durch die Verwendung eines Fremdschlüssels im Attribut *artikel*. Um einen effizienteren Zugriff auf die gespeicherten Artikel gewährleisten zu können, wurden die Attribute *artikel_id*, *url*, *artikel_data_id*, und *artikel* mit einem Index versehen. Der Index auf dem Attribut *url* ist hierbei

von besonderer Bedeutung, da dieser später zur Unterscheidung von Artikeln benutzt werden könnte, damit diese nicht redundant gespeichert werden können.

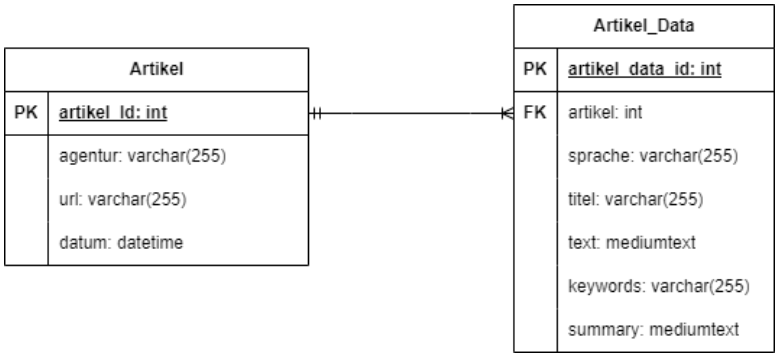

Abbildung 7.: ER-Modell der Datenbank

4.1.3. Grafische Benutzeroberfläche

Die grafische Benutzeroberfläche beeinflusst die, durch die Nutzer wahrgenommene, Benutzbarkeit der Anwendung im großen Stil. Aus diesem Grund ist es wichtig, eine simple und intuitive Benutzung zu gewährleisten. Die Oberfläche wird visuell klar nach Funktionen geteilt, wobei der Anwender immer nur die Informationen sieht, die zum jeweiligen Zeitpunkt von Bedeutung sind. Außerdem werden die Bedienungselemente der Oberfläche in simpler, englischer Sprache dargestellt, was eine potentielle Sprachbarriere verhindert und so die Anzahl an potentiellen Nutzern erhöht.

Hauptansicht

Die Hauptansicht ist in Abbildung 8 zu sehen. Visuell lässt sich diese in drei Komponenten aufteilen: die Suchfunktion am linken Rand, die Informationsansicht im Zentrum und die Statuszeile am unteren Rand der Anwendung. Die Suchfunktion bietet dem Nutzer die Möglichkeit, die Artikel nach verschiedenen Kriterien zu filtern. Hierzu gehören: die Suchbegriffe (Search Term), die Quellen (Agency), die Sprache (Language) und das Datum (Date from/to) bzw. die Zeitspanne (Time range). Bei der Verwendung einer Zeitspanne soll ein Nutzer zwischen heute (today), letzter Woche (last week) und letztem Monat (last month) auswählen können. Nach der Auswahl an Filterparametern kann der Nutzer auf den Search-Button klicken, welches die Extraktion und Verarbeitung der Nachrichtenartikel anstößt. Während dieses Prozesses soll der Status der Verarbeitung in der Statusleiste angezeigt werden. Nachdem dieser Prozess abgeschlossen ist, werden die Artikel in der Informationsansicht angezeigt. Die Artikel werden

als Liste dargestellt und haben die Attribute: Agentur (Agency), Titel (Title), Schlüsselwörter (Keywords) und Datum (Date). Ein Nutzer kann mit Hilfe einer Scrollleiste am rechten Rand durch den Artikel navigieren. Durch einen Doppelklick auf einen Artikel gelangt der Nutzer zur Detailansicht des jeweiligen Artikels.

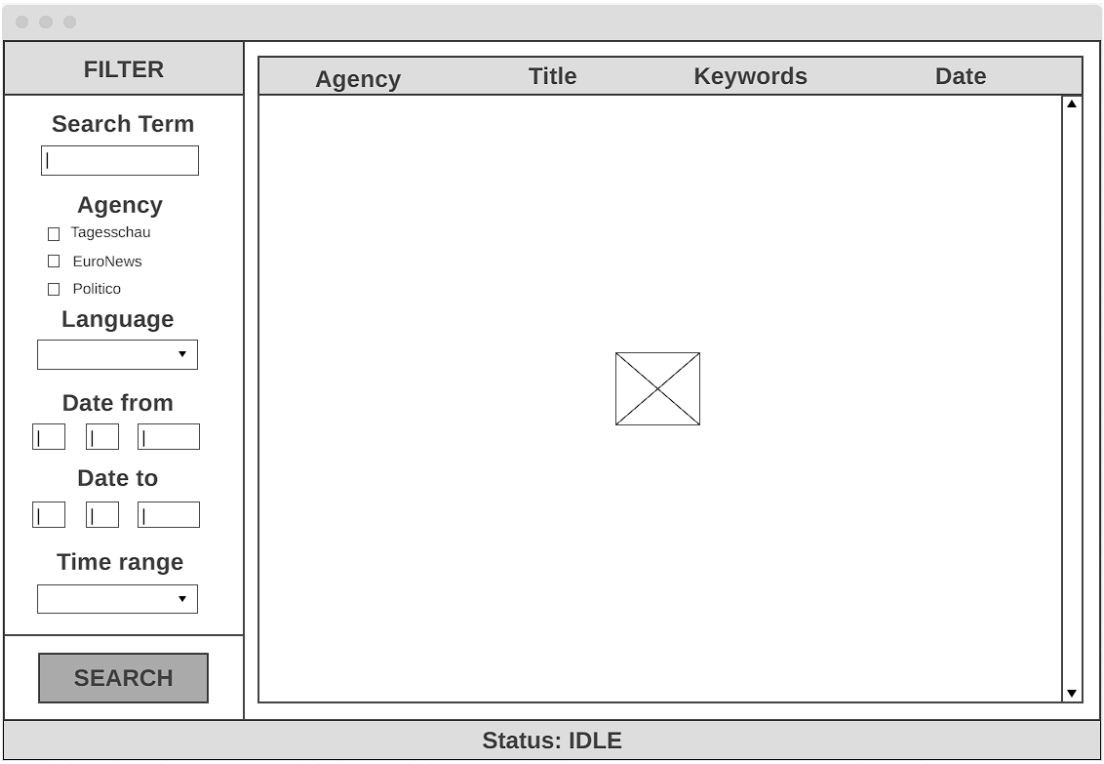

Abbildung 8.: Hauptansicht der Anwendung

Detailansicht

Die Detailansicht ist in Abbildung 9 zu sehen. Diese öffnet sich nach einem Doppelklick in einem neuen Fenster und enthält am oberen Rand die grundlegenden Informationen: Titel (Title), Agentur (Agency), Datum (Date) und URL. Die eigentliche URL wird hierbei nicht angezeigt, sondern durch einen klickbaren Hyperlink ersetzt, welcher den Artikel in der ursprünglichen Sprache im Browser öffnet. Im Zentrum der Detailansicht ist zu Beginn die Zusammenfassung des Artikels zu sehen. Nach einem Klick auf den Button in der unteren, rechten Ecke, wird die Zusammenfassung durch den vollständigen, gegebenenfalls übersetzten, Text des Artikels ersetzt.

Abbildung 9.: Detailansicht eines Artikels

4.2. Architektur

4.2.1. Architekturmuster

Die Architektur eines Systems hat großen Einfluss auf die späteren Eigenschaften des fertigen Produktes. Die konkret gewählte Architektur wird im besonderen Maße durch die bereits erstellten, funktionalen Anforderungen und die Qualitätsziele beeinflusst Posch u. a. (2012). Je nach Anforderungen kommen verschiedene Architekturmuster in Frage. Architekturmuster sind etablierte Lösungsansätze für spezifische Problemstellungen, die bei der Gestaltung von Systemen berücksichtigt werden sollten, da sie sich bereits in verschiedenen Systemen als effektiv erwiesen haben Goll und Dausmann (2014). Da es sich im Rahmen dieser Arbeit um eine Anwendung mit einer grafischen Benutzeroberfläche handelt, kommen einige, bereits erprobte, Architekturmuster in Frage. Darunter Model-View-Controller (MVP), Model-View-ViewModel (MVVM) und Model-View-Presenter (MVP). Aufgrund der spezifischen Anforderungen der er-

dachten Anwendung ist die Wahl auf das MVP-Muster gefallen, welches aus den Komponenten Model, View und Presenter besteht.

Model

Das Modell ist hierbei für die Geschäftslogik verantwortlich und enthält in der Regel die für die Anwendung relevanten Daten und die dazugehörigen Methoden zum Ändern und Abfragen dieser. Falls die Daten persistent innerhalb einer Datenbank gespeichert werden, erfolgt der Zugriff ebenfalls über das Modell, welches eine Abstraktion der Datenbank bietet.

View

Die View stellt die Basis für die visuelle Darstellung der Software dar. Sie baut die grafische Oberfläche auf und verändert diese. Außerdem ist die View dafür verantwortlich, dass Aktionen von Benutzern registriert und an weitere Komponenten weitergegeben werden.

Presenter

Der Presenter ist im weitesten Sinne das Bindeglied zwischen dem Model und dem View. Die im View registrierten Benutzeraktionen werden an den Presenter weitergegeben, welcher den konkreten Ablauf auf diese steuert und die im Modell enthaltenen Daten daraufhin abfragt oder ändert. Die Interaktionen zwischen den genannten Komponenten im MVP-Muster sind in Abbildung 10 zu sehen. Bei dem verwendeten View handelt es sich um einen sogenannten passiven View, da dieser keine direkte Verbindung mehr zu dem Model hat. Die View kann sich somit nicht mehr selbst aktualisieren und ich bin vollständig vom Presenter abhängig. Daraus resultiert eine strenge Kapselung der grafischen Elemente und der Datenhaltung. Diese Kapselung führt, neben einer klareren Verteilung der Aufgaben, zu einer verbesserten Testbarkeit des gesamten Systems, insbesondere durch Mock-Testing. Als Nachteil ist jedoch anzuführen, dass nun neben einem Großteil der Business-Logik auch die Logik des Views komplett innerhalb des Presenters umgesetzt werden muss. Dies führt zu einem Mehraufwand beim Entwickeln und, unter Umständen, zu einer Gottklasse Fowler.

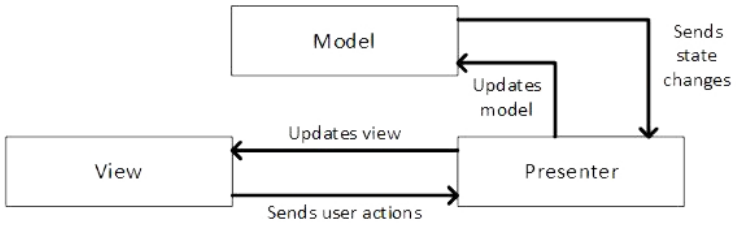

Abbildung 10.: MVP-Muster Ingeno (2018)

4.2.2. Bausteinsicht

Im folgenden Abschnitt wird die erdachte Architektur anhand einer Bausteinsicht dargestellt und kurz erläutert. Bei der Bausteinsicht werden die Elemente einer Anwendung in White- und Blackboxes eingeteilt. Zu Beginn wird das System im Kontext mit anderen Systemen dargestellt. Danach wird das System im ersten Level in White- und Blackboxes eingeteilt. Diese Einteilung ermöglicht eine Übersicht über die Komponenten und deren Beziehungen, ohne unnötige Details anzuzeigen. Mit jedem zusätzlichen Level werden die auf dem vorherigen Level enthaltenen Blackboxes geöffnet und detailierter dargestellt. Die Erstellung der Bausteinsicht erfolgt auf Grundlage des arc42-Templates Starke und Hruschka (2023).

Umfang & Kontext

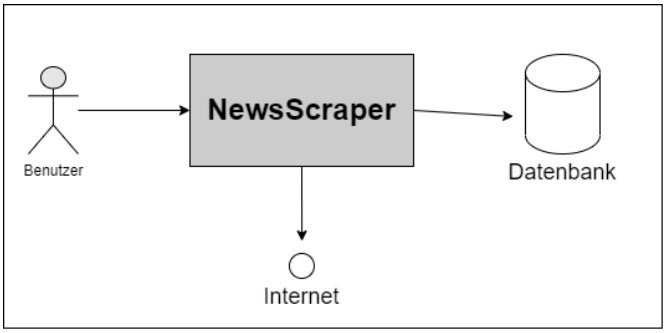

Abbildung 11.: Umfang & Kontext der Anwendung

Die Anwendung, die im Folgenden als NewsScraper bezeichnet wird, besitzt wenige Schnittstellen zu externen Systemen. Es erfolgt lediglich ein Zugriff auf eine Datenbank sowie das Internet.

Level 1

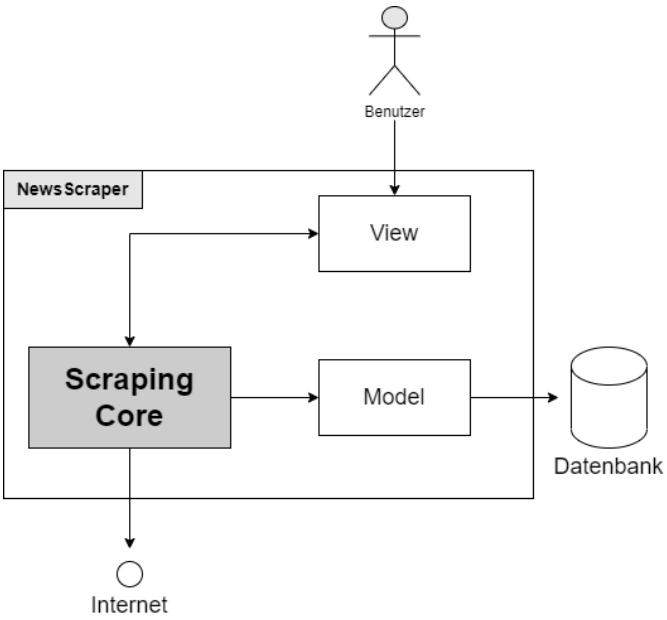

Abbildung 12.: Level 1 mit NewsScraper als Whitebox

Die Level 1 Ansicht zeigt die grundlegende Struktur der Anwendung. Wie bereits zu Beginn des Kapitels erwähnt, wird als Architektur das MVP-Muster verwendet, wobei der Presenter durch eine weitere Blackbox gekapselt wird.

Blackbox	**Beschreibung**
View	Basis für visuelle Darstellung. Registriert Nutzereingaben und gibt diese weiter.
Model	Abstraktion der Datenbank. Bietet Methoden zum Exportieren und Verändern der Daten.
ScrapingCore	Erhält Benutzereingaben vom View und interpretiert diese. Extrahiert und verarbeitet die Nachrichtenartikel und reicht diese anschließend an das Model.

Tabelle 4.1.: Die im Level 1 enthaltenen Blackboxes

Level 2

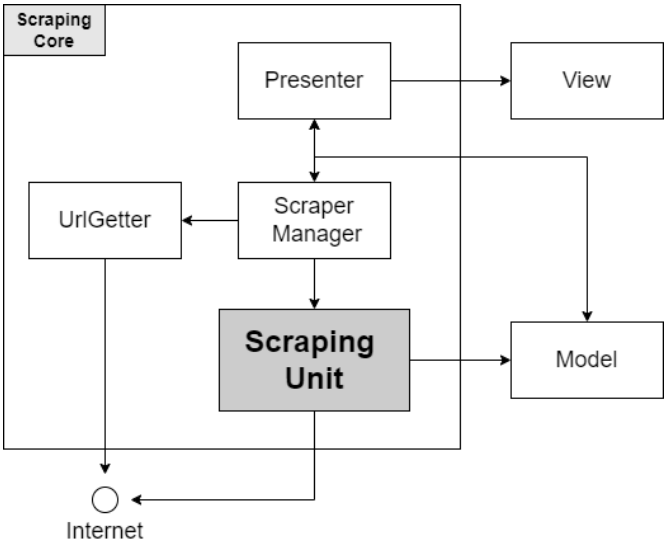

Abbildung 13.: Level 2 mit Scraping Core als Whitebox

Blackbox	**Beschreibung**
Presenter	Reagiert auf die Benutzereingaben des Views und überprüft diese auf Korrektheit. Bestimmt, was im View angezeigt wird. Delegiert Aufgaben an den ScraperManager.
ScraperManager	erhält die Suchanfrage vom Presenter und reicht diese an den UrlGetter weiter. Startet mit erhaltenen URLs die ScrapingUnit. Dient als Bindeglied zwischen Presenter und ScrapingUnit.
UrlGetter	Besucht die Suchwebseiten der gewählten Quellen und extrahiert die benötigten URLs auf Basis der Suchanfrage.
ScrapingUnit	Extrahiert, auf Basis der vom ScraperManager erhaltenen URLs, die benötigten Informationen von den Webseiten. Nach der Verarbeitung werden die erstellten Artikel an das Modell gereicht.

Tabelle 4.2.: Die im Level 2 enthaltenen Blackboxes

Level 3

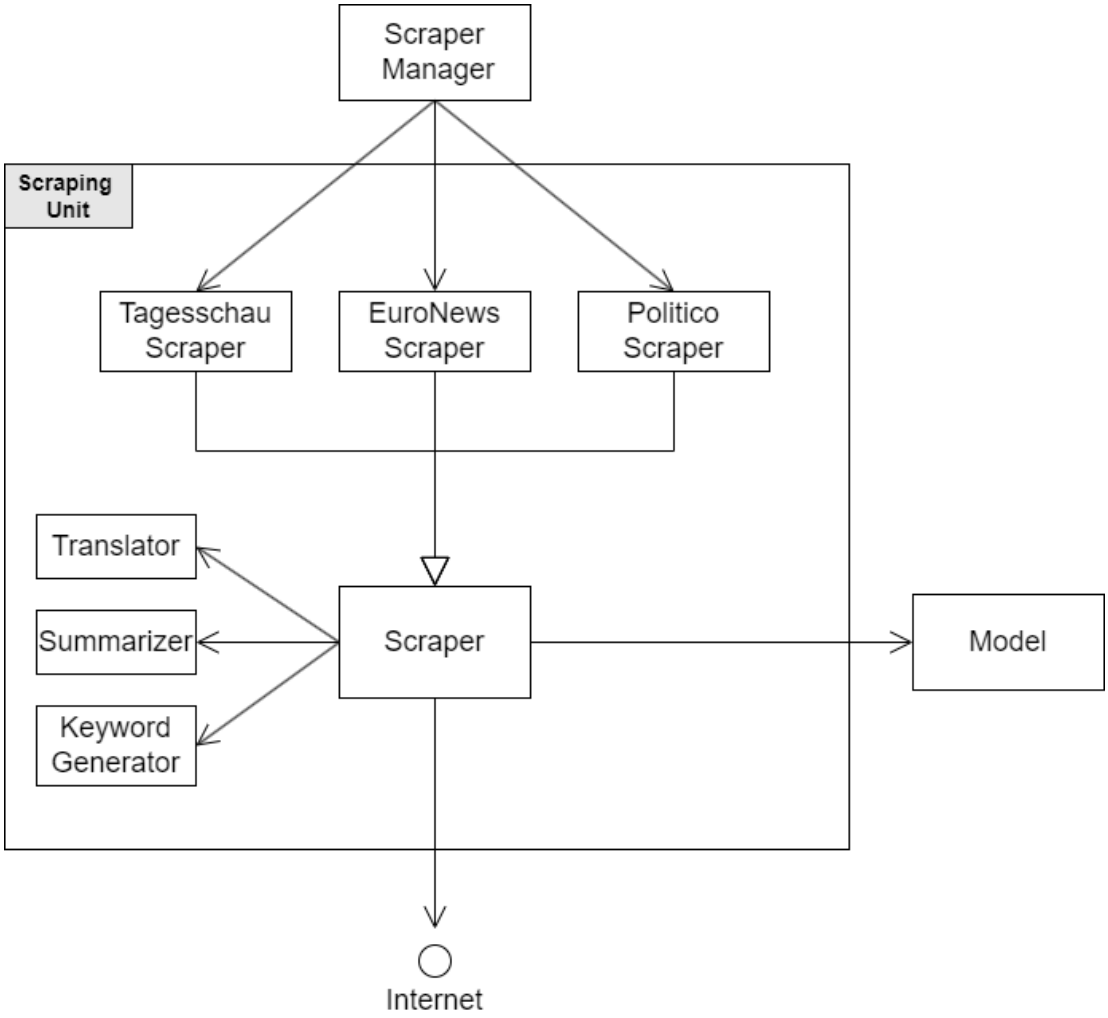

Abbildung 14.: Level 3 mit Scraping Unit als Whitebox

Die in Level 3 zu sehenden Klassen TagesschauScraper, EuroNewsScraper und PoliticoScraper erben ebenfalls von der Klasse Thread. Dies ist aus Gründen der Übersicht nicht angegeben, spielt aber bei der späteren Implementierung sowie bei der Laufzeit eine große Rolle.

Blackbox	Beschreibung
TagesschauScraper	Extrahiert Informationen von Tagesschau-Artikeln. Erweitert Scraper-Modul.
EuroNewsScraper	Extrahiert Informationen von EuroNews-Artikeln. Erweitert Scraper-Modul.
PoliticoScraper	Extrahiert Informationen von Politico Artikeln. Erweitert Scraper-Modul.
Scraper	bietet Basisfunktionen für jeden Scraper an. Verwendet den Translator, Summarizer und KeywordGenerator zur Verarbeitung der extrahierten Informationen. Reicht Artikel an Model weiter.
Translator	Übersetzt die erhaltenen Texte in die gewünschte Sprache.
Summarizer	Fasst erhaltene Texte zusammen.
KeywordGenerator	erstellt auf Basis des erhaltenen Textes eine Liste an passenden Schlüsselwörtern.

Tabelle 4.3.: Die im Level 3 enthaltenen Blackboxes

4.2.3. Laufzeitsicht

Die Laufzeitsicht beschreibt die wechselseitigen Beziehungen der erdachten Bausteine und Module. Es werden die wichtigsten Prozesse und Merkmale der Anwendung beschrieben und visuell dargestellt Starke und Hruschka (2023). Die Anwendung besteht grundlegend aus zwei Kernprozessen, die durch einen Nutzer gestartet werden. Die Erstellung einer Suchanfrage und die Detailansicht eines bereits gesuchten Artikels. Im Folgenden werden diese beiden Prozesse kurz erläutert und in der Laufzeitsicht dargestellt.

Laufzeitsicht einer Suchanfrage

Das in Abbildung 16 gezeigte Diagramm beschreibt die Laufzeitsicht einer beliebigen Suchanfrage durch einen Nutzer. Es ist jedoch zu erwähnen, dass teilweise Prozesse abstrahiert dargestellt werden, um das Verständnis und die Übersicht über den gesamten Prozess weitestgehend zu vereinfachen. Hierbei steht vor allem der Informationsfluss im Vordergrund, wobei einige konkrete Details fehlen. So wird darauf verzichtet, jeden einzelnen Scraper gesondert darzustellen und anstatt dessen den generischen Scraper zu verwenden. Aus diesem Grund kommuniziert der ScraperManager direkt mit dem Scraper-Modul und nicht, wie in Abbildung 14 zu sehen, mit den konkreten Scrapern der Datenquellen. Außerdem ist die Kommunikation zwischen dem Scraper und dem Model nur vereinfacht, jedoch für das grundlegende Verständnis ausreichend, dargestellt. Des Weiteren wurde aus Gründen der Übersicht auf die Darstellung des

Summarizer-, Translator- und KeywordGenerator-Moduls verzichtet. Diese sind ebenfalls in den generischen Scraper integriert. Die wohl für die Laufzeit wichtigste, entfallene Eigenschaft ist die Nebenläufigkeit der Anwendung. Wie bereits in 4.2.2 erwähnt, erben die drei Scraper von der Thread-Klasse und werden jeweils in separaten Threads gestartet. Dies ermöglicht eine effiziente, parallele Abarbeitung der Artikel. Eine visuelle Darstellung der Nebenläufigkeit würde die Übersicht und Lesbarkeit jedoch im erheblichen Maße einschränken.

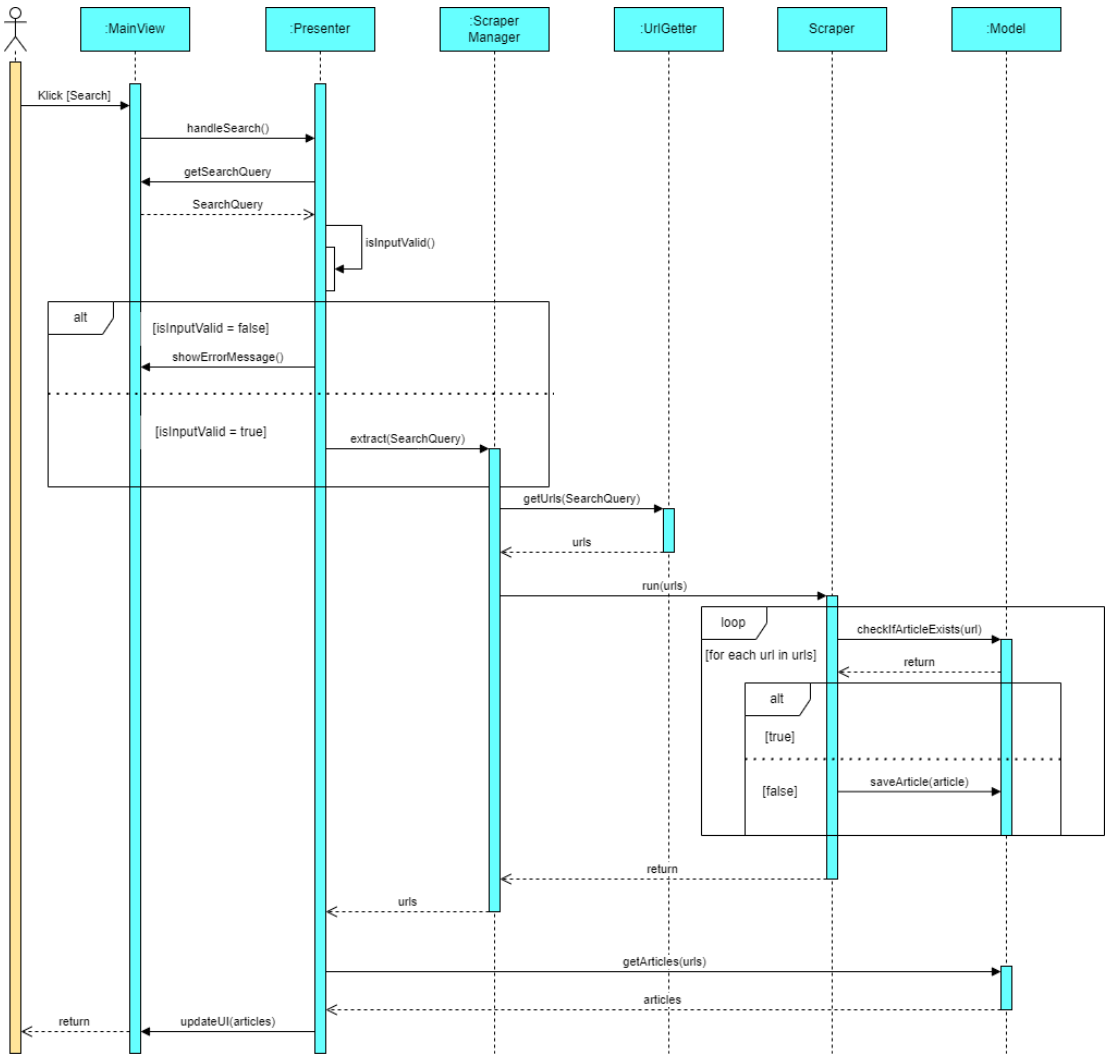

Abbildung 15.: Laufzeitsicht des Scraping Prozesses

Laufzeitsicht einer Detailanfrage

Die Laufzeitsicht einer Detailanfrage ist im Vergleich zu der einer Suchanfrage weitaus simpler. Dies ist damit zu erklären, dass alle benötigten Daten bereits durch die vorhergegangene Ausführung der Suchanfrage extrahiert und gespeichert wurden. Folglich kommt es bei der Ausführung einer Detailanfrage lediglich zu einem lesenden Zugriff auf die Daten. Dieser erfolgt über die ID eines Artikels, welche in Abbildung 7 als Primärschlüssel ausgewiesen ist. Außerdem ist in dieser Laufzeitsicht zu erkennen, dass der Presenter ebenfalls den aktuell ausgewählten Artikel im Feld currentArticle speichert. Dies ermöglicht den in Kapitel 4.1.3 festgelegten Wechsel zwischen der Zusammenfassung und dem eigentlichen Text eines Artikels, ohne auf das Model zuzugreifen. Eine solche Lösung ist eher ungewöhnlich, da in der Regel nur das Modell die Daten hält und der Presenter diese lediglich abfragt bzw. ändert.

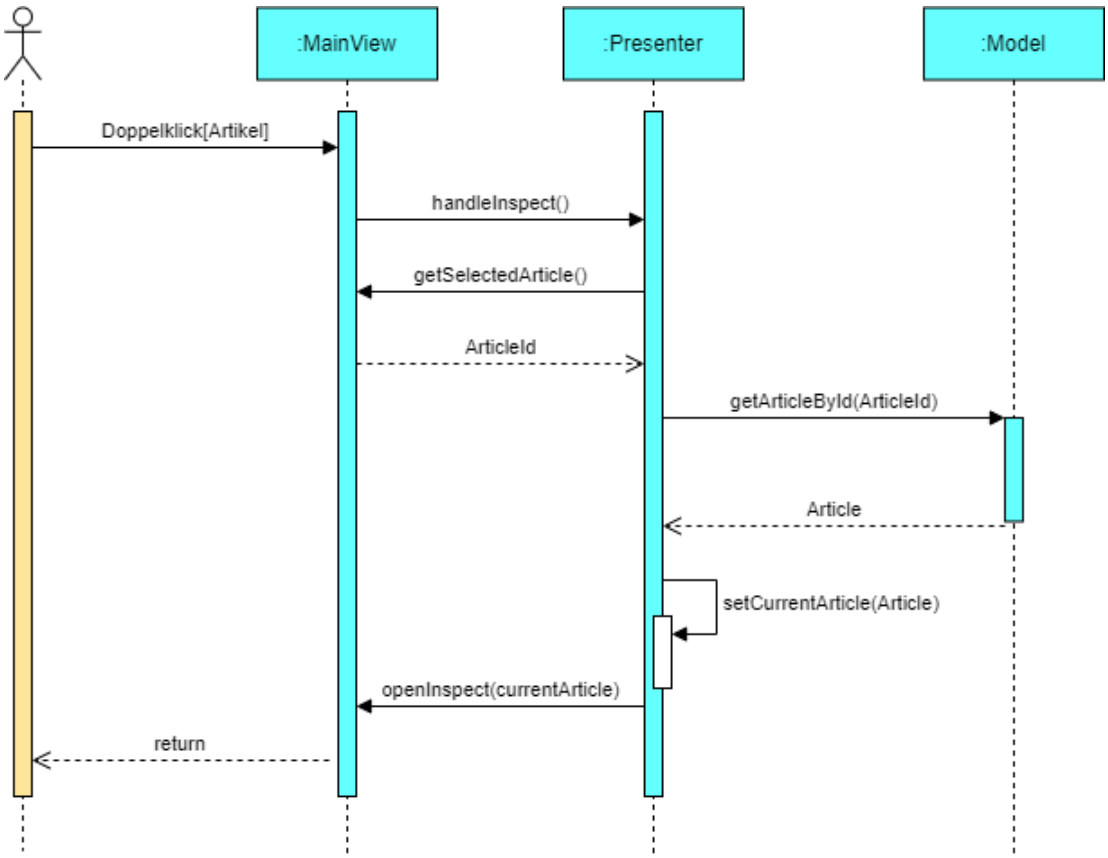

Abbildung 16.: Laufzeitsicht einer Detailanfrage

5. Realisierung

Nach der Konzeption der Entwurfsentscheidungen, wird nun die Realisierung dieser genauer erläutert. Hierzu werden zunächst die verwendeten Werkzeuge vorgestellt und dessen konkreter Anwendungsbereich erklärt. Anschließend wird die Implementierung der realisierten Web Scraper genauer beleuchtet und anhand von Code-Beispielen verdeutlicht.

5.1. Auswahl der Technologie

Die Auswahl der Technologie hat großen Einfluss auf das Verhalten und die Qualität der späteren Anwendung. Es herrscht eine enorme Variation zwischen Technologien, die zur Auswahl stehen. Jede einzelne verwendete Technologie bringt Vor- und Nachteile mit sich, welche gegeneinander abgewogen werden müssen. Im Folgenden werden die gewählten Technologien, zusammen mit den erdachten Anwendungsbereichen, kurz erläutert.

5.1.1. Python

Die grundlegendste Entscheidung für die Realisierung des entwickelten Entwurfs ist die Auswahl einer Programmiersprache. Für die Umsetzung der in dieser Arbeit konzipierten Anwendung wurde Python gewählt. Python ist zum Zeitpunkt dieser Arbeit, laut des PYPL-Indexes (Popularity of Programming Language), die beliebteste Programmiersprache der Welt Carbonnelle (2023). Dies bedeutet, dass die Anwendung auch auf längere Sicht zukunftsfähig, wartbar und erweiterbar sein wird. Durch die Popularität bringt Python eine Vielzahl an bereits entwickelten Software-Bibliotheken mit, die vor allem beim Thema Web Scraping eine übergeordnete Rolle spielen und den benötigten Zeit- und Arbeitsaufwand um ein Vielfaches verringern. Außerdem ermöglicht Python eine objektorientierte Programmierung, welches im Entwurf konzipierte Konzepte, wie Vererbung, ermöglicht.

5.1.2. MySQL

Wie bereits im Kapitel 4.1.2 erdacht, wird für die Umsetzung der Anforderungen eine relationale Datenbank benötigt. MySQL ist ein verbreitetes Datenbankverwaltungssystem, welches als

Open-Source-Software kostenlos angeboten wird. MySQL bietet alle grundlegenden Funktionen einer relationalen Datenbank sowie die ACID-Eigenschaften. Die Kommunikation zwischen der Datenbank und der Python-Anwendung wird durch die MySQL-Connector-Bibliothek, welche in dem Model verwendet wird, ermöglicht. Der MySQL-Connector unterstützt parametrisierte Abfragen der Datenbank, welche eine Attacke durch SQL-Injection verhindern sollen. Außerdem ist es möglich, über die MySQL-Workbench eine grafische Benutzeroberfläche des Datenbanksystems aufzurufen MySQL (2023).

5.1.3. TkInter

Für die Erstellung der grafischen Benutzeroberfläche wird die TkInter-Bibliothek verwendet. TkInter erweitert das GUI-Toolkit Tcl/Tk um einige objektorientierte Eigenschaften und bildet den de-facto Standard für grafische Benutzeroberflächen in Python Tkinter (2023). Der objektorientierte Ansatz ermöglicht eine effiziente und intuitive Benutzung. Die Umsetzung der in 4.1.3 erstellten Vorgaben ist in A.1 zu sehen.

5.1.4. Requests

Die Requests-Bibliothek wird verwendet, um HTTP-Anfragen an die verschiedenen Webserver der Quellen zu stellen. Durch ein hohes Abstraktionsniveau ist die Verwendung simpel und ermöglicht effiziente Arbeit. Erhaltene HTTP-Antworten können durch angebotene Methoden umgewandelt und gefiltert werden Reitz (2023). Im Kontext dieser Arbeit wird mit der Requests-Bibliothek das gesamte HTML-Dokument der jeweiligen Webseite angefragt. Hierbei verwenden der UrlGetter und der Scraper diese, um die Suchwebseiten bzw. die Artikelwebseiten zu erhalten. Neben der Url der Webseite werden ebenfalls User Agents als Parameter übergeben, welche dem Webserver Informationen über das verwendete Gerät übermitteln. Außerdem werden eventuell auf dem HTML-Dokument enthaltene Scripts nicht ausgeführt, weswegen nur statische Elemente auf diesem angezeigt werden. Mit Hinblick auf die in 3.1 festgehaltenen Gegebenheiten wird die Request-Bibliothek bei der Abfrage von statisch geladenen Webseiten verwendet.

5.1.5. Beautiful Soup

Die BeautifulSoup-Bibliothek verarbeitet HTML- oder XML-Dokumente und stellt Methoden zur Analyse dieser. Dies ermöglicht es, die von der Requests-Bibliothek erhaltenen HTML-Dokumente nach spezifischen Attributen, wie z.B. Datum oder Titel, zu filtern und die benötigten Daten gezielt zu extrahieren Richardson (2023). Durch diese zentrale Rolle innerhalb des

Programms ist es von äußerster Bedeutung, die BeautifulSoup-Bibliothek effizient einzusetzen. Um dies zu gewährleisten, ist eine genaue Kenntnis der HTML-Dokumente notwendig. Die Filterung der Daten findet hierbei über verschiedenste Merkmale wie den HTML-Tags oder der CSS-Class statt. Die konkrete Ausprägung dieser Merkmale variiert zwischen den Datenquellen.

5.1.6. Selenium

Das Selenium-Framework ist sehr umfangreich und bietet eine Vielzahl an Funktionen. Der zentrale Aspekt ist die automatisierte Nutzung von Webbrowsern, um dadurch die Eingaben eines menschlichen Nutzers zu simulieren. Durch die Verwendung eines vollständigen Browsers bietet Selenium, im Gegensatz zu Requests, die Möglichkeit, auch dynamisch geladene Elemente auszuführen Selenium (2023). Aus diesem Grund wird Selenium im Kontext dieser Arbeit für das Bereitstellen von dynamisch geladenen Inhalten verwendet, welches ausschließlich bei der Suchwebseite von Politico der Fall ist. Der Aufwand an Ressourcen ist durch die Simulation eines vollständigen Browsers jedoch um einiges höher als der von Requests.

5.1.7. Yake

Die Yake-Bibliothek (Yet Another Keyword Extractor) wird verwendet, um Schlüsselwörter aus Texten zu generieren. Hierbei ist es nicht notwendig, das hinterlegte KI-Model mit eigenen Texten zu trainieren. Außerdem ist es möglich, Schlüsselwörter aus Texten in vielen verschiedenen Sprachen und variabler Länge zu generieren. Campos u. a. (2020). Dies ist hinsichtlich der Variation der Artikel von besonderer Bedeutung und schlussendlich auch der Grund, warum die Wahl auf Yake gefallen ist. Yake wird in der KeywordGenerator-Klasse verwendet, um Schlüsselwörter auf Basis der Artikel zu generieren. Die Generierung kann auf Basis verschiedener Parameter geschehen. Diese beeinflussen Eigenschaften wie die Sprache oder die Anzahl an Schlüsselwörtern. Für eine Übersicht der übergebenen Parameter, siehe A.2.1.

5.1.8. Deep-translator

Für die Übersetzung der Artikel wurde die Deep-Translator-Bibliothek verwendet. Diese bietet Zugriff auf eine Vielzahl an Übersetzern. Angeboten werden unter anderen: Google Translate, PONS Translator, ChatGPT Translator oder DeeplTranslator. Der Zugriff auf diese erfolgt über eine, durch Deep-Translator abstrahierte, API, für die im Falle von ChatGPT-Translator und DeeplTranslator ein API-Key benötigt wird. Aus diesem Grund ist die Wahl auf Google Translate gefallen. Dieser Übersetzer erkennt die übergebene Sprache, weshalb lediglich die

Zielsprache angegeben werden muss. Das führt dazu, dass die Wiederverwendung eines Übersetzer-Objektes ermöglicht wird. Es ist jedoch anzumerken, dass die maximale Textlänge lediglich 5000 Zeichen beträgt Baccouri (2023). Da manche Artikel weitaus mehr vorweisen, musste der Text in Fragmenten übersetzt werden. Für die übergebenen Parameter und den Code zur Fragmentierung der Artikel siehe A.2.2.

5.1.9. OpenAI

Die Zusammenfassung der Texte ist im Kontext dieser Arbeit von besonderer Bedeutung. Aus diesem Grund erfolgt diese über die API von OpenAI, welche über die OpenAI-Bibliothek in Python aufgerufen werden kann. Für die Verwendung ist ein API-Key notwendig, der nach Erstellung eines Benutzerkontos auf der OpenAI-Webseite abrufbar ist. Der API-Aufruf findet parametrisiert statt, welches unter anderem Einfluss auf das verwendete Model oder die Länge der Zusammenfassung hat Openai (2023). Die genaue Verwendung sowie die Wahl der Parameter sind in A.2.3 zu finden. Außerdem ist zu erwähnen, dass, je nach verwendetem Model und der Länge der Eingabe, Kosten pro API-Aufruf anfallen. Um die Kosten pro Artikel zu minimieren, ist es von besonderer Bedeutung, Zusammenfassungen in abweichender Sprache lediglich zu übersetzen, anstatt diese erneut zusammenzufassen. Um weitere Kosten zu vermeiden, werden Texte bereinigt, wobei für den Nutzer unnötige Leerzeichen oder Zeilenumbrüche entfernt werden.

5.2. Implementierung der Web Scraper

Die Implementierung der drei WebScraper stellt das Herzstück der Anwendung dar. Diese sind für die Extraktion, Verarbeitung und Speicherung der Artikel verantwortlich. Ein Großteil der Module zielt darauf ab, die Scraper entweder mit Daten zu versorgen oder von diesen zur Verarbeitung der Artikel verwendet zu werden. Das wichtigste Element des Scrapers ist die Methode *fetchAllData*, welche von dem AbstractScraper an den erweiternden Scraper vererbt wird und in Abbildung 17 zu sehen ist. *FetchAllData* erhält als Parameter ein Objekt der Klasse *ArtikelReduced*, welches lediglich die Url, die Agentur und das Datum eines einzelnen Artikels als Felder enthält. Durch diese drei Felder ist es dem Scraper möglich, den Rest der Daten, wie z.B. den Titel, zu extrahieren. Zu Beginn der Methode *fetchAllData* wird durch das bereits im Konstruktor erstelle Model-Objekt geprüft, ob der Artikel bereits in der Datenbanktabelle *Artikel* enthalten ist. Hierzu wird die Url des Artikels als Suchparameter verwendet, da diese einzigartig und indexiert ist, was eine effiziente Suche ermöglicht. Falls der Artikel noch nicht

existiert, wird die Hilfsmethode *createOriginalArticle* aufgerufen, welche in Abbildung 18 zu sehen ist.

```
"""
Prueft, ob Artikel bereits in der korrekten Sprache existiert.
"""
def fetchAllData(self, article: ArtikelReduced) -> None:
    if not (self.model.urlAlreadyInDB(article.url)):
        self.createOriginalArticle(article)

    if not (self.model.existsArticleInLg(article.url, self.
       destLanguage.value)):
        self.translateArticle(article)
```

Abbildung 17.: Quellcode der Methode *fetchAllData*

CreateOriginalArticle erhält über den Aufruf durch *fetchAllData* ebenfalls ein Objekt der Klasse *ArtikelReduced*. Zu Beginn wird ein Tupel erstellt und gespeichert, welches eine Zeile der Tabelle *Artikel* repräsentiert. Anschließend wird die gesamte Url des Artikels zusammengefügt, da die Url innerhalb des *ArtikelReduced*-Objektes nur den Pfad und nicht die Domain der Webseite enthält. Dies ist damit zu erklären, dass die zusätzliche Speicherung der jeweiligen Domain die Suche per Url ineffizienter machen würde. Die zusammengesetzte Url wird nun verwendet, um durch die Requests- und BeautifulSoup-Bibliothek das HTML-Dokument zu erhalten und anschließend in ein BeautifulSoup-Objekt zu parsen. Das Objekt wird nun an die Methoden *extractHeadline* und *extractText* als Parameter übergeben, welche den Titel und Text des Artikels extrahieren und säubern. Diese beiden Methoden werden, aufgrund der unterschiedlichen Quellen, durch die drei Scraper erweitert und sind in A.3 zu sehen. Anschließend wird der extrahierte Text verwendet, um Schlüsselwörter und eine Zusammenfassung zu erstellen. Hierzu werden die Objekte der Helferklassen *KeywordGenerator* und *Summarizer* verwendet, welche bereits im Konstruktor des Scrapers erstellt wurden. Der Code der beiden Klassen ist unter A.2.1 und A.2.3 zu finden. Nachdem alle benötigten Daten gesammelt oder erstellt wurden, werden diese als Tupel in der Tabelle *Artikel_Data* gespeichert, welches erneut über das Model-Objekt geschieht.

```
"""
Extrahiert Daten des angegebenen Artikels.
Speichert diese in originaler Sprache in Datenbank.
"""
def createOriginalArticle(self, article: ArtikelReduced) -> None:
    artikelTupel = (self.AGENTUR.value, article.url, article.datum)
    self.model.insertArtikel(artikelTupel)
    fullUrl = self.URL + article.url
    rawPage = requests.get(fullUrl, headers=self.HEADERS)
    articlePage = BeautifulSoup(rawPage.content, 'html.parser')

    originalHeadline = self.extractHeadline(articlePage)
    originalText = self.extractText(articlePage)
    keywords = self.keywordGenerator.getKeywords(originalText, self.
        LANGUAGE)
    summary = self.summarizer.getSummary(originalText)
    artikelDataTupel = (self.model.getArtikelIdByUrl(article.url),
                        self.LANGUAGE.value,
                        originalHeadline,
                        originalText,
                        str(keywords),
                        summary)
    self.model.insertArtikelData(artikelDataTupel)
```

Abbildung 18.: Quellcode der Methode *createOriginalArticle*

Nach dem eventuellen Aufruf der *CreateOriginalArticle*-Methode innerhalb von *fetchAllData*, wird zunächst geprüft, ob der Artikel bereits in der vom Benutzer geforderten Sprache vorhanden ist. Um dies zu überprüfen, wird neben der Url ebenfalls die Sprache als Suchkriterium an die Methode *existsArticleInLg* des Models übergeben. Die Kombination aus Url und Sprache stellt ein Schlüsselattribut in der *Artikel_Data* Tabelle dar und wird benötigt, da Artikel mit derselben Url in verschiedenen Sprachen existieren können. Falls die If-Bedingung nicht zutreffen sollte, wird die Methode *translateArticle* aufgerufen, welche in Abbildung 19 zu sehen ist. *TranslateArticle* dient der Übersetzung der relevanten Artikeldaten. Zu Beginn wird der, aufgrund der sequentiellen Abfolge, existierende, originale Artikel abgefragt. Danach werden Titel, Text, Schlüsselwörter und Summary übersetzt und zusammen mit der benötigten Sprache sowie der ID des originalen Artikels gespeichert.

```
"""
Übersetzt einen bereits vorhandenen Artikel in angegebene Sprache
"""
def translateArticle(self, article: ArtikelReduced) -> None:
    artikel = self.model.getArtikel(article.url, self.LANGUAGE.value
        )
    dataTupel = (artikel.baseArtikel,
                 self.destLanguage.value,
                 self.translator.translate(artikel.titel),
                 self.translator.translate(artikel.text),
                 self.translator.translate(artikel.keywords),
                 self.translator.translate(artikel.summary))
    self.model.insertArtikelData(dataTupel)
```

Abbildung 19.: Quellcode der Methode *translateArticle*

6. Evaluierung

Nach der Realisierung der entwickelten Konzepte, wird in diesem Kapitel die Anwendung evaluiert und anschließend bewertet. Dies geschieht auf Basis von unterschiedlichen Softwaretests, die Aussagen über die Qualität ermöglichen. Die Bewertung ist grundlegend ein Soll-Ist-Vergleich der konzipierten Anforderungen mit den tatsächlich realisierten Merkmalen und Funktionen.

6.1. Testing

Um die Evaluierung der Software grundlegend zu ermöglichen, wurden Softwaretests entwickelt. Diese ermöglichen Aussagen über die Funktion und Qualitätsmerkmale der Applikation und ermöglichen einen Soll-Ist-Vergleich der gestellten Anforderungen. Die Tests wurden parallel zur Realisierung der Anwendung konzipiert und entwickelt, welches eine kontinuierliche Qualitätssicherung ermöglichte. Die verwendeten Tests können traditionell in Unit-Tests, Integrationstests, und Systemstests unterteilt werden und werden im Folgenden kurz erläutert.

Bei einem Unit-Test werden die einzelnen Module des Systems separat voneinander getestet. Hierbei geht es konkret um einzelne Methoden innerhalb der Module, die entweder als Black-Box oder White-Box getestet werden können. In der Regel wird eine Testabdeckung von 100% als erstrebenswert angesehen Hubertz (2016). Im Kontext dieser Arbeit wurden die meisten der konzipierten Klassen durch Unit-Tests getestet, jedoch nicht alle. Es wurde mit Hilfe des AAA-Patterns getestet, welches zu Beginn den Test vorbereitet (*Arrange*), dann die zu testenden Elemente ausführt (*Act*) und zuletzt die erhaltenen Ergebnisse mit den Erwarteten abgleicht (*Assert*) Jorgensen (2013). Umgesetzt wurden die Tests mit Hilfe des in Python integrierten Frameworks *unittest*, welches eine automatische Ausführung aller Tests ermöglicht. Zu den Modulen, die gar nicht bzw. nur geringfügig getestet worden sind, zählen der Summarizer, KeywordGenerator, Translator, Presenter, sowie die drei Scraper. Der Summarizer, KeywordGenerator und Translator wurden nicht getestet, da diese lediglich eine Abstraktionsstufe einer bereits getesteten Bibliothek bzw. API bilden, welche bereits ausreichend getestet wurde. Außerdem ist das Verhalten dieser drei Module nur schwer, mit traditionellen Verfahren, testbar.

So ist eine Zusammenfassung oder eine List von Schlüsselwörter stark vom jeweiligen Text abhängig und, unter Umständen, sogar jedes Mal verschieden. Dies verhält sich bei übersetzten Texten ähnlich. Die Presenter-Klasse ist ebenfalls schwer isoliert testbar, da diese direkt mit dem View verbunden ist und somit mit der grafischen Benutzeroberfläche kommuniziert. Dies ist mit klassischen Unit-Tests nur schwer umsetzbar und würde ein Mock-Objekt der View benötigen, welches jedoch im Rahmen dieser Arbeit keine Anwendung gefunden hat. Die drei Scraper-Klassen wurden nicht mit Unit-Tests versehen, da diese direkt mit dem Model kommunizieren und deswegen nicht isoliert getestet werden können. Trotzdem werden die oben genannten Module und Klassen im Rahmen von Integrations- und Systemtests auf die gewünschte Funktion geprüft.

Nachdem für die meisten Klassen bereits Unit-Tests implementiert wurden, wird nun die Kommunikation dieser untereinander getestet Hubertz (2016). Diese Art von Tests werden auch Integrationstests genannt und sind ebenfalls nach dem AAA-Pattern strukturiert. Der Fokus der Integrationstest lag auf der korrekten Funktionalität der drei Scraper-Klassen, die, wie bereits erwähnt, nicht isoliert getestet werden konnten. Da die Scraper direkt mit der Model-Klasse kommunizieren, wurde diese durch umfangreiche Unit-Tests getestet, um sicherzustellen, dass mögliche Fehler alleine durch die Scraper ausgelöst werden.

Nach der Implementierung der Integrationstests, wurde ein manueller Systemtest durchgeführt, welcher das Verhalten der grafischen Benutzeroberfläche, sowie die Verwendbarkeit, aus Sicht des Anwenders, testen sollte Hubertz (2016). Zu Beginn wurden einige Negativtests durchgeführt, welche die Fehlerbehandlung der Anwendung prüfen sollte. Hierzu wurden klar falsche bzw. fehlende Parameter übergeben, wie z.B. ein fehlender Suchbegriff oder ein vonDatum, welches nach dem bisDatum liegt. Die Anwendung hat jeden möglichen Fehlerfall festgestellt und visuell an den Anwender weitergegeben. Anschließend wurden mehrere Positivtests durchgeführt, welche ein normales Suchverhalten eines Anwenders repräsentieren sollen. Die Parameter eines dieser Tests ist in Abbildung 6.1 zu sehen. Da alle möglichen Quellen verwendet wurden, ist die Aussagekraft der Ergebnisse, die in Abbildung 6.2 zu sehen sind, als hoch einzuschätzen. Hierbei ist zu beachten, dass ein Artikel als erfolgreich extrahiert angesehen wird, wenn Titel, Datum, Agentur, Url und Text vollständig und fehlerfrei vorhanden sind. Die Qualität der Zusammenfassungen und Schlüsselwörter ist hierbei subjektiv bewertet worden und hat dementsprechend keinen Einfluss auf die erhaltenen Ergebnisse. Bei der Übersetzung ist lediglich die korrekte Sprache für das Ergebnis von belangen. Außerdem

wurde für jeden Durchlauf die Zeit gemessen, die benötigt wurde um alle gewünschten Artikel anzuzeigen.

Parameter	Wert
Suchbegriff	Armenien
Quellen	Tagesschau, EuroNews, Politico
Sprache	Deutsch
Datum von	12.09.2023
Datum bis	19.09.2023

Tabelle 6.1.: Parameter eines Testfalls

	Erwartet	Ergebnis
Tagesschau	4	4
EuroNews	3	3
Politico	6	6
Gesamt	**13**	**13**

Tabelle 6.2.: Ergebnisse eines Testfalls

6.2. Ergebnisse

Die Ergebnisse der durchgeführten Systemtests dienen der grundlegenden Bewertung der Anwendung und haben bereits die grundlegende Funktionalität dieser bewiesen. Die Bewertung findet hierbei auf Basis der konzipierten funktionalen Anforderungen und Qualitätsmerkmalen statt, die unter 3.2 zu finden sind. Die gestellten funktionalen Anforderungen wurden grundlegend zufriedenstellend erfüllt. Lediglich *F2.5: Verarbeitungsstatus anzeigen*, wurde nur im bedingten Maße umgesetzt. Zwar wird dem Nutzer über die Statusanzeige angezeigt, ob das System aktuell mit der Verarbeitung beschäftigt ist, jedoch erhält der Nutzer darüber hinaus keine genaueren Informationen zum konkreten Status dieser. Das ist mit der Nebenläufigkeit der Anwendung zu erklären. Da die Scraper in verschiedenen Threads ausgeführt werden, ist der Verarbeitungsstatus möglicherweise bei jedem Scraper unterschiedlich, weshalb es keinen universellen Status der Anwendung gibt. Die konzipierten Qualitätsmerkmale, sind bei der Bewertung der Anwendung ebenfalls von großer Bedeutung. Im Rahmen des Systemtests wurde deutlich, dass die Anwendung die Mermale: *Q1: Bedienbarkeit*, *Q2: Effiziente Leistung* im vollem Umfang vorweist. Einzig das Merkmal *Q3: Wartbarkeit* ist nur grundlegend implementiert, welches mit der Struktur der Anwendung zu erklären ist. Dadurch, dass der URLGetter die Urls

für die Scraper beschafft, wurde wissen über die Datenquelle in diesen verbaut. Dies erschwert die Fehlersuche, sowie die Wart- und Erweiterbarkeit der Anwendung.

7. Schluss

In diesem Kapitel wird zunächst die im Kontext dieser Bachelorarbeit erbrachte Leistung zusammengefasst und bewertet. Hierbei wird sowohl auf positive als auch auf negative Aspekte eingegangen. Anschließend wird ein Ausblick erstellt, in welchem zukünftig denkbare Erweiterungen und Funktionen konzipiert werden.

7.1. Fazit

Im Rahmen dieser Arbeit wurde eine Anwendung entwickelt, die Nachrichtenartikel von mehreren Quellen durch Web Scraper extrahiert und diese durch, teilweise durch KI-gestützte, Tools für den Nutzer aufbereitet. Die Umsetzung ist hierbei in die Schritte Grundlagenrecherche, Analyse, Konzeption, Realisierung und Evaluation unterteilt worden, was eine strukturierte Vorgehensweise ermöglicht hat. Die Anwendung erfüllt grundlegend alle gestellten funktionalen Anforderungen und weist die drei gewählten Qualitätsmerkmale auf. Jedoch gab es bei der Entwicklung einige Probleme, die nun kurz erläutert werden. Zu Beginn der Analyse von potentiellen Datenquellen wurde schnell klar, dass nur eine geringe Menge an Webseiten für die Anwendung in Frage kommen würde. Der Grund hierfür waren die restriktiven Nutzungsbedingungen und robots.txt-Dateien der verschiedenen Nachrichtenwebseiten. Da diese nicht von Beginn an klar war, stellte sich die Recherche als zeitintensiever als geplant heraus. Außerdem musste die Zahl der Quellen auf drei reduziert werden. Ein weiteres Problem war die grundlegende Architektur der Scraper, insbesondere die Kommunikation mit anderen Klassen bzw. Modulen. Zu Beginn der Entwicklung wurde nirgendwo festgehalten, wie die Schnittstellen konkret miteinander kommunizieren. Aus diesem Grund wurde teilweise aus Notwendigkeit nach dem Bottom-up-Prinzip entwickelt, welches Folgen für weitere, noch nicht entwickelte, Module und Schnittstellen hatte. Besonders klar wird dies durch die indirekte Kommunikation von UrlGetter-Modul und den einzelnen Scrapern, welche stark von der Ausgabe des UrlGetters abhängig sind. Dies führt zur Verletztung des Dependency-Inversion-Prinzips und erschwert die zukünftige Wart- und Erweiterbarkeit der Anwendung. Neben den getroffenen, hinderlichen Architekturentscheidungen ist ebenfalls die Wahl der Benutzeroberfläche kritisch zu betrachten. Der gewählte Ansatz eines Thick Client erfüllt die erdachten

Funktionen im vollem Maße, ist jedoch nicht wirklich zeitgemäß. Als alternative Option wäre ein Webservice wohlmöglich die bessere und modernere Wahl gewesen. Ein Webservice würde eine geringere Einstiegshürde für potentielle Nutzer bieten, wodurch es zu einer vermehrten Nutzung kommen würde. Ein Nachteil dieses Ansatzes ist, dass speziell für die Anwendung ein Webserver gehostet werden müsste, welches zu einem höheren Aufwand an Ressourcen führen würde.

Neben den eben behandelten Problemen sind ebenfalls positive Aspekte der Konzeption und Umsetzung aufzuzeigen. Die grundlegende Architektur erfüllt im vollem Umfang ihre Funktion und trennt die Anwendung, je nach Zuständigkeiten, in einzelne Module. Dies hat die Implementierung der parallelen Verarbeitung von Artikeln stark vereinfacht und so die Komplexität möglichst niedrig gehalten. Darauf aufbauend wurde die Vererbungsstruktur der Scraper-Klassen erfolgreich umgestetzt, was die Erweiterbarkeit um neue Datenquellen vereinfacht. Des Weiteren ist im Besonderen die Wahl der Programmiersprache hervorzuheben. Die Entwicklung mit Python hat unnötige Komplexität verhindert und den Zugriff auf eine Vielzahl an Bibliotheken und Frameworks ermöglicht. Die Einbindung der verwendeten Bibliotheken und Frameworks hat die Komplexität der Anwendung erheblich verringert, sowie die Umsetzung einzelner Funktionen, wie der Übersetzung, Zusammenfassung oder Schlüsselwortgenerierung, überhaupt erst ermöglicht. Besonders die Qualität dieser Funktionen bietet der Anwendung einen hohen Mehrwert.

7.2. Ausblick

Die im Kontext dieser Arbeit entworfene Anwendung bietet bereits einige Funktionalitäten, die zusammen ein kohärentes Produkt bilden. Trotzdem konnten aus Zeit- und Ressourcenmangel einige denkbare Funktionen und Eigenschaften nicht implementiert werden. Diese werden im Folgenden kurz vorgestellt und beziehen sich zum Teil auf die im Fazit erwähnten Probleme.

Eine zentrale Erweiterung wäre das dynamische Hinzufügen von Quellen durch den Benutzer. Durch das Angeben einer Vielzahl an Parametern, wie z.B. die URL der Suchwebseite, könnte so jede erdenktliche Nachrichtenwebseite in die Anwendung integriert werden. Technisch wäre diese Erweiterung im Rahmen des Möglichen, rechtlich hingegen eher als kompliziert anzusehen. Dadurch, dass Nutzer unter Umständen die rechtlichen Grundlagen des Web Scraping nicht vollständig verstehen, könnte es zu Verletzungen von Nutzungsbedingungen und/oder Gesetzen kommen. Dies könnte rechtliche Konsequenzen in Form einer Klage nach sich ziehen und ist somit nur mit Beistand eines rechtlichen Beraters umsetzbar.

Neben der Erweiterung um zusätzliche Quellen wäre es ebenfalls möglich, die extrahierten Nachrichtenartikel für weitere Analysen zu benutzen. So wäre eine Sentimentanalyse der Artikel denkbar, die die vertretene Haltung als positiv oder negativ beurteilen kann Siegel und Alexa (2020). Dadurch wird die Transparenz für den Nutzer erhöht und dieser könnte sich differenzierter zu gewünschten Themen informieren. Ein weitere Einsatzmöglichkeit für die extrahierten Texte wäre das Erstellen eines eigenen KI-Modells zur verbesserten Zusammenfassung. Als Basis für dieses Modell könnten bereits vortrainierte Modelle dienen, die lediglich auf die konkreten Eigenschaften von Nachrichtenartikeln spezialisiert werden müssten. Wahrscheinlich wäre hierfür jedoch eine große Menge an bereits aufgearbeiteten Daten notwendig, welches die Implementation erschweren könnte.

A. Anhang

A.1. Umsetzung der Wireframes

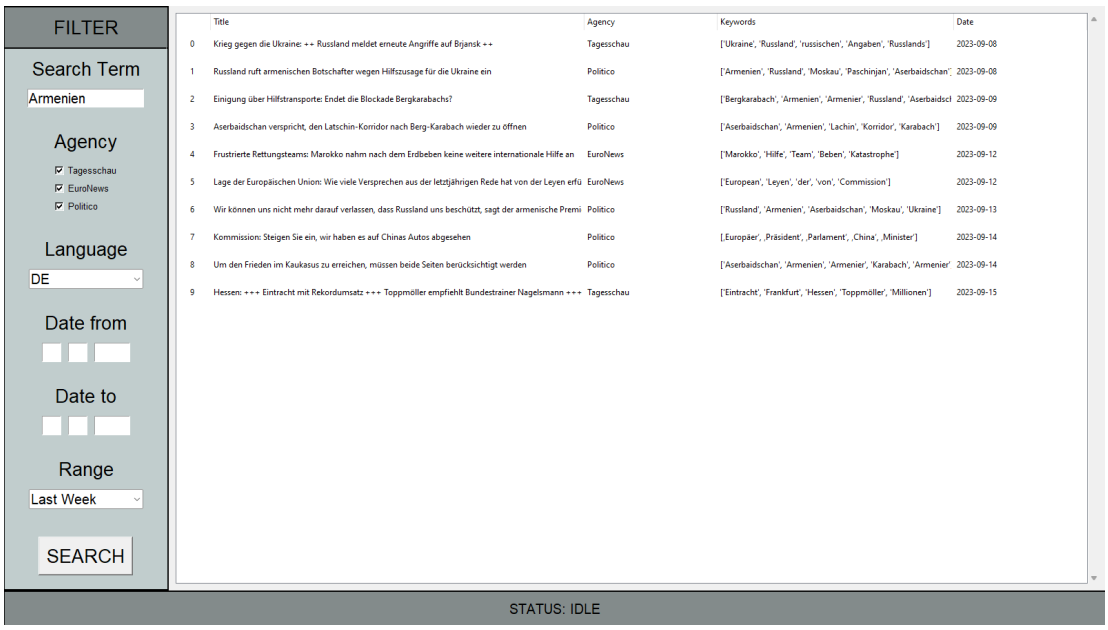

Abbildung 20.: Umsetzung der Hauptansicht

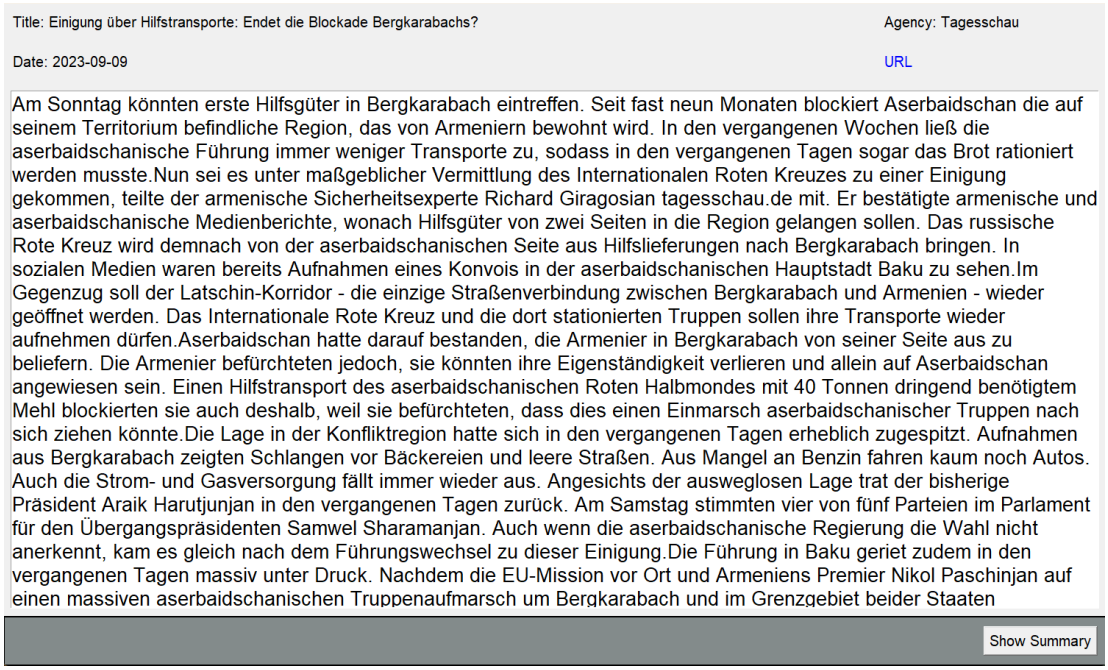

Abbildung 21.: Umsetzung der Detailansicht

A.2. Verwendete Werkzeuge: Quellcode

A.2.1. Verwendung: Yake-Bibliothek

```
import yake
class KeywordGenerator():
    def getKeywords(self, text, language) -> [str]:
        keywordList = []
        max_ngram_size = 1
        deduplication_threshold = 0.9
        numOfKeywords = 5
        keyword_extractor = yake.KeywordExtractor(
            lan=language.value,
            n=max_ngram_size,
            dedupLim=deduplication_threshold,
            top=numOfKeywords,
            features=None)
        keywords = keyword_extractor.extract_keywords(text)
        for kw in keywords:
            keywordList.append(kw[0])
        return keywordList
```

Abbildung 22.: Quellcode der *KeywordGenerator* Klasse

A.2.2. Verwendung: Deep-Translator-Bibliothek

```
from deep_translator import GoogleTranslator
class Translator():
    def __init__(self, dest_language: str):
        self.translator = GoogleTranslator(source='auto', target=
            dest_language.value.lower())
        self.MAX_LENGTH = 5000-1

    def translate(self,text: str) -> str:
        translated = ''
        if len(text) >= self.MAX_LENGTH:
            toTranslate = text
            while len(toTranslate) >= self.MAX_LENGTH:
                translated = translated + self.translator.translate(text=
                    toTranslate[:self.MAX_LENGTH])
                toTranslate = toTranslate[self.MAX_LENGTH:-1]
            return translated + self.translator.translate(text=toTranslate)
        translated = self.translator.translate(text=text)
        return translated
```

Abbildung 23.: Quellcode der *Translator* Klasse

A.2.3. Verwendung: OpenAI-API

```python
import tiktoken
import openai

openai.api_key = "API-KEY"
class Summarizer():
    def __init__(self):
        self.encoding = tiktoken.encoding_for_model("text-curie-001")
        self.LENGTH_OF_CALL = len(self.encoding.encode("tl;dr"))
        self.MAX_TOKENS = 2048 - self.LENGTH_OF_CALL - 200

    def getSummary(self, text: str):
        prompt = text
        encodedPrompt = self.encoding.encode(prompt)[:self.MAX_TOKENS]
        prompt = self.encoding.decode(encodedPrompt)
        response = openai.Completion.create(
            model="text-curie-001",
            prompt= prompt + "tl;dr",
            temperature=1,
            max_tokens=200,
            top_p=1.0,
            frequency_penalty=0.0,
            presence_penalty=1
        )
        response_list = list(response.choices)[0]
        return response_list.to_dict()["text"].strip()
```

Abbildung 24.: Quellcode der *Summarizer* Klasse

A.3. TagesschauScraper: Quellcode

```
class TagesschauScraper(AbstractScraper, threading.Thread):

    def __init__(self, articleList: [], destLanguage: str):
        threading.Thread.__init__(self)
        AbstractScraper.__init__(self,articleList,destLanguage)
        self.LANGUAGE: Sprache = Sprache.DEUTSCH
        self.AGENTUR: Agentur = Agentur.TAGESSCHAU
        self.URL: str = "https://www.tagesschau.de"
        self.cssTextClass = 'textabsatz m-ten m-offset-one l-eight l-offset-
            two columns twelve'
        self.cssToplineClass = 'seitenkopf__topline'
        self.cssHeadlineClass = 'seitenkopf__headline--text'

    def extractText(self, articlePage) -> str:
        articleText = articlePage.find_all('p', class_=self.cssTextClass)
        pList = []
        for p in articleText:
            if p is not None:
                pList.append(p.text.replace("\r\n\t", "").strip())
        result = ''.join(pList)
        return result

    def extractHeadline(self, articlePage) -> str:
        articleTopline = articlePage.find('span', class_=self.
            cssToplineClass)
        articleHeadline = articlePage.find('span', class_=self.
            cssHeadlineClass)
        resultHeadline = ""
        if articleTopline is not None:
            resultHeadline = articleTopline.text.strip() + ': '
        if articleHeadline is not None:
            resultHeadline = resultHeadline + articleHeadline.text.strip()
        else:
            header = articlePage.find('header', class_='articlehead boxed')
            if header is not None:
                resultHeadline = header.find('h1').text.strip()
        resultHeadline.replace("\r\n\t", "")
        return resultHeadline
```

Abbildung 25.: Quellcode der *TagesschauScraper* Klasse

Literaturverzeichnis

[Openai 2023] : *Introduction - OpenAI API*. 2023. – URL https://platform.openai.com/docs/introduction/overview

[MySQL 2023] : *MySQL 8.0 Reference Manual*. 2023. – URL https://dev.mysql.com/doc/refman/8.0/en/

[Selenium 2023] : *The Selenium Browser Automation Project: Documentation*. 2023. – URL https://www.selenium.dev/documentation/

[Tkinter 2023] : *TkInter - Pytohn Wiki*. 2023. – URL https://wiki.python.org/moin/TkInter

[Acar u. a. 2013] Acar, Gunes ; Juarez, Marc ; Nikiforakis, Nick ; Diaz, Claudia ; Gürses, Seda ; Piessens, Frank ; Preneel, Bart: FPDetective: dusting the web for fingerprinters. In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, S. 1129–1140

[Baccouri 2023] Baccouri, Nidhal: *Deep-Translator: Translation for humans*. 2023. – URL https://deep-translator.readthedocs.io/en/latest/README.html

[Bühler u. a. 2018] Bühler, Peter ; Schlaich, Patrick ; Sinner, Dominik: *Webtechnologien*. Springer, 2018

[Campos u. a. 2020] Campos, Ricardo ; Mangaravite, Vítor ; Pasquali, Arian ; Jorge, Alípio ; Nunes, Célia ; Jatowt, Adam: YAKE! Keyword extraction from single documents using multiple local features. In: *Information Sciences* 509 (2020), S. 257–289. – URL https://www.sciencedirect.com/science/article/pii/S0020025519308588. – ISSN 0020-0255

[Carbonnelle 2023] Carbonnelle, Pierre: *PYPL popularity of Programming Language index*. 2023. – URL https://pypl.github.io/PYPL.html

[Diouf u. a. 2019] Diouf, Rabiyatou ; Sarr, Edouard N. ; Sall, Ousmane ; Birregah, Babiga ; Bousso, Mamadou ; Mbaye, Sény N.: Web scraping: state-of-the-art and areas of application. In: *2019 IEEE International Conference on Big Data (Big Data)* IEEE (Veranst.), 2019, S. 6040–6042

[Doran und Gokhale 2011] Doran, Derek ; Gokhale, Swapna S.: Web robot detection techniques: overview and limitations. In: *Data Mining and Knowledge Discovery* 22 (2011), S. 183–210

[Fowler] Fowler, Martin: *Gui Architectures*. – URL https://www.martinfowler.com/eaaDev/uiArchs.html#Model-view-presentermvp

[Giles u. a. 2010] Giles, C L. ; Sun, Yang ; Councill, Isaac G.: Measuring the web crawler ethics. In: *Proceedings of the 19th international conference on World wide web*, 2010, S. 1101–1102

[Goll und Dausmann 2014] Goll, Joachim ; Dausmann, Manfred: *Architektur-und Entwurfsmuster der Softwaretechnik*. Springer, 2014

[Grasso u. a. 2013] Grasso, Giovanni ; Furche, Tim ; Schallhart, Christian: Effective web scraping with oxpath. In: *Proceedings of the 22nd international conference on world wide web*, 2013, S. 23–26

[Hubertz 2016] Hubertz, Johannes: *Softwaretests mit Python*. Springer-Verlag, 2016

[Ingeno 2018] Ingeno, Joseph: *Software Architect's Handbook: Become a successful software architect by implementing effective architecture concepts*. Packt Publishing Ltd, 2018

[ISO/IEC25010:2011 2011] ISO/IEC25010:2011: Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models / International Organization for Standardization. Genf, März 2011. – Standard

[Jorgensen 2013] Jorgensen, Paul C.: *Software testing: a craftsman's approach*. Auerbach Publications, 2013

[Karthikeyan u. a. 2019] Karthikeyan, T ; Sekaran, Karthik ; Ranjith, D ; Balajee, JM u. a.: Personalized content extraction and text classification using effective web scraping techniques. In: *International Journal of Web Portals (IJWP)* 11 (2019), Nr. 2, S. 41–52

[Khder 2021] Khder, Moaiad A.: Web Scraping or Web Crawling: State of Art, Techniques, Approaches and Application. In: *International Journal of Advances in Soft Computing & Its Applications* 13 (2021), Nr. 3

[Krotov und Silva 2018] Krotov, Vlad ; Silva, Leiser: Legality and ethics of web scraping. (2018)

[Mehta u. a. 2020] Mehta, Kunal ; Salvi, Maya ; Dand, Rumil ; Makharia, Vineet ; Natu, Prachi: A comparative study of various approaches to adaptive web scraping. In: *ICDSMLA 2019: Proceedings of the 1st International Conference on Data Science, Machine Learning and Applications* Springer (Veranst.), 2020, S. 1245–1256

[Mitchell 2018] Mitchell, Ryan: *Web scraping with Python: Collecting more data from the modern web*. Ö'Reilly Media, Inc.", 2018

[Moradi und Keyvanpour 2015] Moradi, Mohammad ; Keyvanpour, MohammadReza: CAPTCHA and its Alternatives: A Review. In: *Security and Communication Networks* 8 (2015), Nr. 12, S. 2135–2156

[Nadee und Prutsachainimmit 2018] Nadee, Winai ; Prutsachainimmit, Korawit: Towards data extraction of dynamic content from JavaScript Web applications. In: *2018 International Conference on Information Networking (ICOIN)*, 2018, S. 750–754

[Pawlik 2023] Pawlik, Victoria: *Genutzte Informationsquellen in der Bevölkerung 2023*. https://de.statista.com/statistik/daten/studie/171257/umfrage/normalerweise-genutzte-quelle-fuer-informationen/. 2023. – Accessed: 2023-9-26

[Pohl 2010] Pohl, Klaus: *Requirements engineering: fundamentals, principles, and techniques*. Springer Publishing Company, Incorporated, 2010

[Pohl und Rupp 2021] Pohl, Klaus ; Rupp, Chris: *Basiswissen requirements engineering: Aus- und Weiterbildung nach IREB-Standard zum certified professional for requirements engineering foundation level*. dpunkt. verlag, 2021

[Posch u. a. 2012] Posch, Torsten ; Birken, Klaus ; Gerdom, Michael: *Basiswissen Softwarearchitektur: verstehen, entwerfen, wiederverwenden*. Dpunkt. verlag, 2012

[Reitz 2023] Reitz, Kenneth: *Requests: HTTP for Humans*. 2023. – URL https://requests.readthedocs.io/en/latest/

[Richardson 2023] Richardson, Leonard: *Beautiful Soup Documentation.* 2023. – URL https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[Siegel und Alexa 2020] Siegel, Melanie ; Alexa, Melpomeni: Sentiment-Analyse deutschsprachiger Meinungsäußerungen. In: *Wiesbaden: Springer. Doi* 10 (2020), S. 978–3

[Sirisuriya u. a. 2015] Sirisuriya, De S. u. a.: A comparative study on web scraping. (2015)

[Snell und Menaldo 2016] Snell, James ; Menaldo, Nicola: Web scraping in an era of big data 2.0. In: *Bloomberg Law News* (2016)

[Starke und Hruschka 2023] Starke, Gernot ; Hruschka, Peter: *arc42 Documentation.* 2023. – URL https://docs.arc42.org/

[Tarannum 2019] Tarannum, Tasnuva: *Cleaning of web scraped data with Python*, Dissertation, 2019

[Zhao 2017] Zhao, Bo: Web scraping. In: *Encyclopedia of big data* 1 (2017)

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 5. Oktober 2023 Leon Marlo König