Duale Hochschule Baden-Württemberg

Stuttgart Campus Horb

Konzeption und prototypische Implementierung einer Search Engine in einer Microservice-Architektur

T3300 - Bachelorarbeit

---

eingereicht von:	Moris Kotsch
Matrikelnummer:	1317681
Kurs:	TINF2018
Studiengang:	Informatik
Hochschule:	DHBW Stuttgart Campus Horb
Ausbildungsfirma:	ENISCO by FORCAM GmbH
Ausbildungsleiterin:	Dipl.-Betriebsw. Angela Rasch
Unternehmen der Bachelorarbeit:	ENISCO by FORCAM GmbH
Betrieblicher Betreuer:	Dipl.-Ing. (FH) Franziska Simmank
Gutachter der DHBW:	Prof. Dr. phil. Antonius van Hoof
Bearbeitungszeitraum:	07.06.2021 - 31.08.2021

Freudenstadt, 30. Juli 2021

Sperrvermerk

Die vorliegende Praxisarbeit zum Thema „Konzeption und prototypische Implementierung einer Search Engine in einer Microservice-Architektur“ beinhaltet interne vertrauliche Informationen der Firma ENISCO by FORCAM GmbH. Die Weitergabe des Inhaltes der Arbeit und eventuell beiliegender Zeichnungen und Daten im Gesamten oder in Teilen ist untersagt. Es dürfen keinerlei Kopien oder Abschriften - auch in digitaler Form - gefertigt werden. Ausnahmen bedürfen der schriftlichen Genehmigung der Firma ENISCO by FORCAM GmbH.

Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich:

1. dass ich meine Praxisarbeit mit dem Thema **„Konzeption und prototypische Implementierung einer Search Engine in einer Microservice-Architektur“** ohne fremde Hilfe angefertigt habe;
2. dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe;
3. dass ich meine Praxisarbeit bei keiner anderen Prüfung vorgelegt habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Freudenstadt, 30. Juli 2021

Moris Kotsch

Zusammenfassung

ToDo

Abstract

ToDo

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Enisco und E-MES 1

1.2 Aufgabenstellung 1

1.3 Vorgehensweise und Aufbau der Arbeit 3

2 Grundlagen **4**

2.1 Software-Architekturen 4

2.1.1 Architektur-Prinzipien 5

2.1.2 Monolithische und verteilte Architekturen 7

2.2 Microservice-Architektur 8

2.2.1 Kommunikation zwischen Microservices 9

2.2.2 Microservice - Anti-Pattern 12

2.3 Suchfunktionen in modernen Informationssystemen 15

2.3.1 Mehrwert einer Suchfunktion 16

2.3.2 Volltextsuche 16

2.3.3 Facettierte Suche 20

2.3.4 Semantische Suche 22

2.3.5 Relevanzbestimmung 24

3 Suchumfang in MCC **26**

3.1 Umfang der Volltextsuche 26

3.2 Umfang der facettierten Suche 26

3.2.1 SCADA 26

3.2.2 PCS 26

3.3 Umfang einer facettierten Volltextsuche 26

3.4 Umfang einer semantischen Suche 26

4 Konzeption **27**

4.1 Konzeptionskriterien 27

4.2 Technische Umsetzung einer Volltextsuche 27

4.2.1 Search Engine 27

4.2.2 DBMS-interne Volltextsuche 27

4.3 Auswahl einer Search Engine . . . 27
4.3.1 Apache Lucene . . . 27
4.3.2 Apache Solr . . . 27
4.3.3 Elasticsearch . . . 28
4.3.4 Vergleich . . . 28
4.4 Datenaktualisierung zwischen Microservices und Search Engine . . . 28
4.4.1 Clientseitige Aktualisierung . . . 28
4.4.2 Pull-or-Push Aktualisierung . . . 28
4.4.3 Change-Data-Capture Aktualisierung . . . 28
4.4.4 Vergleich . . . 28
4.5 Gesamtkonzept . . . 28

5 Prototypische Umsetzung **29**
5.1 Client . . . 29
5.2 Search Service . . . 29
5.3 Anbindung an Apache Kafka . . . 29

6 Fazit und Ausblick **30**

Literaturverzeichnis **31**

Abbildungsverzeichnis

2.1 Zusammenspiel von loser Kopplung und hoher Kohäsion - [Vog09] . . . 6
2.2 Ablauf einer Anfrage mit der Service-Discovery - in Anlehnung an [Mic19] 10
2.3 Funktion eines Message Brokers - [Mic19] 11
2.4 Ergebnisse der Suchanfrage „Studenten“ - entnommen aus [Eni21a][1] . 17
2.5 Beispiel für ein Stichwortverzeichnis [wek14] 18
2.6 Beispielhafter Ablauf der Transformationsschritte - in Anlehnung an [Seb17] 19
2.7 Analyzer bei Indexierung und Abfrage - in Anlehnung an [Seb17] 20
2.8 Facettierte Suche bei Schubert Motors [Sch21] 21
2.9 Ergebnis einer semantischen Suche in Google 23

Abkürzungsverzeichnis

E-MES	Enisco Manufacturing Execution System
IDF	Inverse Document Frequency
MCC	Manufacturing Control Cloud
MES	Manufacturing Execution System
NLP	Natural Language Processing
QoS	Quality of Service
REST	Representational State Transfer
TF	Term Frequency
TF-IDF	Term Frequency - Inverse Document Frequency

Kapitel 1

1 Einleitung

Im folgenden Kapitel wird die betreuende Firma mit dem dazugehörigem Kernprodukt, die Aufgabenstellung und das geplante Vorgehen erläutert. Um das Studieren der Arbeit zu erleichtern, wird ein Überblick über den Aufbau der Arbeit gegeben.

1.1 Enisco und E-MES

Die vorliegende Arbeit wurde im Rahmen einer Bachelorarbeit bei der Firma Enisco verfasst. Enisco GmbH & Co. KG wurde 2015 als Tochtergesellschaft der Eisenmann SE gegründet und ist mittlerweile ein eigenständiges Unternehmen, welches unter dem Namen „Enisco by Forcam GmbH[1]“ agiert.

Als Kernprodukt vertreibt die Firma Enisco das Produktionsleitsystem „Enisco Manufacturing Execution System (E-MES)“. Dieses wird für die Überwachung und Steuerung von Produktionsanlagen eingesetzt. E-MES vernetzt dabei die Anlage sowohl horizontal, über den gesamten Fertigungsprozess, als auch vertikal, über alle Prozessebenen hinweg und bildet so ein System, welches zwischen Unternehmensebene (ERP, engl. für Enterprise Ressource Planning) und Steuerungsebene (PLC, engl. für Programmable Logic Controller) agiert. [Eni21b]

E-MES ist dafür modular aufgebaut und besitzt als Basis ein Platform-Modul. Dieses beinhaltet alle Grundlagen und Schnittstellen für die Installation weiterer Module. Die zusätzlichen Module (engl. Add-Ons) ergänzen E-MES um bestimmte Funktionen und können kundenspezifisch installiert und konfiguriert werden.

1.2 Aufgabenstellung

Derzeit erfährt das aktuelle Produktionsleitsystem E-MES eine Neugestaltung. Dabei wird von einer monolithisch betriebenen, modularen 3-Schichten-Architektur auf eine verteilte Microservice-Architektur gewechselt. Ein Wechsel der Architektur beruht auf dem Eintritt der Muttergesellschafft Forcam GmbH in die „Open Industry 4.0 Alliance“.

[1] Im Folgenden wird aus Gründen der Lesbarkeit auf die Rechtsform der Enisco by Forcam GmbH verzichtet

Durch den Zusammenschluss von mehreren Unternehmen aus dem Bereich „Industrie 4.0“ können einheitliche Schnittstellen definiert werden, um so die Interoperabilität zwischen den Softwarelösungen der beteiligten Firmen zu stärken [Ope21]. Um die benötigte Interoperabilität zu ermöglichen, setzen die beteiligten Unternehmen vermehrt auf Technologien wie Docker und Kubernetes. Um nun auch die Neugestaltung von E-MES in diesem Umfeld anzubieten, wurde sich für eine verteilte Microservice-Architektur entschieden. Neben der Architektur wird auch der Produktname von „E-MES“ in den Produktnamen „Manufacturing Control Cloud (MCC)“ abgeändert.

Im Zuge der Neugestaltung von E-MES werden neue Funktionalitäten, wie die Suchfunktion integriert. Eine Suchfunktion in modernen Informationssystemen wird von den Benutzern als gewohnter Komfort wahrgenommen. Auftretende Problemstellungen können dabei mithilfe von Suchabfragen gelöst werden. Solch eine Funktion liefert dem Benutzer eine Liste mit Suchergebnissen, aus der er den geeignetsten Treffer auswählen kann.

Für die Umsetzung einer Suchfunktion in einem Manufacturing Execution System (MES) muss festgelegt werden, welche Funktionen und Inhalte des MES von der Suchfunktion abgedeckt werden sollen. Neben der Suche nach Funktionalitäten des Systems, kann es auch hilfreich sein, nach bestimmten „Objekten“ innerhalb des Systems zu suchen. Solche Objekte können zum Beispiel in Form von eindeutigen Aufträgen oder Maschinen in einem MES vorkommen. Gibt der Benutzer die Kennung eines Objektes in das Suchfeld ein, sollen ihm alle Funktionen und Informationen bezüglich dieses Objektes angezeigt werden.

Neben der Anforderungsklärung bezüglich der Suchoptionen und der Granularität der Suchanfragen, gilt es auch eine Konzeption für die Integration einer Search Engine zu entwerfen. Hierbei sind die besonderen Anforderungen zu beachten, welche durch die Einführung der verteilten Microservice-Architektur entstanden sind. So ist zu klären, welche Strategie für die Datenaktualisierung zwischen einer Search Engine und den Datenhaltungsschichten der einzelnen Services den Anforderungen am besten entspricht. Anhand von selbstgewählten technischen und lizenzbezogenen Kriterien sollen diesbezüglich Strategien und auch potentiell geeignete Search Engines verglichen werden. Bei der Konzeption für die Integration einer Search Engine sind monolithische Seiteneffekte, die durch die Missachtung von Prinzipien der Microservice-Architektur entstehen könnten, zu vermeiden.

Das erstellte Konzept gilt es anschließend mithilfe einer Proof of Concept - Anwendung umzusetzen.

1.3 Vorgehensweise und Aufbau der Arbeit

Die Vorgehensweise und die schriftliche Ausarbeitung der vorliegenden Arbeit gliedert sich in drei Hauptteile. Als Vorarbeit für die eigentliche Bearbeitung werden in **Kapitel 2** die theoretischen Grundlagen über die verteilte Microservice-Architektur, den Search Engines und dem Entwicklungsstand von MCC beschrieben.

Im ersten Schritt wird definiert, mit welchem Suchumfang die Search Engine innerhalb von MCC nach Funktionen und Objekten agieren soll. Da zum Zeitpunkt der Erstellung dieser Arbeit noch keine produktreife Version von MCC existiert, wird sich an dem Produktumfang und den Funktionalitäten des aktuellen Produktes E-MES orientiert. Im **Kapitel 3** wird der Suchumfang für die Suchfunktionalität definiert. Hierbei wird bei E-MES analysiert, welche Objekte innerhalb des Systems „suchbar“ gemacht werden sollen.

Ein weiterer Schritt ist die Konzeption für die Integration einer Search Engine in MCC. Es werden hierbei in **Kapitel 4** verschiedene Search Engines anhand von technischen und lizenzbezogenen Kriterien miteinander verglichen. Ebenso werden verschiedene Möglichkeiten der Datenaktualisierung zwischen einer Search Engine und den Datenhaltungsschichten der einzelnen Services erläutert und anhand der erarbeiteten Entscheidungskriterien miteinander verglichen. Vorbereitend für die prototypische Umsetzung wird zusätzlich ein Gesamtkonzept erstellt.

Anschließend an die Konzeption folgt in **Kapitel 5** die prototypische Umsetzung anhand einer Proof of Concept - Anwendung. Hierfür wird zunächst festgelegt, welchen Umfang jene prototypische Umsetzung besitzen soll und ob bereits Softwareteile aus MCC verwendet werden können.

Abgeschlossen wird die Arbeit mit einem Fazit und einem Ausblick **(Kapitel 6)**.

Kapitel 2

2 Grundlagen

Um eine Konzeption und prototypische Umsetzung einer Volltextsuche in MCC zu ermöglichen, wird in folgendem Kapitel auf die architekturellen Hintergründe von MCC eingegangen. Hierfür werden zu Beginn allgemein gültige Architektur-Prinzipien erläutert, welche auch bei der Integration einer Volltextsuche berücksichtigt werden müssen und somit bei der Auswahl eines geeigneten Konzeptes von Bedeutung sind.

Aufbauend auf den Architektur-Prinzipien wird die verwendete Microservice-Architektur erläutert, wobei neben einer allgemeinen Einführung in die Architektur auch auf die Kommunikation von Microservices untereinander eingegangen wird. Um mögliche Fehlkonzeptionen zu vermeiden, werden häufig auftretende Anti-Pattern aufgezeigt, welche unter Umständen zu monolithischen Seiteneffekten führen können.

Grundlegend wird noch der Begriff „Volltextsuche“ vorgestellt. Hierfür wird auf den Mehrwert und die verschiedenen Umsetzungsmöglichkeiten eingegangen. Ergänzend wird in **Kapitel 3** auf eine Volltextsuche im Umfeld von MCC eingegangen. Neben dem Umfang der Suchergebnisse wird auch deren Relevanzbestimmung erläutert.

2.1 Software-Architekturen

> Die Software-Architektur eines Systems ist die Menge von Strukturen, die benötigt werden, um Entscheidungen über das System zu treffen, welche die Software-Elemente, die Relationen zwischen ihnen und die Eigenschaften von beiden betreffen. ~ Len Bass [BCK13, S. 4]

Wie aus der Definition von Len Bass zu entnehmen ist, beschreibt eine Software-Architektur die Eigenschaften und Beziehungen von Software-Bausteinen zueinander [BCK13, S. 4]. Ein Software-Baustein wird hierbei als eine Teil-Komponente der gesamten Software betrachtet und wird bei der Erstellung einer Architektur als elementarer Bestandteil angesehen. Dabei wird ein Software-Baustein nicht näher spezifiziert, sondern als Komponente betrachtet, dessen konkrete Implementierung für die Architektur nicht von Bedeutung ist. Der Fokus einer Software-Architektur liegt auf den Schnittstellen der Software-Bausteine, über welche die Bausteine miteinander kommunizieren können.

2.1.1 Architektur-Prinzipien

Für das Erstellen einer guten Software-Architektur wurden von Vogel [Vog09, S. 128-147] einige Grundprinzipien definiert. Diese Prinzipien sollten bei der Erstellung einer Software-Architektur beachtet werden: [Vog09, S. 128-147]

Lose Kopplung:

Der Kern einer Software-Architektur besteht aus der Beschreibung der Software-Bausteine eines Software-Systems und deren Interaktionen zueinander. Unter dem Begriff Kopplung versteht man hierbei die Beziehung unter den Software-Bausteinen einer Software-Architektur. Eine Kopplung charakterisiert demnach die Interaktionen der Software-Bausteine.

Eine starke Kopplung von Software-Bausteinen hat zur Folge, dass beim Verstehen und Ändern eines Software-Bausteines auch zwingend weitere Software-Bausteine verstanden und geändert werden müssen. Um jenes Problem zu umgehen, besagt das Prinzip der losen Kopplung, dass die Kopplung zwischen Software-Bausteinen möglichst niedrig gehalten werden sollen.

Um eine lose Kopplung in einer Architektur zu erreichen, ist die Einführung von Schnittstellenabstraktionen ein wichtiger Aspekt. Dabei werden die Implementierungsinformationen hinter den Schnittstellen verborgen. Durch die Begrenzung von Schnittstellenelementen und der Häufigkeit des Austauschs der Schnittstellenelemente, kann eine Kopplung von Software-Bausteinen kontrollierbar gemacht werden.

Hohe Kohäsion:

Im Gegensatz zur Kopplung, in welcher die Beziehungen zwischen Software-Bausteinen gemeint ist, versteht man unter dem Begriff Kohäsion die Abhängigkeiten innerhalb eines Software-Bausteins.

Beim Prinzip der hohen Kohäsion ist das Ziel die Abhängigkeiten innerhalb eines Software-Bausteins möglichst hoch zu gestalten. Wie bei der losen Kopplung geht es auch hier um die lokale Änderbarkeit und Verstehbarkeit von Software-Bausteinen.

Wie in Abbildung 2.1 zu erkennen, stehen Kopplung und Kohäsion normalerweise miteinander in einer Wechselbeziehung. Hierbei gilt, dass je höher die Kohäsion individueller Software-Bausteine einer Architektur ist, desto geringer ist die Kopplung zwischen den Software-Bausteinen. Schematisch ist dieser Zusammenhang in Abbildung 2.1 abgebildet, worin zu erkennen ist, das eine Gesamtstruktur mit einer hohen Kohäsion und einer losen Kopplung (rechte Seite) eine höhere Übersichtlichkeit besitzt.

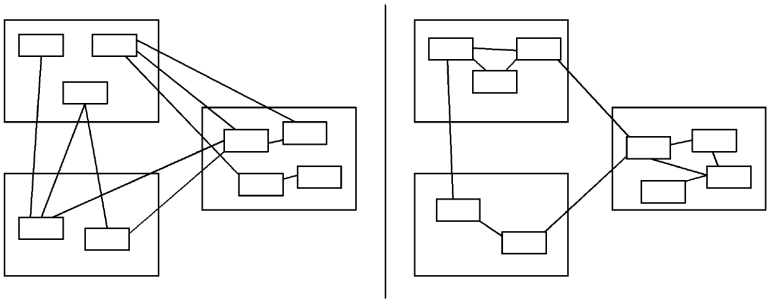

Abb. 2.1: Zusammenspiel von loser Kopplung und hoher Kohäsion - [Vog09]

Entwurf für Veränderung:

Durch den stetigen Wandel von Software-Systemen in Form von Anforderungen und Technologien, ist es von Vorteil solche Änderungen bereits in der Phase der architekturellen Konzeption zu berücksichtigen. Das Prinzip des Entwurfs für Veränderung (englisch: Design for Change) sieht nun vor, dass man vorhersehbare Änderungen architektonisch vorausplant. Dabei sollte man versuchen, die Architektur so zu entwerfen, dass man leicht mit den wahrscheinlichen Änderungen eines Software-Systems umgehen kann.

Separation of Concerns:

Abgeleitet von dem römischen Prinzip „Teile und herrsche“ wird beim Prinzip Separation of Concerns ausgesagt, dass ein Software-System in individuelle Software-Bausteine zerlegt werden soll.

Separation of Concerns unterstützt hierbei die Modularisierung eines Software-Systems. Es geht darum Teile eines Software-Systems zu identifizieren, welche für bestimmte Angelegenheiten, Aspekte und Aufgaben verantwortlich sind. Jene Teile werden dann als eigene Software-Bausteine gekapselt. Eine Zerteilung des Gesamtsystems in relativ unabhängige Einzelteile ermöglicht zudem noch die Verteilung von Verantwortlichkeiten für verschiedene Software-Bausteine und auch das parallele Arbeiten an dem Software-System durch mehrere Entwickler wird dadurch ermöglicht.

Durch das Aufteilen des Software-Systems in relativ unabhängige Software-Bausteine, werden auch die Prinzipien lose Kopplung und hohe Kohäsion begünstigt.

Information Hiding:

Das Prinzip Information Hiding sagt aus, dass man einem Klienten nur die für die Bearbeitung eines Problems notwendigen Informationen zeigen soll. Dies erleichtert die Gliederung und das Verständnis von komplexen Software-Systemen. Die restlichen Informationen sollen nach außen hin verborgen bleiben. Ermöglicht wird solch ein „geheim halten“ von Informationen durch die Bereitstellung von

definierten Schnittstellen, über welche nur bestimmte Informationen zu erreichen sind.

Abstraktion:

Als übergeordnetes Prinzip dient eine Abstraktion dazu, ein komplexes System verständlicher zu machen. Dazu werden wichtige Aspekte identifiziert und unwichtige Details vernachlässigt. Im Bereich der Software-Architektur gilt die Schnittstellenabstraktion als Teilprinzip der Abstraktion. Hierbei liegen die Schnittstellen im Fokus, welche für das Zustandekommen und die Qualität von Beziehungen verantwortlich sind.

Solch eine Schnittstellenabstraktion in einem Software-System ist eng verbunden mit dem Prinzip der losen Kopplung und dem Information Hiding. Ein Aspekt für den starken Zusammenhang zwischen der Abstraktion und dem Information Hiding ist die Portabilität von Software-Systemen. So sollte eine Architektur oder ihre Software-Bausteine auch in anderen Umgebungen verwendbar sein. Um solch eine Plattformunabhängigkeit sicherzustellen, werden Abstraktionen verwendet, die ein Information Hiding der Platform-Details leisten.

Modularität:

Das Modularitätsprinzip, welches bereits auch in den Beschreibungen der anderen Prinzipien vorkam, definiert die Aufteilung eines Systems in klar definierte Software-Bausteine mit abgegrenzten funktionalen Verantwortlichkeiten. Die Modularität ist dabei eine Kombination aus den Prinzipien Abstraktion, Separation of Concerns und Information Hiding, welche bei der Umsetzung der Prinzipien der losen Kopplung und der hohen Kohäsion kombiniert werden.

Auch für die spätere Konzeption einer Volltextsuche in einer Microservice-Architektur werden die eingeführten Prinzipien als Grundlage dienen.

2.1.2 Monolithische und verteilte Architekturen

Bei der Neugestaltung von E-MES wird von einer monolithischen 3-Schichten-Architektur auf eine verteilte Microservice-Architektur gewechselt.

In einer monolithischen Architektur wird die gesamte Architektur in nur einem Software-Baustein zusammengefasst. Dadurch erfolgt keine explizite Gliederung in Teilsysteme und Architektur-Prinzipien, wie lose Kopplung und Separation of Concerns sind nur schwer umsetzbar [Vog09, S. 216]. Zu finden sind monolithische Architekturen oftmals in Altsystemen, welche oft über Jahrzehnte gewachsen sind. Aufgrund der mangelnden Modularisierung steigt die Kompliziertheit des Systems. Die Wartung und Anpassung des Quellcodes wird erschwert [Pro12b]. Ein weiterer Nachteil der mangelnden Modularisierung ist die kaum mögliche nebenläufige Ausführung von Teilen des Systems

auf verschiedenen Rechnern [Pro12b]. Somit kann eine horizontale Skalierung nicht ermöglicht werden, und eine effiziente, lastverteilende Programmausführung ist nicht gegeben.

Die Architektur von MCC wird eine verteilte Struktur aufweisen. Hierbei werden Teile des Gesamtsystems in unterschiedliche Software-Bausteine aufgeteilt. Eine Modularisierung der Software ist dadurch möglich und Architekturen-Prinzipien, wie lose Kopplung und Separation of Concerns sind umsetzbar [Vog09]. Durch die strikte Aufteilung der Geschäftslogik kann auch die Komplexität aufgeteilt werden. Somit können die einzelnen Software-Bausteine mit wenig Aufwand angepasst oder erweitert werden. Durch die Modularisierung von verteilten Architekturen kann die Ausführung bestimmter Aufgaben auf redundanter Hardware nebenläufig erfolgen [Pro12a]. Durch jene horizontale Skalierung kann eine effiziente und lastverteilende Programmausführung erfolgen, welche auch zur Ausfallsicherheit des Gesamtsystems beiträgt [Pro12a].

2.2 Microservice-Architektur

Bei der Neugestaltung des MES der Firma Enisco wird auf eine Microservice-Architektur aufgebaut, welche eine verteilte Struktur aufweist.

Die Kernelemente dieser Architektur sind die Microservices, welche der Modularisierung der Software dienen. Somit ist eine Aufteilung des Gesamtsystems in verschiedene Software-Bausteine möglich. Ein Software-Baustein stellt dabei jeweils eine Funktionalität des Gesamtsystems dar. Im Gegensatz zu einer monolithischen Architektur läuft das Gesamtsystem nicht innerhalb eines Prozesses, sondern auf verschiedenen Prozessen. Dabei wird jedem Software-Baustein ein eigener Prozess zugeordnet. Jene Prozesse können nun nahezu beliebig auf verschiedene Rechner verteilt und durch Replizierung ausfallsicher gemacht werden. [Gar18]

Neben den Vorteilen der horizontalen Skalierung ergeben sich aus der Aufteilung des Gesamtsystems in unterschiedliche Software-Bausteine auch Auswirkungen auf die Entwicklungsorganisation. So wird beim Umgang mit Microservices nach der Unix-Philosophie von Ken Thompson „Do one thing and do it well“ [ION21] gearbeitet. Durch die Modularität von Microservices können diese von unterschiedlichen Entwicklerteams unabhängig entwickelt werden. Durch die Abstraktion der Microservices können diese mit unterschiedlichen Technologien und Programmiersprachen implementiert werden. Auch der Datenhaushalt kann von jedem Microservice separat verwaltet werden. Zudem wird die Einarbeitung eines Entwicklers in die Codebasis reduziert, da durch die Aufteilung weniger Code verstanden werden muss.

Die Microservice-Architektur berücksichtigt die Architektur-Prinzipien Separation of Concerns, Information Hiding und Modularität und gewährleistet somit eine lose Kopp-

lung zwischen den Microservices. Innerhalb der Microservices entsteht dadurch eine hohe Kohäsion.

Da die jeweiligen Microservices repliziert auf verschiedenen Rechnern laufen können, ist die Kommunikation zwischen den Microservices schwieriger als bei einem monolithischen System. Auf die Kommunikation zwischen Microservices wird in Unterabschnitt 2.2.1 näher eingegangen.

Eine Herausforderung bei der Konzeption einer Microservice-Architektur ist die Vermeidung von Abhängigkeiten, welche eine lose Kopplung der Microservices verhindern würden. Um solche monolithischen Seiteneffekte zu vermeiden, werden in Unterabschnitt 2.2.2 die häufigsten Microservice-Anti-Pattern aufgezeigt.

2.2.1 Kommunikation zwischen Microservices

Auch wenn das Ziel einer Microservice-Architektur ist, dass einzelne Funktionalitäten des Gesamtsystems in getrennte Microservices gekapselt werden, müssen diese miteinander kommunizieren. Aufgrund der Modularität können die Microservices horizontal skaliert und auf verschiedenen Rechnern betrieben werden. Dies erhöht die Komplexität bei der Kommunikation der Microservices untereinander. [Mic19]

Bei der Wahl der Kommunikation zwischen Microservices kann zwischen einer synchronen und asynchronen Kommunikation entschieden werden.

Synchrone Kommunikation

Bei der synchronen Kommunikation handelt es sich um eine eins-zu-eins Kommunikation, bei der eine Anfrage geschickt und auf eine Antwort gewartet wird. Klassischerweise erfolgt die Kommunikation über HTTP mit einer REST-Schnittstelle. Representational State Transfer (REST) ist hierbei eine Spezifikation, wie eine über HTTP kommunizierende API konzipiert werden soll [Mic19]. Eine solche API sollte demnach vordefinierte HTTP-Methoden implementiert haben. Unter anderem sind das Methoden wie GET, POST, PUT und DELETE. Bei einer GET-Anfrage werden hierbei Ressourcen angefragt. Soll ein neuer Datensatz übermittelt werden, wird die POST-Methode verwendet. Zur Änderung eines bestehenden Datensatzes gibt es die PUT-Methode und mit der DELETE-Methode kann ein Datensatz entfernt werden. Jeder Datensatz bekommt hierbei einen eigenen Endpunkt, und die jeweiligen Anfragen können mit URL-Parametern und Query-Parametern spezifiziert werden. Das Standard-Datenformat bei REST ist JSON.

Bei der Kommunikation von zwei Microservices über die jeweiligen REST-Schnittstellen muss die Adresse des anderen Microservice bekannt sein. Durch die Verteilung der Microservices auf unterschiedliche Rechner in Folge einer horizontalen Skalierung kann

es während des Betriebes vorkommen, dass einzelne Microservices auf zum Beispiel Rechner A gestoppt und auf Rechner B wieder gestartet werden. Dadurch ändern sich auch die Adressen der Microservices. Da die Verwaltung der Adressen ab einer Vielzahl an Microservices nicht mehr trivial ist, wird in einer Microservice-Architektur eine sogenannte Service-Discovery eingesetzt. Die Service-Discovery ist eine Software, bei der sich alle neuen Microservices registrieren. Bei einem REST-Aufruf wird dann zuerst eine Liste mit allen verfügbaren Adressen abgerufen.

Wie in Abbildung 2.2 dargestellt ist, wird bei einer Anfrage eines Services, jene Anfrage zunächst an einen Router-Service geleitet. Nach dem Abfragen der Adresse mithilfe der Service-Discovery wird gezielt der entsprechende Service beziehungsweise dessen Instanz mit einer REST-Anfrage angesprochen.

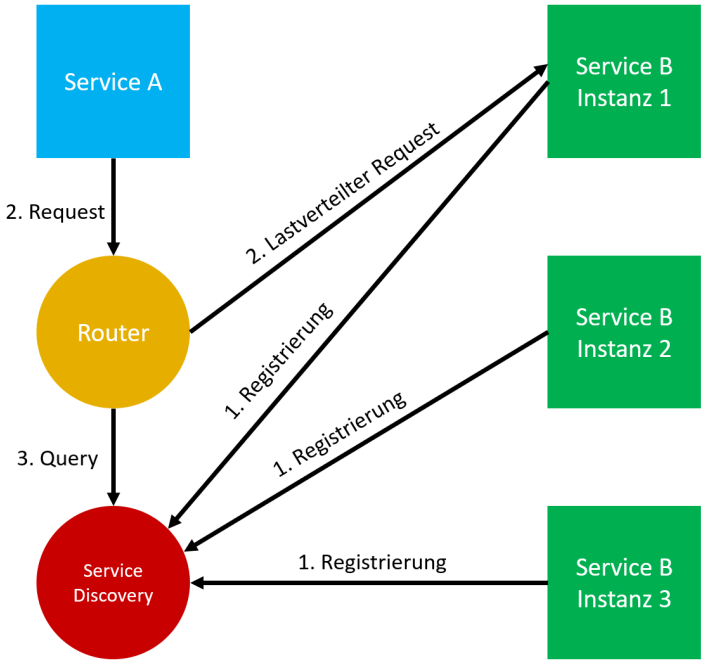

Abb. 2.2: Ablauf einer Anfrage mit der Service-Discovery - in Anlehnung an [Mic19]

Asynchrone Kommunikation

Anders als bei der synchronen Kommunikation wird bei der asynchronen Kommunikation nicht auf jede Anfrage eine Antwort erwartet. Kern einer asynchronen Kommunikation ist ein zentraler Nachrichtenkanal, auf dem Nachrichten ausgetauscht werden. Ein Service schickt Nachrichten an solch einen Nachrichtenkanal, auch Message Broker genannt, und wenn ein anderer Service an der Nachricht interessiert ist, kann er diese konsumieren. Eine Nachricht kann dabei auch von mehreren Services konsumiert werden.

Zu den bekanntesten Message Brokern zählen RabbitMQ und Apache Kafka. Während RabbitMQ leichtgewichtiger ist als Apache Kafka, hat letzteres mehr Features und weist eine größere Stabilität auf. Die Verantwortlichkeiten innerhalb des Systems werden bei

der Verwendung eines Message Brokers umgedreht. Ein Service wird nun nicht mehr explizit von einem anderen Service aufgerufen, sondern schickt seine Nachrichten an den Message Broker, ohne zu wissen, wer diese Nachrichten konsumiert. Sender und Empfänger sind somit lose gekoppelt.

Zur Einteilung der Nachrichten werden in Message Brokern sogenannte Message Queues verwendet. Über eine Message Queue, welche einen bestimmten Topic besitzt, können Services dann entweder Nachrichten senden oder sich als Konsumenten an diesem Topic registrieren. In Abbildung 2.3 ist die Funktionsweise von Apache Kafka abgebildet. Durch den Publish/Subscribe-Mechanismus kann Service A in diesem Beispiel sowohl Publisher von zwei Topics sein, als auch Konsument von einem Topic.

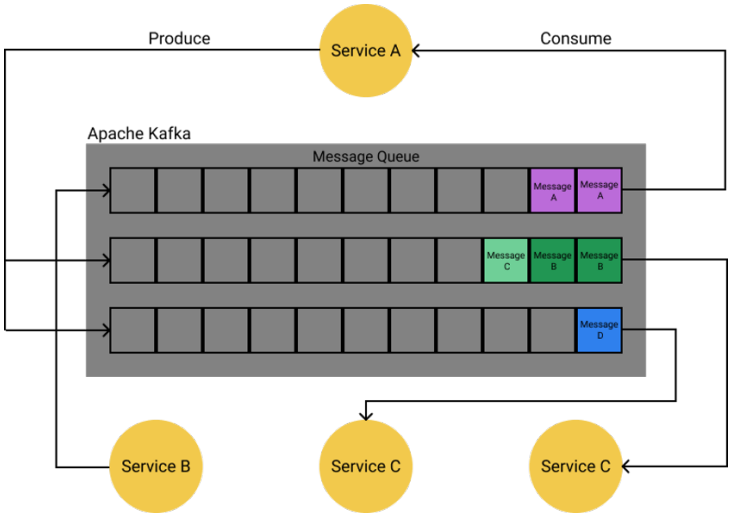

Abb. 2.3: Funktion eines Message Brokers - [Mic19]

Die Nachrichten werden in der Message Queue zwischengespeichert, bis die registrierten Konsumenten diese Nachrichten gelesen haben. Wenn ein Service sich an einer Message Queue registriert, können verschiedene Servicequalitäten für die Übertragung der Nachrichten angegeben werden. Folgende drei „Quality of Service (QoS)“ können angegeben werden: [Apa21]

QoS 0 „At most once“:

Eine Nachricht wird genau einmal versendet. Nach dem Prinzip „fire and forget“ wird keine Rückbestätigung erwartet. Verwendet wird diese Servicequalität meist für Nachrichten, welche mit einer hohen Frequenz versendet werden. So ist es für viele Sensoren in einer IoT-Umgebung nicht erforderlich, dass jeder gemessene Wert erfolgreich übermittelt wird.

QoS 1 „At least once“:

Es wird sichergestellt, dass eine Nachricht genau einmal beim Empfänger ankommt. Hierzu werden die Nachrichten beim Sender solange zwischengespeichert, bis vom Empfänger eine Bestätigung (PUBACK) empfangen wird. Wird über eine gewisse

Zeitspanne keine Bestätigung empfangen, wird die Nachricht erneut versendet. Hierbei kann es vorkommen, dass ein Empfänger eine Nachricht mehrfach erhält. [Flo17]

QoS 2 „Exactly once“:

Eine Nachricht wird genau einmal zugesendet. Hierbei wird ein 4-Wege-Handshake zwischen Sender und Empfänger versendet. Durch diesen Ablauf ist eine sichere Übertragung der Nachricht gewährleistet. Durch den großen Overhead ist jedoch diese Servicequalität die langsamste Übertragungsmöglichkeit.

2.2.2 Microservice - Anti-Pattern

Um die Vorteile einer Microservice-Architektur im Bezug auf lose Kopplung und der damit eingehenden Unabhängigkeit der Microservices bezüglich Wartbarkeit und Erweiterbarkeit zu gewährleisten, ist es von Nöten die möglichen Anti-Pattern einer Microservice-Architektur zu kennen. Mit Hilfe von Anti-Pattern können nach Roth und Hafner [SM19] wiederkehrende Fehler bei der Softwareentwicklung identifiziert und in generalisierter Form dokumentiert werden. So können Anti-Pattern dabei helfen häufig auftretende Fehler von vornherein zu vermeiden.

In einer Studie von Tighilt et al. [TAM+20] wurden 27 Paper und 67 Open-Source Anwendungen, welche alle Bezug zur Microservice-Architektur haben, untersucht. Die Ergebnisse der Studie [TAM+20] ergeben eine Sammlung von häufig aufgetretenen Anti-Pattern, welche durch Tighilt et al. in vier Kategorien unterteilt wurden. Folgend werden die Kategorien mit Beispielen erläutert:

Design Anti-Pattern

In diese Kategorie gehören Anti-Pattern, welche bereits in der architektonischen Entwurfsphase einer Microservice-Architektur zu Fehlern führen. Beispiele sind hierfür das „Falscher Schnitt“ - Anti-Pattern und das „Zyklische Abhängigkeiten“ - Anti-Pattern.

Falscher Schnitt:

Beim „Falscher Schnitt“ - Anti-Pattern wird die Aufteilung des Gesamtsystems in unterschiedliche Microservices nicht nach Geschäftsfunktionen, sondern nach technischen Aspekten durchgeführt. Ähnlich wie bei einer monolithischen 3-Schichten-Architektur übernehmen dann ein Teil der Microservices die Präsentationsschicht, andere Microservices wiederum bedienen die Geschäftsschicht und die anderen bedienen die Datenzugriffsschicht. Durch dieses Vorgehen verliert die Architektur die lose Kopplung zwischen den Microservices, und die Modularität geht verloren. Auch müssen bei einer Änderung einer Geschäftslogik mehrere Microservice abgeändert werden, was zu einem Mehraufwand führt.

Verhindert werden kann dieses Anti-Pattern, indem man die Trennung des Gesamtsystems anhand der Geschäftsfunktionen durchführt. Jeder Microservice sollte von einem Team implementiert und gewartet werden können. Die Abhängigkeiten zu anderen Microservices sollten so gering wie möglich ausfallen, um eine lose Kopplung und die Modularität der Architektur zu gewährleisten.

Zyklische Abhängigkeiten:

Ein weiterer Fehler ist die Missachtung der losen Kopplung, indem man mit zyklischen Abhängigkeiten eine starke Gebundenheit an andere Microservices verursacht. Dadurch müssen die abhängigen Microservices sowohl bei der Implementierung als auch bei der Wartung berücksichtigt werden. Dies erhöht die Komplexität und die Wahrscheinlichkeit, dass Fehler auftreten.

Eine zyklische Abhängigkeit zwischen solchen Microservices kann verhindert werden, indem man stark voneinander abhängige Microservices in einen gemeinsamen Microservice zusammenfasst.

Implementation Anti-Pattern

Implementation Anti-Pattern entstehen aus der Art und Weise wie Microservices implementiert werden. Beispiele sind hierfür das „Festcodierte Endpunkte“ - Anti-Pattern.

Festcodierte Endpunkte:

Bei der synchronen Kommunikation zwischen Microservices werden in der Regel Anfragen über REST APIs gestellt. Hierfür müssen die IP-Adresse und der Port des anderen Microservices bekannt sein. Ein Ansatz für schnelle Laufzeiten und eine einfache Implementierung ist es, die Endpunkte fest im Quellcode zu hinterlegen. Dieser Ansatz ist jedoch bei einer größeren Anzahl von Microservices nicht mehr wartbar. Sobald sich ein Endpunkt eines Microservice ändert, müssen alle abhängigen Microservices geändert und neu deployed werden.

Abhilfe bringt die Einführung einer Service-Discovery (Siehe auch Abschnitt 2.2.1), bei welcher sich die einzelnen Services registrieren. Ein anfragender Service kann nun über diese Service-Discovery die aktuelle Adresse des anderen Services erhalten.

Deployment Anti-Pattern

Auch in der Bereitstellungsphase einer Microservice-Architektur können Fehler auftreten. Diese Anti-Pattern werden Deployment Anti-Pattern genannt. Beispiele sind hierfür das „Manuelle Konfiguration“ - Anti-Pattern und das „Zeitüberschreitungen“ - Anti-Pattern.

Manuelle Konfiguration:

In einer Microservice-Architektur müssen aufgrund der Verteilung der Microservices auf unterschiedliche Rechner verschiedene Konfigurationen durchgeführt werden. Eine manuelle Konfiguration von jedem Microservice führt zu unzähligen Konfigurationsdateien und Abhängigkeiten zu Umgebungsvariablen. Mit wachsender Anzahl von Microservices führt die manuelle Konfiguration zu einer ansteigenden Komplexität bei der Wartung der Konfigurationen.

Durch die Einführung eines Konfigurations-Servers kann dieses Anti-Pattern vermieden werden. Durch geeignete Konfigurationsmanagement-Tools können sämtliche Konfigurationen zentral verwaltet werden.

Zeitüberschreitungen:

In einer verteilten Microservice-Architektur kann es vorkommen, dass einzelne Services ausfallen und nicht mehr zur Verfügung stehen. Hat nun ein anderer Microservice eine Anfrage über eine REST API gestellt, kann es vorkommen, dass unter Umständen gar keine Antwort kommt. Hierfür kann der Entwickler Timeouts festlegen, nach welchen einem anfragenden Service mitgeteilt wird, dass ein anderer Service nicht erreichbar ist. Den richtigen Wert für solch einen Timeout zu wählen, hängt von der Art des Service ab. Ein zu kurzer Timeout kann dazu führen, dass langsame Service frühzeitig beendet werden. Bei einer zu langen Wahl des Timeouts kann es vorkommen, dass der Endnutzer zu lange warten muss, bis ihm signalisiert wird, dass der gewünschte Service nicht verfügbar ist.

Um dem entgegen zu wirken, verwendet man in einer Microservice-Architektur üblicherweise einen Circuit Breaker. Ein Circuit Breaker überwacht hierbei alle Instanzen einzelner Microservices und stellt fest, ob einzelne Timeouts bei der Kommunikation mit der jeweiligen Instanz überschritten wurden. Da solch ein Circuit Breaker als Proxy für alle Kommunikationskanäle verwendet wird, werden alle Anfragen an einen ausgefallenen Service mit einer Fehlermeldung beantwortet. Die Anfragen werden solange durch den Circuit Breaker blockiert, bis der ausgefallene Service wieder zur Verfügung steht. Auch der Timeout des Circuit Breaker muss vom Entwickler festgelegt werden.

Monitoring Anti-Pattern

Zu dem Betrieb einer Microservice-Architektur gehört auch die Überwachung der Änderungen im System. Hierbei kann es auch zu häufig auftretenden Anti-Pattern kommen. Beispiele sind hierfür das „Kein Gesundheitscheck“ - Anti-Pattern und das „Lokale Protokollierung“ - Anti-Pattern.

Kein Gesundheitscheck:

In der Natur der Microservices kann es vorkommen, dass einige Services über

gewisse Zeitspannen nicht erreichbar sind. Um ein kompletten Systemausfall zu vermeiden, dienen Mechanismen, wie der Circuit Breaker, dazu die anfragenden Services über das Überschreiten von Timeouts zu benachrichtigen.

Mit der Integration von Gesundheitschecks können solche Ausfälle von einzelnen Services im Voraus erkannt und abhängige Microservices darüber informiert werden, dass das Senden von Anfragen zu unterlassen ist. Für solche Gesundheitschecks werden API-Endpoints angeboten, welche periodisch abgefragt werden können.

Lokale Protokollierung:

Um eine spätere Fehlerdiagnose zu ermöglichen, speichern Microservices einige Informationen über Änderungen in Log-Dateien. Ein häufiger Fehler ist, dass solch eine Protokollierung lokal in den Datenbanken der einzelnen Microservices gespeichert werden. Die Herausforderung liegt nun bei der Analyse und Abfrage von Logs aus verschiedenen Microservices.

Mit einem verteilten Logging-Mechanismus können die Logs der unterschiedlichen Microservices in einem zentralen Speicherort gespeichert werden. Durch eine einheitliche Formatierung der Log-Einträge kann auch deren Abfrage und Analyse vereinfacht werden.

Auch bei der Integration einer Suchfunktion in eine Microservice-Architektur gilt es die vorgestellten Anti-Pattern zu beachten, um so monolithische Seiteneffekte zu vermeiden.

2.3 Suchfunktionen in modernen Informationssystemen

Aufgrund der fortlaufenden Digitalisierung und Themen wie „Industrie 4.0“ und „Internet of Things“ steigt die jährlich anfallende Datenmenge im weltweiten Internet. Das aufkommende Datenvolumen wird bereits für das Jahr 2025 auf 175 Zettabyte prognostiziert [F. 18]. Dies wäre ein Wachstum, vom Jahr 2018 bis zum Jahr 2025, um mehr als den Faktor fünf.

Neben den reinen Daten wachsen zunehmend auch die Anzahl der Websites und Web-Inhalten. So gibt es Stand 2021 rund 1,83 Milliarden Websites [Guy21], welche durch Suchmaschinen, wie Google gefunden werden können. Ein Nutzer kann hierbei einen Volltext als Sucheingabe eingeben und erhält eine Liste mit den entsprechenden Treffern sortiert nach der Relevanz für den Nutzer.

Solche Suchfunktionalitäten werden auch direkt auf den Websites angeboten, um den Besuchern so das Navigieren durch die Website und das Auffinden von Produkten und Dienstleistungen zu erleichtern. Nach einer Studie des eCommerce-Leitfadens [Aar]

geben 80% der Shopbesitzer an, dass die Suchfunktion auf ihrer Website ihnen „sehr wichtig“ ist. Demnach geben 97% der Befragten an, dass sie eine Suchfunktion im Shop anbieten.

Je nach Art der Benutzer und Umfang der Produkte und Informationen stehen bei der Umsetzung einer Suchfunktion zwei Möglichkeiten zur Verfügung. Eine Möglichkeit ist die Suche mit Hilfe einer Volltextsuche. Für einen Benutzer stellt solch eine Art der Suchfunktion einen bekannten Komfort dar, da diese Art bereits aus Web-Suchmaschinen, wie zum Beispiel Google bekannt ist. Eine weitere Möglichkeit bietet die Suche über verschiedene Filter. Hierbei werden die Details der Informationen über verschiedene Facetten definiert. Durch Filterung der Facetten können dann die Informationen gefunden werden.

Oft zu finden ist eine Kombination aus einer Volltextsuche und einer facettierten Suche. Hierbei wird dem Benutzer primär eine Volltextsuche in Form eines Eingabefeldes angeboten und anschließend kann über eine „Erweiterte Suche“ die Facettierung vorgenommen werden. Ein bekanntes Beispiel für die Verwendung einer Kombination beider Möglichkeiten ist der Onlineversandhändler „Amazon“. Durch die Eingabe eines Suchbegriffes wird dem Benutzer eine Auswahl geeigneter Suchtreffer angezeigt. Für eine weitere Eingrenzung der Suchergebnisse kann der Benutzer die Ergebnisse zum Beispiel anhand der Preise, Bewertungen oder Hersteller filtern.

2.3.1 Mehrwert einer Suchfunktion

Der Mehrwert bei der Einführung einer Suchfunktion in einer Website oder in einem Software-Produkt, wie MCC, liegt bei der Unterstützung der Benutzer beim Navigieren.

Aufgrund der wachsenden Anzahl der Websites und Web-Inhalten [Guy21] ist ein eigenständiges Navigieren durch das World Wide Web nicht mehr möglich. Durch die Verwendung von Suchmaschinen können Benutzer mit beliebigen Suchbegriffen nach expliziten Websites oder Web-Inhalten suchen.

Der Benutzerkreis von Software-Produkten, wie MCC, besteht überwiegend aus einem expliziten Benutzerkreis. Im Umfeld von MCC sind dies vor allem die Betreiber der jeweiligen Anlagen. Damit sich auch neue und unerfahrene Benutzer in einer solchen umfangreichen Software zurechtfinden, kann eine Suchfunktionalität den Einstieg erleichtern. So kann der Produktumfang von MCC benutzt werden, ohne die genaue Struktur und den kompletten Funktionsumfang zu kennen.

2.3.2 Volltextsuche

Eine Möglichkeit nach Produkten oder Informationen zu suchen, ist die Volltextsuche. Hierbei kann der Benutzer einen beliebigen Freitext in ein Suchfeld eingeben und erhält

eine Auflistung der Suchtreffer, welche den eingegebenen Freitext enthalten.

Da auch Suchmaschinen wie Google auf einer Volltextsuche basieren, ist diese Art der Suchfunktion, für die Benutzer ein gewohnter Komfort. Eine Volltextsuche wird dabei nicht nur in den Suchmaschinen oder direkt in auf den Websites angeboten, sondern auch für die Navigation und Suche in Software-Produkten. Ein Beispiel hierfür ist die Wiki-Software „Atlassian Confluence“, welche standardmäßig eine Suchfunktion zur Verfügung stellt. Für die Wissensaufnahme und Weitergabe der Firma Enisco wird diese Wiki-Software verwendet.

In Abbildung 2.4 ist beispielhaft die Trefferauflistung des Suchbegriffes „Studenten“ abgebildet. Hierbei dursucht die Suchfunktion den Datenbestand der Confluence-Anwendung der Firma Enisco und gibt als Ergebnis alle Seiten zurück, welche mit dem Suchbegriff übereinstimmen. In der gezeigten Ansicht erscheinen dabei alle Seiten, welche einen Treffer im Titel aufweisen. Durch die Schaltfläche „Suche nach **Studenten**“, welche in der Ansicht unten angezeigt wird, werden auch Confluence-Seiten angezeigt, in welchen der Suchbegriff auch im eigentlichen Inhalt der Seite gefunden wird.

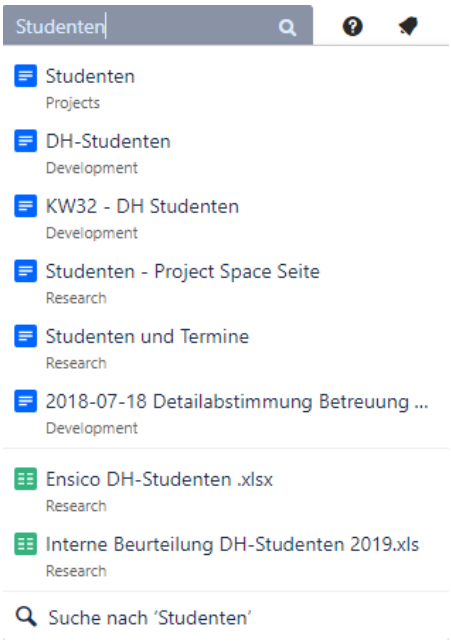

Abb. 2.4: Ergebnisse der Suchanfrage „Studenten“ - entnommen aus [Eni21a][1]

Grundlage für die technische Umsetzung einer Volltextsuche ist die Erstellung eines Suchindexes. Ein Suchindex für die Volltextsuche wird hierbei als invertierter Index umgesetzt [Seb17]. Die Funktionsweise von einem invertierten Index ist vergleichbar mit dem Stichwortverzeichnis in Büchern. Bei der Suche nach einem bestimmten Wort oder Begriff in Büchern, ist eine Suche durch das gesamte Buch umständlich und mit einem hohen Zeitaufwand verbunden. Durch die Einführung eines Stichwortverzeichnis,

[1] Quelle aus dem Intranet (nicht öffentlich zugänglich) von Enisco

wie in Abbildung 2.5 abgebildet, werden die Begriffe aus dem Buch den einzelnen Seiten zugewiesen. Eine Suche im Stichwortverzeichnis ist aufgrund der Vorsortierung schneller und man kann direkt an die entsprechende Stelle im Buch springen.

A

AMG 35
Arbeitskleidung 53
Arbeitsschuhe 56
Arbeitsschutzgesetz 42
Arbeitsstättenverordnung . . . 44
ArbSchG 42
ArbStättV 44
Arzneimittelgesetz 35
Aufenthaltsräume 61

B

Bakterien 23
Berufsgenossenschaftliche
Vorschriften 47
Berufskleidung 52
BioStoffV 44
Biostoffverordnung 44

Abb. 2.5: Beispiel für ein Stichwortverzeichnis [wek14]

Bei der Eingabe von Freitexten durch menschliche Benutzer ist das Aufkommen von Fehleingaben nicht auszuschließen. Aus diesem Grund müssen sowohl beim Aufbau des invertierten Indexes, als auch beim Bearbeiten der Suchanfrage gewisse Transformationsschritte berücksichtigt werden [Seb17]. Die Transformationsschritte sorgen für eine Fehlertoleranz innerhalb der Suchfunktion und verbessern die User Experience bezüglich der Benutzung der Suchfunktion.

Die Transformationsschritte werden benötigt, um die Vielzahl der unterschiedlichen Suchanfragen korrekt zu beantworten [Seb17]. Hierbei werden je nach Anforderungen verschiedene Tokenizer und Analyzer verwendet. In Abbildung 2.6 ist eine beispielhafte Abfolge von Transformationsschritten aufgezeigt. Ziel ist es die variantenreiche Suchphrase in ein einheitliches Format zu überführen. Dadurch kann eine ausreichende Fehlertoleranz geschaffen werden. Neben der Suchphrase müssen die Transformationsschritte auch bei der Erstellung des Indexes berücksichtigt werden [Seb17].

Die in Abbildung 2.6 aufgezeigten Transformationsschritte werden nachfolgend näher erläutert.

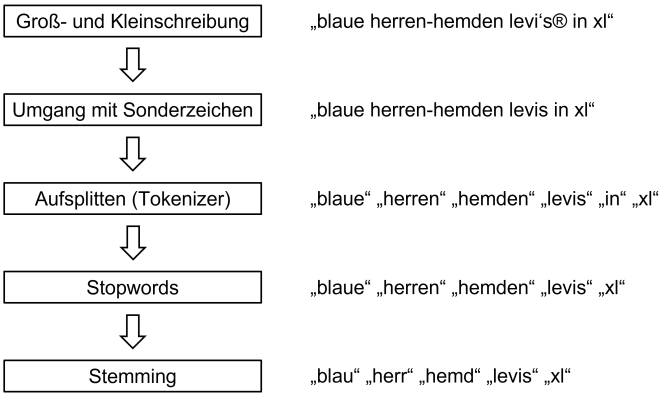

Abb. 2.6: Beispielhafter Ablauf der Transformationsschritte - in Anlehnung an [Seb17]

Tokenizer

Hierbei wird die komplette Suchphrase in verschiedene Suchbegriffe (auch Tokens genannt) aufgeteilt. So wird beispielsweise aus der Suchphrase „Hose Blau“ die Tokens „Hose“ und „Blau“. Neben der Trennung der Tokens anhand von Leerzeichen, ist auch eine Trennung anhand von bestimmten Zeichen möglich. Solche Zeichen können zum Beispiel das Subtraktionszeichen „-“ oder Semikolonzeichen „;“ sein. Die Aufteilung in Tokens ist hierbei die Vorbereitung für die Verwendung von einigen Analyzern.

Analyzer

Die Analyzer agieren bei der Transformation der Suchanfrage als Filter und ersetzen, löschen oder verändern einzelne Tokens. Darunter zählen unteranderem Filter bezüglich der Groß- und Kleinschreibung und eventuellen Sonderzeichen. Durch dieses Vorgehen wird eine gewisse Fehlertoleranz gegenüber den Eingaben der Benutzer gewährt. Zu den Sonderzeichen gehöhren zum Beispiel Symbole, wie die Kennzeichnung für eingetragene Handelsmarken „®“, oder Trennzeichen für verschiedene Begriffe. So sollen die Terme „USA“ und „U.S.A“ auf einen gemeinsamen Term gemappt.

Eine weitere Möglichkeit die Suchphrase zu filtern ist das Entfernen von Stopwords. Dies sind häufig auftretende Begriffe, welche eine geringe semantische Relevanz für die Suchphrase besitzen [Seb17]. Die Stopwords werden als Liste angelegt, welche überwiegend aus Füllwörtern wie zum Beispiel „mit“, „für“ und „von“ besteht. Lediglich bei der Verwendung einer semantischen Suche, werden die Stopwords benötigt, um die Bedeutung zu erkennen [Seb17]. Die Funktionsweise einer semantischen Suche wird in Unterabschnitt 2.3.4 erläutert.

Um die Fehlertoleranz zu erhöhen, muss auch die morphologische Varianz der einzelnen Begriffe berücksichtigt werden. So können Begriffe in unterschiedlichsten Zeitformen und

Numeri vom Benutzer eingegeben werden. Um die Varianz der menschlichen Sprache zu beseitigen, müssen die Begriffe durch Normalisierung vereinheitlicht werden. Ein Verfahren für die Normalisierung ist das „Stemming“ (zu deutsch Stammformreduktion). Beim Stemming werden die unterschiedlichen morphologischen Varianten eines Begriffes auf einen gemeinsamen Wortstamm zurückgeführt. Als ein Arbeitsschritt aus dem Natural Language Processing (NLP), wird Stemming auch in Suchmaschinen verwendet [Ste20]. Hierbei muss dieses Verfahren sowohl bei der Behandlung der Suchphrase, als auch bei der Erstellung des Indexes berücksichtigt werden [Seb17]. Folgend sind ein paar Beispiele für Stemming aufgelistet [Seb17]:

- rot, rotes, roter, rote → rot
- schuh, schuhe → schuh
- töpfe, topf → topf

Ein beispielhafter Ablauf der Transformation sowohl beim Zeitpunkt der Abfrage, als auch zum Zeitpunkt der Erzeugung des Indexes ist in Abbildung 2.7 abgebildet. Hierbei wird deutlich, dass bei der technischen Umsetzung eines invertierten Indexes die Wahl der Analyzer sowohl für die Bearbeitung der Suchphrasen, als auch für die Erstellung des Indexes von Bedeutung ist.

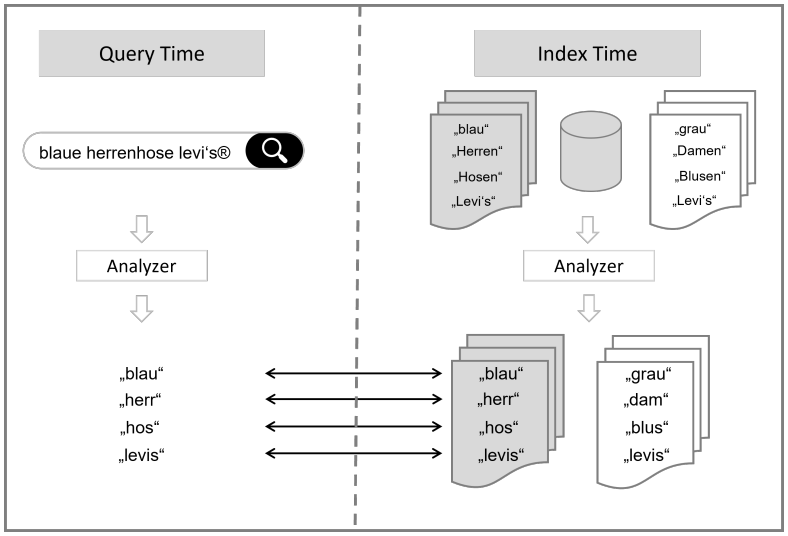

Abb. 2.7: Analyzer bei Indexierung und Abfrage - in Anlehnung an [Seb17]

2.3.3 Facettierte Suche

Bei der Facetten-Suche wird die Menge der Suchergebnisse durch das Setzen von Filtern immer weiter eingeschränkt. Die Objekte und Produkte eines Informationssystems werden dabei mit Metadaten angereichert, welche die jeweiligen Facetten definieren. Der Begriff Facette beschreibt die verschiedenen Teilaspekte eines Objekts oder Produkts.

So kann beispielhaft ein „Hemd“ unteranderem mit den Facetten „Größe“, „Farbe“ oder „Muster“ versehen werden.

Eine Mindestanforderung nach Sperling [Mar18] ist, dass die Filter sich dynamisch aus den bereits gefundenen Suchergebnistreffern bilden. So wechseln dynamisch die Filtermöglichkeiten je nach Suchergebnis und sinnfreie Filtermöglichkeiten beziehungsweise Filtermöglichkeiten ohne Suchergebnisse werden vermieden. In der Praxis ist ein solch dynamischer Wechsel der Filtermöglichkeiten in Online-Shops zu finden. Ein Benutzer ist zum Beispiel bei einem USB-Stick daran interessiert mithilfe eines Filters die maximale Datenmenge zu begrenzen und bei der Suche nach Waschmaschinen ist die Angabe des maximalen Fassungsvermögen eine aussagekräftige Filterung.

Wie in Abbildung 2.8 zu erkennen ist, kann man diese Filter mit unterschiedlichsten Bedienelementen visualisieren. Dabei ist zu unterscheiden, ob die Werte der Facetten explizit genannt werden können oder sich in einem Bereich finde. Wie von Schubert Motors [Sch21] in Abbildung 2.8 umgesetzt, wurde zum Beispiel für die Preis-Facette ein Slider und für die Auswahl der Marke ein Drop-Down-Bedienelement ausgewählt.

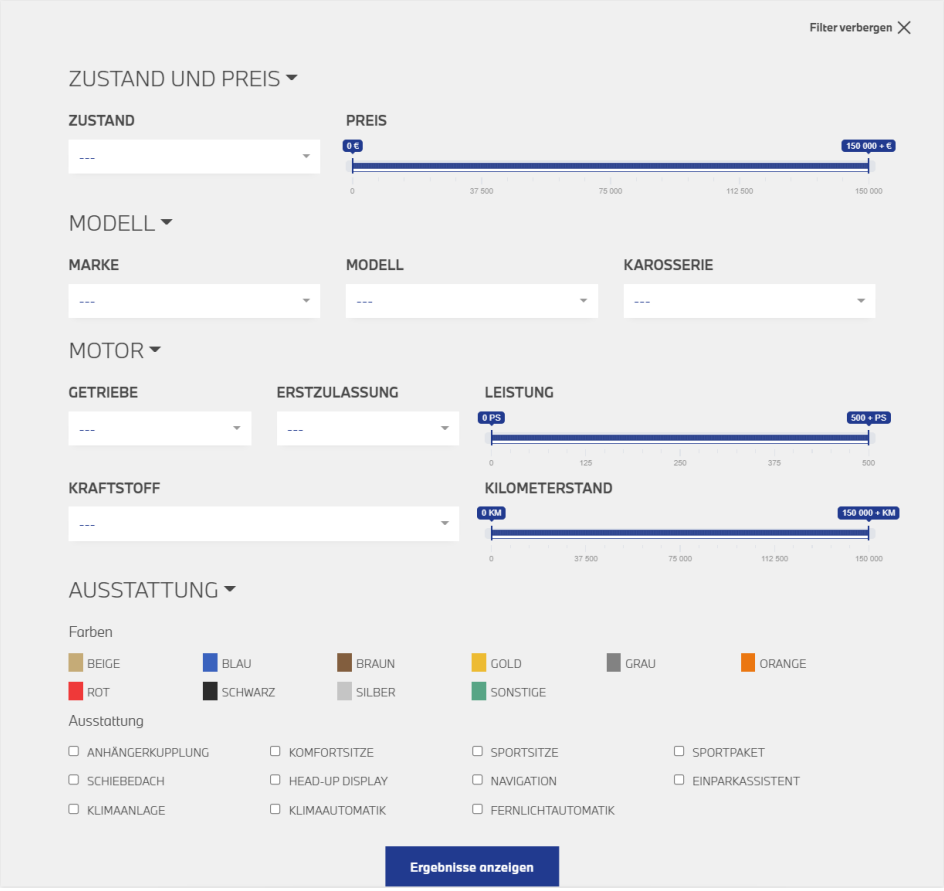

Abb. 2.8: Facettierte Suche bei Schubert Motors [Sch21]

Bei der praktischen Umsetzung einer facettierten Suche müssen die logischen Verknüpfungen zwischen den einzelnen Filter konfiguriert werden [Mar18]. Aus den Ergebnissen

der unterschiedlichen Filter müssen Vereinigungs- oder Schnittmengen gebildet werden, welche wiederum sortiert, durch eine Relevanzbestimmung, dem Benutzer präsentiert werden. Zum Einsatz kommen hierfür neben „UND“- auch „ODER“- Verknüpfungen. Die Wahl der richtigen Verknüpfung ist vom jeweiligen Kontext abhängig. So ist die Intention eines Benutzers bei der Benutzung der Facette „Farbe“, dass die einzelnen Ergebnisse mit einer ODER-Verknüpfungen miteinander kombiniert werden. Ein Benutzer erhält somit beispielsweise als Ergebnis alle Produkte der Farbe rot **UND** blau. Verwendet man jedoch mehrere Facetten bei der Suche, wie zum Beispiel die „Farbe“ und „Größe“, ist die Schnittmenge aus den Ergebnissen von Vorteil. Hier ist eine UND-Verknüpfungen notwendig.

2.3.4 Semantische Suche

Während die Varianten „Volltextsuche“ und „Facettierte Suche“ lediglich auf einer syntaktischen und statistischen Auswertung von einzelnen Suchbegriffen basieren, greift eine semantische Suche zusätzlich noch auf ein semantisches Modell zurück [Hop20, S. 4f]. Solch ein, auch als „Wissensmodell“ bezeichnetes, semantisches Modell, beschreibt die begrifflichen Zusammenhänge und Beziehungen. Dadurch ist es möglich neben syntaktisch ähnlichen, auch inhaltlich verwandte Treffer zu finden [Hop20, S. 3]. Nach Hoppe [Hop20, S. 3] kann demnach eine semantische Suche, im Sprachgebrauch der Künstlichen Intelligenz, als wissensbasiert bezeichnet werden.

Ziel ist es die Suchfunktion weiter der menschlichen Sprache anzunähern. So liegt es, laut Tamblé [Mel12], in der Natur der menschlichen Sprache, dass mit verschiedenen Wörtern inhaltlich gleiche oder ähnliche Bedeutungen gemeint sind. So kann eine Landfläche in der deutschen Sprache auch mit den Begriffen „Bereich“, „Gebiet“ oder „Areal“ beschrieben werden. Neben Synonymen werden Begriffe auch durch Assoziationen miteinander verbunden. So wird der Begriff „Rot“ mit dem Begriff „Liebe“ assoziiert.

Für die Umsetzung einer semantischen Suche bedarf es einer Wissensdatenbank, in welcher Entitäten und deren Beziehungen untereinander abgespeichert werden. Mit dem Knowledge Graphen betreibt Google seit 2012 solch eine Wissensdatenbank, um Funktionen einer semantischen Suche bereitzustellen [Ola21]. Der Knowledge Graph wird dabei in drei Ebenen unterteilt: [Ola21]

Entitäten-Katalog:

Die unterste Ebene ist eine reine Auflistung der verschiedenen Entitäten, welche über die Zeit hinweg identifiziert wurden. Die Art der Entitäten ist abhängig vom Kontext der Anwendung, in welcher eine Suchfunktion integriert ist. Allgem,ein handelt es sich um Objekte und Konzepte, welche eindeutig identifiziert werden können. Darunter zählen zum Beispiel Persönlichkeiten, Orte, Organisationen oder auch Farben und Gefühle.

Knowledge Repository:

Die unterschiedlichen Entitäten aus dem Entitäten-Katalog werden im Knowledge Repository mit Informationen beziehungsweise Attributen erweitert. Neben dem Ergänzen der Entitäten um Beschreibungen, werden diese auch in semantische Klassen und Gruppen eingeteilt. Die Gruppierungen werden auch Entitätstypen genannt.

Knowledge Graph:

Im eigentlichen Graphen werden die Beziehungen zwischen den unterschiedlichen Entitäten hergestellt.

In Abbildung 2.9 ist das Ergebnis einer semantische Suchanfrage von Google abgebildet. Gesucht wurde mit der Suchphrase „**small green guy with lightsaber as child**“. Als Ergebnis erhält der Benutzer eine sofortige Übersicht über das wahrscheinlichste Ergebnis. In diesem Beispiel wurden die einzelnen Begriffe aus der Suchphrase mit dem Knowledge-Graphen von Google verglichen. Dabei erwiesen die Beziehungen der Begriffe untereinander, dass der fiktive Charakter „Grogu“ am wahrscheinlichsten gesucht wird.

Während bei einer reinen syntaktischen Suchfunktion die einzelnen Begriffe durch Transformationsschritte vereinheitlicht und anschließend mit einem invertierten Index verglichen werden, kann die semantische Suche nun auch mit Synonymen und Assoziationen umgehen. So würde die Suchmaschine Google das gleiche Ergebnis, wie in Abbildung 2.9, liefern, wenn statt dem Begriff „small“ das Synonym „little“ verwendet wird.

Abb. 2.9: Ergebnis einer semantischen Suche in Google

Um genauere und personalisierte Suchergebnisse zu erhalten, verwendet Google bei der semantischen Suche zusätzlich noch die persönlichen Daten und den Suchverlauf der Benutzer [Mic20]. So ist ein Programmierer mit der Suchphrase „Python“ eher daran

interessiert Suchergebnisse über die selbige Programmiersprache zu erhalten, anstatt über das Tier.

2.3.5 Relevanzbestimmung

Unabhängig, ob eine Volltextsuche, facettierte Suche, semantische Suche oder eine Kombination zum Einsatz kommt, müssen die Ergebnisse der Suchanfrage in einer geeigneten Reihenfolge angezeigt werden. Es gilt die einzelnen Treffer der Suche mit einer Relevanz zu gewichten und dem Benutzer die Treffer mit der höchsten Relevanz als erstes anzuzeigen.

Eine Möglichkeit der praktischen Umsetzung einer Relevanzbestimmung ist die Durchführung einer Termgewichtung mit dem „TF-IDF“-Modell. „Term Frequency - Inverse Document Frequency (TF-IDF)“ ist dabei ein statistisches Maß, welches angibt, wie relevant ein Begriff für ein Dokument in einer Sammlung von Dokumenten ist [Bru19]. Durch die Multiplikation der beiden Metriken „Term Frequency (TF)“ und „Inverse Document Frequency (IDF)“ kann eine Sortierung der Suchergebnisse nach Relevanz erfolgen.

Term Frequency - TF

Eine der Metrik bezieht sich auf die Häufigkeit eines Begriffes in einem Dokument. Um diese Häufigkeit zu bestimmen, gibt es verschiedene Möglichkeiten. Das einfachste Vorgehen ist die reine Zählung des Vorkommens des Begriffes in einem Dokument [Bru19]. Weitergehend kann man die Häufigkeit auch nach der Länge des Dokumentes oder nach der Häufigkeit des häufigsten Wortes abhängig machen [Bru19].

Unter der Einbeziehung der Gesamtzahl aller Terme eines Dokumentes, wird mit folgender Formel [Jen20] die Begriffshäufigkeit eines Begriffes berechnet:

$$TF_{(i)} = \frac{\log_2(Freq(i,j)+1)}{\log_2(L)}$$

- i = Term
- j = Dokument
- L = Gesamtzahl aller Terme im Dokument j
- Freq(i,j) = Häufigkeit des Terms i im Dokument j

Inverse Document Frequency - IDF

Multipliziert wird die erste Metrik mit der inversen Dokumentenhäufigkeit. Darunter versteht man die Häufigkeit des Vorkommens eines Begriffes über die gesamte Dokumentenmenge hinweg. Einem Begriff wird hierbei ein größerer IDF-Wert zugeordnet, wenn

der Begriff seltener vorkommt. Stopwords würden hierbei einen niedrigen IDF-Wert erhalten, da diese Begriffe in vielen Dokumenten vorkommen.

Für die Berechnung der invertierten Dokumentenhäufigkeit wird folgende Formel [Jen20] verwendet:

$$IDF_{\mathrm{t}} = \log(\frac{N_{\mathrm{D}}}{f_{\mathrm{t}}})$$

- t = Term
- N_D = Gesamtzahl der Dokumente in der Dokumentensammlung D
- f_t = Anzahl aller Dokumente, die t enthalten

Für die Bestimmung der Relevanz eines Begriffes in einem Dokument, wird die Multiplikation beider Metriken benötigt. Bei lediglich der Bestimmung der Begriffshäufigkeit, würden Stopwords eine große Relevanz bekommen. Erst durch die Einbeziehung der inversen Dokumentenhäufigkeit werden solche Begriffe, wie Stopwords mit einer niedrigeren Relevanz versehen.

Beispiel

Bei der Anwendung des TF-IDF - Wertes für die Relevanzbestimmung der Suchtreffer, werden die IDF-Werte der einzelnen Begriffe aus der Suchphrase miteinander verglichen. Sucht der Benutzer zum Beispiel nach der Suchphrase „Website Analytics“, wird vermutlich dem Begriff „Website“ ein niedriger IDF-Wert zugeordnet, da dieser Begriff in vielen Dokumenten vorkommt.

Weitergehend werden dann die jeweiligen TF-Werte bestimmt. Hierbei wird jeder Begriff aus der Suchphrase mit jedem Dokument aus der Trefferliste in die TF-Formel eingegeben. So wird ein Dokument, welches den Begriff „Analytics“ häufig enthält einen hohen TF-Wert erhalten.

Durch die Multiplikation beider Metriken pro Dokument kann eine Sortierung nach Relevanz durchgeführt werden.

Kapitel 3

3 Suchumfang in MCC

Text

3.1 Umfang der Volltextsuche

Text

3.2 Umfang der facettierten Suche

Text

3.2.1 SCADA

Text

3.2.2 PCS

Text

3.3 Umfang einer facettierten Volltextsuche

Text

3.4 Umfang einer semantischen Suche

Text

Kapitel 4

4 Konzeption

Inhalt

4.1 Konzeptionskriterien

Inhalt

4.2 Technische Umsetzung einer Volltextsuche

Inhalt

4.2.1 Search Engine

Inhalt

4.2.2 DBMS-interne Volltextsuche

Inhalt

4.3 Auswahl einer Search Engine

Inhalt

4.3.1 Apache Lucene

Inhalt

4.3.2 Apache Solr

Inhalt

4.3.3 Elasticsearch

Inhalt

4.3.4 Vergleich

Inhalt

4.4 Datenaktualisierung zwischen Microservices und Search Engine

Inhalt

4.4.1 Clientseitige Aktualisierung

Inhalt

4.4.2 Pull-or-Push Aktualisierung

Inhalt

4.4.3 Change-Data-Capture Aktualisierung

Inhalt

4.4.4 Vergleich

Inhalt

4.5 Gesamtkonzept

Inhalt

Kapitel 5

5 Prototypische Umsetzung

Inhalt

5.1 Client

Inhalt

5.2 Search Service

Inhalt

5.3 Anbindung an Apache Kafka

Inhalt

Kapitel 6

6 Fazit und Ausblick

Inhalt

Literaturverzeichnis

[Aar] Aaron Schrettenbrunner ; eCommerce Leitfaden (Hrsg.): *Wer suchet, der findet – auch im Online-Handel? Status quo bei Shop-Suche und Produktvorschlägen im Online-Shop*. https://www.ecommerce-leitfaden.de/studien/item/wer-suchet-der-findet-auch-im-online-handel

[Apa21] Apache Software Foundation ; Apache Software Foundation (Hrsg.): *Kafka 2.8 Documentation*. https://kafka.apache.org/documentation/. 2021

[BCK13] Bass, Len ; Clements, Paul ; Kazman, Rick: *Software architecture in practice*. Third edition, fifth printing. Upper Saddle River, NJ : Addison-Wesley, 2013 (SEI series in software engineering). – ISBN 9780132942775

[Bru19] Bruno Stecanella ; monkeylearn.com (Hrsg.): *What Is TF-IDF?* https://monkeylearn.com/blog/what-is-tf-idf/. 2019

[Eni21a] Enisco by Forcam GmbH ; Enisco by Forcam GmbH (Hrsg.): *Confluence*. https://ensrd003.enisco.corp/. 2021

[Eni21b] Enisco by Forcam GmbH ; Enisco by Forcam GmbH (Hrsg.): *Homepage: Das Produktionsleitsystem E-MES*. https://enisco.com/. 2021

[F. 18] F. Tenzer ; Seagate (Hrsg.): *Prognose zum Volumen der jährlich generierten digitalen Datenmenge weltweit in den Jahren 2018 und 2025*. https://de.statista.com/statistik/daten/studie/267974/umfrage/prognose-zum-weltweit-generierten-datenvolumen/. 2018

[Flo17] Florian Raschbichler ; Informatik Aktuell (Hrsg.): *MQTT - Leitfaden zum Protokoll für das Internet der Dinge*. https://www.informatik-aktuell.de/betrieb/netzwerke/mqtt-leitfaden-zum-protokoll-fuer-das-internet-der-dinge.html. 2017

[Gar18] Gary Calcott ; NetMediaEurope Deutschland GmbH (Hrsg.): *Microservices vs. Monolithische Architekturen: ein Leitfaden*. https://www.silicon.de/41666855/

microservices-vs-monolithische-architekturen-ein-leitfaden. 2018

[Guy21] Guy Fawkes ; vpnmentor.com (Hrsg.): *Internet-Trends 2021: Statistiken & Fakten USA + weltweit*. https://de.vpnmentor.com/blog/internet-trends-statistiken-fakten-aus-den-usa-und-weltweit/. 2021

[Hop20] Hoppe, Thomas: *Semantische Suche*. Wiesbaden : Springer Fachmedien Wiesbaden, 2020. – ISBN 978–3–658–30426–3

[ION21] IONOS SE ; IONOS SE (Hrsg.): *Microservice-Architectures: Mehr als die Summe ihrer Teile?* https://www.ionos.de/digitalguide/websites/web-entwicklung/microservice-architecture-so-funktionieren-microservices/. 2021

[Jen20] Jens Fröhlich ; indexlift.com (Hrsg.): *TF-IDF Termgewichtung*. https://www.indexlift.com/de/blog/tf-idf-termgewichtung. 2020

[Mar18] Martin, Sperling ; Martin Sperling (Hrsg.): *Die Suchfunktion als Umsatzbringer – "Must-Haves" für Ihren Onlineshop*. https://ecommerce-berater.eu/die-suchfunktion-wie-sie-die-beste-loesung-fuer-ihren-onlineshop-finden/. 2018

[Mel12] Melanie Tamblé ; marketing-BÖRSE GmbH (Hrsg.): *Was ist eigentlich: Semantische Suche? Die semantische Suche ist ein neuer Algorithmus der Suchmaschinen*. https://www.marketing-boerse.de/fachartikel/details/1216-was-ist-eigentlich-semantische-suche/35736. 2012

[Mic19] Michael Schwab ; Host Europe GmbH (Hrsg.): *Microservices — Grundlagen und Technologien von verteilter Architektur*. https://www.hosteurope.de/blog/microservices-grundlagen-und-technologien-von-verteilter-architektur/. 2019

[Mic20] Michal Pecánek ; Ahrefs Pte. Ltd. (Hrsg.): *Was ist die semantische Suche? Wie sie SEO beeinflusst*. https://ahrefs.com/blog/de/semantische-suche/. 2020

[Ola21] Olaf Kopp ; Aufgesang GmbH (Hrsg.): *Semantische Suche: Suchmaschinen, Definition, Funktion, Geschichte*. https://www.sem-deutschland.de/seo-glossar/semantische-suche/#FAQ_zu_semantischen_Suchmaschinen. 2021

[Ope21] Open Industry 4.0 Alliance ; Open Industry 4.0 Alliance (Hrsg.): *Über Uns*. https://openindustry4.com/de/About-Us.html. 2021

[Pro12a] Prof. Dr. Andreas Fink ; Universität Potsdam (Hrsg.): *Monolithisches IT-System*. https://www.enzyklopaedie-der-wirtschaftsinformatik.de/lexikon/is-management/Systementwicklung/Softwarearchitektur/Architekturparadigmen/Monolithisches-IT-System. 2012

[Pro12b] Prof. Dr. Andreas Fink ; Universität Potsdam (Hrsg.): *Verteiltes IT-System*. https://www.enzyklopaedie-der-wirtschaftsinformatik.de/wi-enzyklopaedie/lexikon/is-management/Systementwicklung/Softwarearchitektur/Architekturparadigmen/Verteiltes-IT-System/index.html. 2012

[Sch21] Schubert Motors GmbH ; Schubert Motors GmbH (Hrsg.): *Unsere Angebote*. https://www.schubert-motors.de/fahrzeuge. 2021

[Seb17] Sebastian Russ ; Tudock GmbH (Hrsg.): *Grundwissen Ecommerce Onsite Search: #1: Funktionsweise*. https://www.tudock.de/archiv/grundwissen-ecommerce-onsite-search-1-funktionsweise/. 2017

[SM19] Stephan Roth ; Martina Hafner ; Vogel Communications Group GmbH & Co. KG (Hrsg.): *Anti-Patterns: Wiederkehrende Entwicklerfehler erkennen und vermeiden*. https://www.embedded-software-engineering.de/anti-patterns-wiederkehrende-entwicklerfehler-erkennen-und-vermeiden-a-67 2019

[Ste20] Stefan Luber ; Vogel IT-Medien GmbH (Hrsg.): *Was ist Stemming?* https://www.bigdata-insider.de/was-ist-stemming-a-980852/. 2020

[TAM+20] Tighilt, Rafik ; Abdellatif, Manel ; Moha, Naouel ; Mili, Hafedh ; Boussaidi, Ghizlane E. ; Privat, Jean ; Guéhéneuc, Yann-Gaël: On the Study of Microservices Antipatterns. In: *Proceedings of the European Conference on Pattern Languages of Programs 2020*. New York, NY, USA : ACM, 072020. – ISBN 9781450377690, S. 1–13

[Vog09] Vogel, Oliver: *Software-Architektur: Grundlagen - Konzepte - Praxis*. 2. Aufl. Heidelberg : Spektrum Akad. Verl., 2009 http://deposit.d-nb.de/cgi-bin/dokserv?id=3123606&prov=M&dok_var=1&dok_ext=htm. – ISBN 978–3–8274–1933–0

[wek14] WEKA.DE ; YUMPU.COM (Hrsg.): *Stichwortverzeichnis*. https://www.yumpu.com/de/document/view/21367951/stichwortverzeichnis. 2014