BACHELORARBEIT

zum Thema

„Optimierung und Evaluierung einer Erweiterung zur Integration von Click-Through-Daten im Apache SOLR Scoring-Algorithmus“

vorgelegt an der

[redacted]

von: [redacted]

Matrikelnummer: [redacted]

Studiengang: Praktische Informatik

Praxispartner: dotSource GmbH
Digital Success right from the Start
Goethestraße 1
07743 Jena

Gutachter der [redacted]**:** [redacted]

Gutachter des Praxispartners (Betreuer i.S.v. § 20 (1) DHGEPrüfO): [redacted]

Head Office Jena
Goethestraße 1
07743 Jena
FON +49 (0) 3641 797 9000
FAX +49 (0) 3641 797 9099
E-MAIL info@dotSource.de

Office Berlin
Pappelallee 78/79
10437 Berlin
FON +49 (0) 30 220 122 360

Office Leipzig
Hainstraße 1–3
04109 Leipzig
FON +49 (0) 341 9919 1000

Bankverbindung
Deutsche Bank Jena
IBAN DE63 8207 0000 0633 7778 00
BIC DEUTDE8EXXX

Commerzbank Jena
IBAN DE31 8204 0000 0259 9934 00
BIC COBADEFF821

Sparkasse Jena
IBAN DE35 8305 3030 0018 0037 61
BIC HELADEF1JEN

Geschäftsführer
Christian Otto Grötsch
Christian Malik
Frank Ertel

Gerichtsstand
Amtsgericht Jena
HRB 210634
USt-IdNr.: DE246243309
Steuer-Nr.: 162/107/03164

Sperrvermerk

Die vorliegende Arbeit beinhaltet interne und vertrauliche Informationen der Firma dotSource GmbH. Die Weitergabe des Inhaltes der Arbeit und eventuell beiliegender Zeichnungen und Daten im Gesamten oder in Teilen ist grundsätzlich untersagt. Es dürfen keinerlei Kopien oder Abschriften - auch in digitaler Form - gefertigt werden. Ausnahmen bedürfen der schriftlichen Genehmigung der Firma dotSource GmbH.

Abstract

Ziel dieser Arbeit ist die Umsetzung einer Software zur Erweiterung des SOLR um einen Click-Through-Algorithmus. Damit soll die Qualität der Suchergebnisse in einem Webshop verbessert werden, indem häufig geklickte Treffer weiter oben gezeigt werden. Dies geschieht auf Basis eines bestehenden Prototyps. Anschließend erfolgt die Evaluierung der Entwicklung. Dadurch soll festgestellt werden, ob ein erfolgreiches Deployment auf einer Produktivumgebung gewährleistet werden kann.
Bei der Analyse des Prototyps wurden verschiedene Verbesserungspotenziale festgestellt und anschließend umgesetzt. Diese verfolgen den Ansatz, sowohl die Performance als auch die Robustheit zu optimieren. Danach erfolgte der Test der Relevanz der Suchergebnisse. Durch eine Umfrage unter Personen, die an der betrachteten Online-Präsenz beteiligt sind, ist eine Verbesserung der Suchergebnisse festgestellt worden. Beim Test des Einflusses auf die Ressourcen ist eine Steigerung der Antwortzeit um durchschnittlich 28 Millisekunden und ein Speicherbedarf von 228 Megabytes gemessen worden. Damit wird auch auf dem Produktivsystem ein positives Ergebnis erwartet.

The aim of this work is to implement a software to extend the SOLR with a click-through algorithm. This is intended to improve the quality of search results in a webshop by showing frequently clicked hits further ahead. This is done on the basis of an existing prototype. It is followed by an evaluation of the development. The aim is to determine whether a successful deployment on a production environment can be guaranteed. During the analysis of the prototype, various potential improvements were identified and subsequently implemented. These follow the approach of optimizing both performance and robustness. Then the relevance of the search results was tested. In a survey of people involved in the online presence an improvement of the quality of the search results has been found. Testing the impact on resources an increase in response time of an average of 28 milliseconds and a storage requirement of 228 megabytes was measured. Finally a positive result is expected on the production system.

I Inhaltsverzeichnis

II Tabellenverzeichnis

III Abbildungsverzeichnis

IV Abkürzungsverzeichnis

CTA	Click-Through-Algorithmus
CTD	Click-Through-Daten
CTR	Click-Through-Rate
DBMS	Datenbankmanagementsystem
DHGE	Duale Hochschule Gera-Eisenach
ORM	Object-Relational Mapping
URL	Uniform Resource Locator

1 Einleitung

„There is no alternative to digital transformation. Visionary companies will carve out new strategic options for themselves — those that don't adapt, will fail.“ [Bez22]

Jeff Bezos betont mit dieser Aussage die Notwendigkeit für Unternehmen, ihre Geschäftsprozesse auf digitale Strategien auszuweiten. Dass dieses Zitat Relevanz besitzt, zeigt der steigende Umsatz im Online-Handel [Sta22]. Viele Firmen befassen sich mit der Bedeutung des E-Commerce und setzen auf Webshops. Im Vergleich zum stationären Handel bringen diese Vorteile, wie Einsparungen von Personalkosten, sowie eine höhere Reichweite und Verfügbarkeit für den Kunden [Dau22].

Bei der Umsetzung dieser Online-Präsenzen sind verschiedene Komponenten zu beachten. Dazu gehören ein Warenkorb, Produktdetailseiten oder die Produktsuche. Letztere hat mit einer Beteiligung an 80 Prozent der Online-Käufe [Mar18] einen entscheidenden Einfluss auf den Erfolg der Webshops. Sie dient der gezielten Navigation des Kunden zu einem Ergebnis. Das Ziel ist es, möglichst passende Elemente zu einer Suchanfrage zu liefern. Dies versuchen viele Shopbetreiber durch intelligente Suchfunktionalitäten zu optimieren. Grund dafür ist, dass naturgemäß weniger Einkäufe getätigt werden, wenn Ergebnisse niedrigerer Relevanz präsentiert werden. Somit könnte ein mögliches Interesse für einen Kauf auf der Online-Präsenz verloren werden. Eine Umfrage zeigt, dass Shopbetreiber eine Umsatzsteigerung von 15 bis 30 Prozent durch verbesserte Produktsuchen erreichen konnten [Mar18].

Um die Qualität der Suchergebnisse zu verbessern, gibt es verschiedene Möglichkeiten. Dazu gehören die Toleranz bei Tippfehlern im Suchbegriff oder die Verwendung von verschiedenen Ranking-Strategien, wie das Bevorzugen von Produkten mit hohem Gewinn oder das Click-Through-Scoring. Das Click-Through-Ranking bezeichnet die Fähigkeit eines Suchalgorithmus, in der Vergangenheit häufig geklickte Treffer präsenter anzuzeigen. Das wird zum Beispiel mittels Anzeigen des Produktes auf einer vorderen Position im Suchergebnis erreicht. Die Grundlage dafür ist, dass Kunden ein bestimmtes Suchergebnis aus einer Auflistung auswählen. Das indiziert eine Relevanz dieses Treffers für den eingegebenen Suchbegriff.

Unternehmen haben die Möglichkeit, Drittanbietersoftware für ihre Produktsuche zu verwenden. Vor allem bei Lösungen, die frei zur Verfügung stehen, ist es möglich, dass diese kein Click-Through-Scoring beinhalten. Ein Beispiel dafür ist die Suchmaschine SOLR, die von der E-Commerce Plattform SAP Commerce verwendet wird. Es ist möglich, die bestehende Implementierung um diese Funktionalität zu erweitern.

Ziel dieser Arbeit ist die Umsetzung einer Software zur Erweiterung des SOLR um einen Click-Through-Scoring Algorithmus. Dies erfolgt auf Basis eines bestehenden Prototyps. Weiterhin soll diese Software auf die Einhaltung funktionaler und nichtfunktionaler Anforderungen geprüft werden, um einen erfolgreichen Einsatz in einem Produktivsystem zu gewährleisten.

Um dies zu erreichen, wird zuerst der bereits existierende Prototyp im Zusammenhang mit dem Projekt, an dem er angebunden werden soll, vorgestellt. Dabei wird auch auf notwendige Verbesserungen eingegangen, deren Umsetzung im darauffolgenden Kapitel beschrieben wird. Weiterhin wird sich mit dem Testvorgehen befasst. Es folgt die Evaluierung der Entwicklung. Dabei wird sowohl auf die Performance als auch auf Qualität der Suchergebnisse eingegangen. Zuletzt folgen eine Zusammenfassung und ein Fazit der Arbeit.

Motivation dieser Arbeit ist die Verbesserung der Produktsuche. Das könnte zur Erhöhung der Kundenzufriedenheit und Steigerung des Umsatzes führen. Aus Sicht der dotSource GmbH ergibt sich der Vorteil, dass eine wiederverwendbare Erweiterung umgesetzt wird. Diese [gekürzt, Anm. d. Red.] könnte auch auf anderen SAP Commerce Systemen implementiert werden.

2 Vorüberlegung

In diesem Kapitel erfolgt die Beschreibung des aktuellen Zustands der Entwicklung. Dieser ist die Grundlage für das weitere Vorgehen. Hierfür wird zunächst das Konzept des Click-Through-Scorings thematisiert. Dann wird der bereits existierende Prototyp und die Produktsuche des Projekts, in dem er implementiert werden soll, vorgestellt.

2.1 Click-Through-Scoring Algorithmen in Webshops

Gemäß [Yue21] ist die Präsentation irrelevanter Suchergebnisse ein Grund für eine hohe Absprungrate in Online-Shops. Um die Kaufbereitschaft der Kunden zu erhöhen, sollte die Produktsuche an die Erwartungen der Benutzer angepasst werden. Eine Möglichkeit dafür ist die Nutzung eines Click-Through-Algorithmus (CTA).

Verwendet ein Nutzer die Suche, wird eine Liste von Alternativen präsentiert. In dieser Auflistung kann zwischen verschiedenen Elementen gewählt werden. Dabei wird nicht automatisch die erste Option ausgesucht. Die Entscheidung für ein bestimmtes Element impliziert, dass dieses der Erwartung des Nutzers, fortlaufend als Suchintention bezeichnet, entspricht. Für das gewählte Produkt wird dadurch eine Relevanz in Abhängigkeit zum Suchbegriff indiziert [Vis21]. Bei weiteren Suchanfragen mit diesem Begriff kann folgende Annahme aufgestellt werden: Das gewählte Element entspricht, mit erhöhter Wahrscheinlichkeit im Vergleich zu anderen Produkten, erneut der Suchintention. Um dem Kunden relevantere Treffer zu zeigen, sollte es häufiger oder auf einer besseren Position im Suchergebnis präsentiert werden.

Auf Basis der impliziten Rückmeldung der Nutzer versuchen Click-Through-Algorithmen, das Suchergebnis in qualitativer Hinsicht zu verbessern. Dafür kommt die sogenannte Click-Through-Rate (CTR) zum Einsatz. Sie ist ein Maß für die Relevanz eines Elements und ergibt sich aus dem Verhältnis zwischen Klicks und Impressionen [SEO22]. Letzteres meint die Anzeige im Suchergebnis, sodass das Produkt durch den Nutzer ausgewählt werden kann. Die CTR wird durch folgende Formel berechnet:

$$CTR = \frac{Klicks}{Impressionen}$$

Die Funktionsweise eines CTA wird im Folgenden an einem Beispiel erläutert. Die in Tabelle 2.1 gezeigte Auflistung stellt eine fiktive Ausgangssituation ohne Anwendung des CTA dar. Als Suchbegriff wird „Hammer groß“ gewählt.

Position in Auflistung	Score	Produkttitel	Click-Through-Rate
1	600	„Spalthammer groß“	40%
2	500	„Stiel für Hammer“	4%
3	400	„großer Gummihammer“	40%

Tabelle 2.1 - Beispiel eines Suchergebnis ohne angewendeten CTA

Es werden drei Suchergebnisse auf unterschiedlichen Positionen betrachtet, deren Reihenfolge von einem Score bestimmt wird. Um diesen zu bestimmen, ist durch den Ranking-Algorithmus auf Übereinstimmungen zwischen Titel und dem Suchbegriff „Hammer groß“ geprüft worden. Die Wertung wird am meisten durch exakte Treffer bestimmt. Das bedeutet, dass ein Wort des Suchbegriffs mit einem Wort des Produkttitels übereinstimmt. So gibt es beispielsweise eine exakte Übereinstimmung beim zweiten Produkt mit „Hammer“. Aufgrund der exakten Treffer sind die Produkte auf den Positionen eins und zwei am höchsten eingestuft worden. Das dritte Element „großer Gummihammer“ besitzt zwei Teiltreffer. Zum Beispiel kommt „groß“ in „großer“ vor. Da keine exakten Übereinstimmungen vorliegen, wird es auf der letzten der drei Positionen eingestuft. Die zugehörigen CTR zeigen, dass das letzte Element im Vergleich zum zweiten prozentual öfter geklickt wird und daher möglicherweise eine höhere Relevanz vorliegt.

Um dieses Suchergebnis anzupassen, soll nun ein Algorithmus angewendet werden, der die indizierte Relevanz der Produkte beachtet. Eine Möglichkeit dafür ist, den Score um einen multiplikativen Boost zu erweitern. Dieser Faktor ergibt sich aus der CTR. Zum Beispiel könnte eine CTR von:

- 100 Prozent in den Faktor „2“,

- 50 Prozent in den Faktor „1,5“ und
- 0 Prozent in den Faktor „1“ (entspricht keiner Veränderung des Scores)

übersetzt werden. In diesem Fall wird die CTR linear auf einen Bereich von Boosts abgebildet. Alternativ ist auch eine logarithmische Übersetzung möglich.
Tabelle 2.2 visualisiert eine mögliche Anpassung, welche ein CTA vornehmen könnte. In diesem Beispiel ist der initiale Score prozentual um die Rate, mit der das Element geklickt wurde, erhöht worden. Die sich ergebenden Boosts werden mit den ursprünglichen Scores der Produkte multipliziert. Es kommt die folgende Formel zum Einsatz:

$$Neuer\ Score = Score\ initial \cdot (1 + CTR)$$

Anschließend werden die Elemente entsprechend ihrer neuen Scores sortiert. Damit wird die in Tabelle 2.1 gezeigte Anordnung verändert. Das Produkt „großer Gummihammer“ verbessert aufgrund seiner CTR die Position von „3“ auf „2“.

Alte Position	Score initial	Boostfaktor	Neue Position	Neuer Score
1	600	1,4	1	840
2	500	1,04	3	520
3	400	1,4	2	560

Tabelle 2.2 - Beispiel der Funktionsweise des SOLR Scorings mit CTA

2.2 Bestehendes Projekt

In dieser Bachelorarbeit wird ein von der dotSource GmbH betreutes E-Commerce System betrachtet. In diesem Projekt kommt das Shopsystem SAP Commerce[1] zum Einsatz. Die Suche dieser Plattform basiert auf der Suchmaschine Apache SOLR[2] [SAP22g]. Sie setzt einige Funktionalitäten um, wie zum Beispiel die Paginierung von Suchergebnissen oder verschiedene Sortierungsmöglichkeiten [Sha19]. Die

[1] https://www.sap.com/germany/products/commerce-cloud.html
[2] https://solr.apache.org/

Verwendung des Click-Through-Scorings ist kein Standardmerkmal. Die durch Kundenverhalten indizierte Relevanz wird somit vernachlässigt. Das führt auf Systemen, welche auf SOLR basieren und keine zusätzliche Erweiterung verwenden, zur Präsentation unpassender Ergebnisse bei einigen Suchanfragen. So zum Beispiel auf dem Produktivsystem des von der dotSource GmbH betreuten Projekts bei der Abfrage nach „Schaufel“. Unter den ersten Elementen der Auflistung befinden sich Schneeschieber, Stapelkarren und Schaufeln für Radlager oder Gabelstapler. Von Mitarbeitenden des Unternehmens ist berichtet worden, dass dies nicht dem vom Kunden erwarteten Ergebnis entspricht. Die relevantesten Produkte seien Aluminiumschaufeln, welche auf den Positionen 14, 16 und 22 aufgeführt werden, sodass diese erst durch Scrollen für den Nutzer sichtbar werden. Weitere geeignetere Treffer sind sogar erst auf der zweiten Seite aufgelistet.

2.3 Vorstellung des bestehenden Prototyps

Grundlage dieser Bachelorarbeit ist ein Prototyp zur Erweiterung der Suchplattform SOLR. Dieser wurde vom Autor der vorliegenden Arbeit in der Programmiersprache Java entwickelt. Er wird in der Arbeit mit dem Titel „Einbeziehung von Click-Through-Daten im SOLR-Scoring“ [Erd22] beschrieben. Dabei wird thematisiert, wie es möglich ist, mit Hilfe von Klickdaten Einfluss auf die Suchergebnisse zu nehmen. In den folgenden Abschnitten wird die Entwicklung vorgestellt.

Die Implementierung des Prototyps lässt sich in zwei Module unterteilen. Das erste ist verantwortlich dafür, Click-Through-Daten (CTD) zu sammeln. Diese ergeben sich durch das Aufzeichnen von Interaktionen auf der Storefront des Webshops. Der zweite Teil kommt bei Suchanfragen durch den Kunden zum Einsatz. Hierbei wird das Suchergebnis auf Basis der erfassten CTD angepasst.

2.3.1 Datenbank zur Speicherung von Click-Through-Daten

Mit Click-Through-Daten werden Informationen bezüglich des Verhaltens von Nutzern erfasst. Um diese zu persistieren, erfolgt die Speicherung in einer Datenbank.

Durch das System SAP Commerce werden verschiedene relationale Datenbanken unterstützt [SAP22e]. Das bestehende Projekt nutzt das

Datenbankmanagementsystem (DBMS) MySQL[3]. Der Prototyp erweitert die existierende Datenbank um eine Tabelle für Benutzerinteraktionen. Die Datensätze dieser Tabelle entsprechen den CTD. Anlage A zeigt den Befehl zur Erstellung der Tabelle. Sie beinhaltet die folgenden fünf Spalten:

- Eine ID als Primärschlüssel
- Eine Produktidentifikation
- Einen Suchterm zur Zuordnung zu einer Abfrage
- Den Zeitpunkt der Erfassung
- Den Typ der Interaktion

Mit der letzten Spalte wird zwischen Klicks und Impressionen unterschieden. Als ein Klick wird der Aufruf einer Produktdetailseite ausgehend von einem Suchergebnis definiert. Eine Impression meint die Präsentation eines Elementes. Dabei muss es möglich sein, dass ein Nutzer dieses Element auswählen kann. Diese Informationen werden serverseitig erfasst und in der Datenbank abgelegt. Der Zeitpunkt wird genutzt, um Interaktionen, deren Erfassung zu weit in der Vergangenheit liegt, zu entfernen. Die Dauer, nach der ein Datenbankeintrag gelöscht wird, kann vom Shopbetreiber definiert werden.

Um eine Verbindung zur Datenbank herzustellen, wird ein JDBC Treiber[4] verwendet. Diese Software ist eine Bibliothek, die Funktionalitäten zur Verfügung stellt, um mit einer auf MySQL basierenden Datenbank zu kommunizieren [Pro22]. Dabei kommt die Programmiersprache Java zum Einsatz.

2.3.2 Bearbeitung von Click-Through-Daten

Bevor die CTD in der Datenbank abgelegt werden, werden diese bearbeitet. Diese Änderungen betreffen den erfassten Suchbegriff. Wie in Kapitel 2.3 von [Erd22] beschrieben wird, erfolgt die Speicherung „case-insensitive“. Das bedeutet, dass keine Unterscheidung zwischen Groß- und Kleinschreibung getroffen wird. Dafür werden

[3] https://www.mysql.com/de/
[4] https://jar-download.com/artifacts/mysql/mysql-connector-java/8.0.22

Buchstaben, welche durch den Benutzer großgeschrieben worden sind, durch kleine ersetzt. Bereits klein geschriebene bleiben unverändert.
Eine weitere Anpassung wird bei Suchanfragen mit mehreren Wörtern vorgenommen. In diesem Fall erfolgt die Verarbeitung in Unabhängigkeit von der Reihenfolge der Teilterme. Um dies zu erreichen, werden sie lexikographisch sortiert. Damit werden verschiedene Anordnungen zusammengeführt.
Der Grund für die Bearbeitung des Suchbegriffs ist die Menge der Daten, die einer Suchanfrage zugeordnet werden können. Durch das Zusammenführen wird diese Menge für einige Anfragen erhöht. Das soll dafür sorgen, ein möglichst repräsentatives Ergebnis zu erzielen [Erd22].
Im Folgenden wird diese Vorgehensweise am Beispiel der Suchbegriffe „HAMMER Groß“, „GrOß hAMMER“ oder „groß hammer“ erläutert. Letzteres entspricht der bereits lexikographisch sortierten Anordnung von Teiltermen, bestehend aus ausschließlich kleingeschriebenen Buchstaben. Für jede dieser Eingaben ist der verwendete String „groß Hammer“. Abbildung 2.1 visualisiert die einzelnen Verarbeitungsschritte.

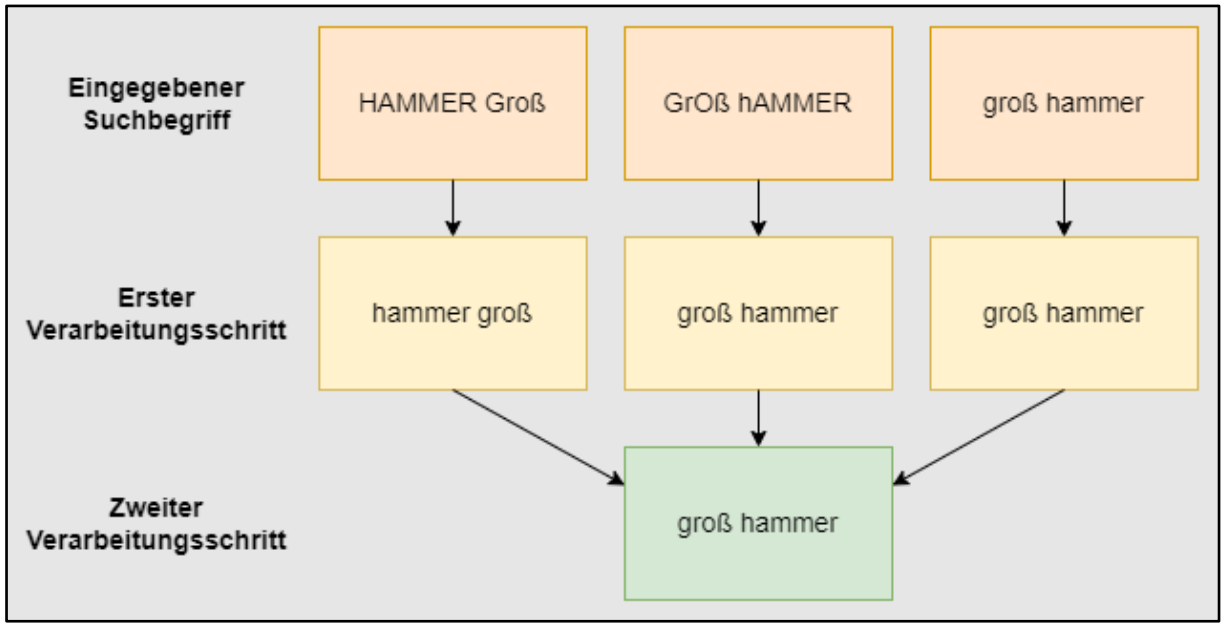

Abbildung 2.1 - Verarbeitungsschritte des Suchbegriffs für CTD

2.3.3 Funktionsweise des Click-Through-Algorithmus

Beim Benutzen der Volltextsuche liefert die Suchmaschine SOLR Produkte, die zum Suchbegriff passen könnten. Die Reihenfolge der Ergebnisse wird durch einen Score bestimmt. Dieser wird in Abgängigkeit zur Summe der indexierten Produktmerkmale, wie zum Beispiel der Produktbeschreibung ermittelt [SAP22d].

Der Click-Through-Algorithmus ist dafür verantwortlich, das Suchergebnis unter Verwendung der CTD anzupassen. Die Idee ist, durch multiplikative Boosts den ursprünglichen Score der Suchergebnisse zu verändern. Dies wird in Abschnitt 2.3.1 von [Erd22] begründet. Im Folgenden wird die Funktionsweise des CTA am Beispiel der Volltextsuche nach „Hammer groß“ erläutert.

Wie bei der Speicherung von CTD wird auch beim Absenden einer Volltextsuche der eingegebene Suchbegriff angepasst. Dies geschieht nur für den CTA und ist für den Benutzer nicht sichtbar. Entsprechend des Abschnitts „2.3.2 Bearbeitung von Click-Through-Daten“ erfolgt die Veränderung. Im Beispiel wird das Zeichen „H“ durch „h“ ersetzt. Weiterhin besteht der Suchbegriff aus mehreren Teiltermen. Diese werden lexikographisch sortiert. Es ergibt sich der String „groß hammer“. Dies wäre auch das Resultat bei den in Abbildung 2.1 aufgeführten Beispielen.

Mit diesem String wird in der Datenbank nach passenden Einträgen gesucht. Aufgrund der Anpassungen des Suchbegriffs werden möglicherweise eine höhere Anzahl von Interaktionen berücksichtigt. Die Antwort enthält Informationen über die Produkte, für die mindestens eine Interaktion zur Suchanfrage erfasst wurde. Es wird eine Liste zurückgegeben, deren Elemente die:

- Anzahl der Impressionen,
- Anzahl der Klicks, sowie die
- zugehörige Produktidentifikation

beinhalten.

Im Prototyp ist umgesetzt, dass zuerst die Click-Through-Rate berechnet wird. Diese Kennzahl wird linear auf den Bereich zwischen eins und zwei abgebildet. Dieser Wert ergibt den Boostfaktor. Damit der Shopbetreiber den Einfluss des Click-Through-

Scorings bestimmen kann, wird zusätzlich eine Gewichtung verwendet. Diese ist konfigurierbar. Der neue Score ergibt sich durch folgende Formel:

$$Neuer\ Score = Score\ \cdot Boostfaktor^{Gewichtung}$$

Dadurch werden folgende Anforderungen realisiert:

- Bei einem Boostfaktor von „1“ wird der Score nicht verändert.
- Mit höherer Gewichtung steigt der Einfluss des Click Through-Scorings.
- Mit höherem Boostfaktor, bzw. einer höheren CTR, wird der Score in Relation zu anderen Elementen erhöht.

Anschließend werden die Elemente anhand des neuen Scores sortiert. Die sich ergebende Reihenfolge wird dem Kunden präsentiert.

2.3.4 Anpassung des Ergebnisses mittels eines SOLR-Plugins

Um mit Hilfe dieses CTA die Suchergebnisse anzupassen, nutzt der Prototyp ein Plugin für den SOLR. Dieses ermöglicht eine individuelle Erweiterung der bestehenden Implementierung [Apa19a]. Es beinhaltet einen Rescorer, der auf Basis der Standard-Berechnung einen neuen Score für die Produkte festlegt. Die Implementierung dieser Komponente wird in Anlage B aufgeführt.

Um dieses Plugin zu verwenden, wird es über die SOLR-Query aufgerufen. Diese charakterisiert eine Anfrage an den SOLR-Server. Sie enthält unter anderem den Suchbegriff oder Filter. Die folgende Abbildung 2.2 zeigt wie das Plugin der Query hinzugefügt wird.

```
1  public void populate(SearchQueryConverterData source, SolrQuery target)
2  {
3      [...]
4      target.add(„rq“, „{!ctrParser“}
5  }
```

Abbildung 2.2 - Aufruf des SOLR-Plugins

2.4 Beteiligte Komponenten beim Click-Through-Scoring im Überblick

Abbildung 2.3 gibt eine Übersicht darüber, welche Komponenten beim Click-Through-Scoring beteiligt sind. Durch den Kunden abgesendete Suchanfragen werden durch

einen Controller der E-Commerce Plattform SAP Commerce entgegengenommen. Dieser formuliert eine Query und erfragt von einem SOLR-Server das Suchergebnis. Durch die Komponente wird zunächst auf Basis der Standard-Berechnung ein Suchergebnis ermittelt. Um dieses entsprechend den erfassten Benutzerinteraktionen anzupassen, werden aus der Datenbank zum Suchbegriff zugehörige CTD abgefragt. Unter Verwendung dieser Informationen verändert der SOLR-Server die Scores der Produkte. Es ergibt sich eine neue Auflistung die an den Controller zurückgegeben und dann dem Benutzer präsentiert wird.

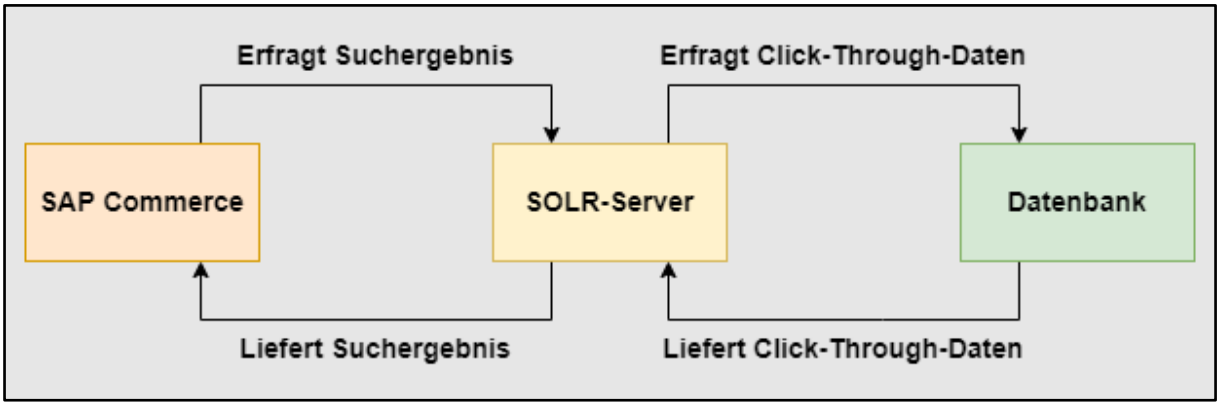

Abbildung 2.3 - Beteiligte Komponenten beim Click-Through-Scoring

3 Analyse des bestehenden Prototyps

Der folgende Abschnitt thematisiert einige Möglichkeiten zur Verbesserung des existierenden Prototyps. Mit diesen Änderungen soll das Deployment auf ein Produktivsystem ermöglicht werden.

3.1 Logik zur Aktivierung oder Deaktivierung der Erweiterung

Um in einem Produktivbetrieb Probleme zu vermeiden, werden Testsysteme zur frühzeitigen Erkennung von Fehlern genutzt. Diese sollten der Produktivumgebung möglichst ähnlich sein [Aug21]. Dennoch ist eine exakte Replikation meistens nur schwer zu ermöglichen [BSW18]. Aus diesem Grund können trotz aufwändig getesteter Software jederzeit Probleme auftreten. Um flexibel reagieren zu können, bietet sich die Entwicklung von Feature Toggles an. Sie ermöglichen Änderungen im laufenden Betrieb, indem bestimmte Funktionen an- oder ausgeschaltet werden [Ebb15]. Diese Vorgehensweise soll auch beim Click-Through-Scoring realisiert werden.

Neben der Flexibilität ergibt sich ein weiterer Vorteil. Aufgrund der Zweiteilung der Implementierung ist es möglich, zwei Feature Toggle umzusetzen. Das Sammeln von Daten und die Anpassung des Scorings sind somit unabhängig voneinander aktivierbar. Es kann ein getrennter Livegang beider Teile realisiert werden. Ein Grund dafür könnte sein, dass das Suchergebnis erst verändert werden soll, wenn eine für den Shopbetreiber ausreichende Menge von CTD erfasst worden ist.

3.2 Integration der Datenhaltung in die Logik des Systems

Im bestehenden Projekt werden das Produktivsystem, mehrere Testsysteme und die lokalen Systeme der Beteiligten des Entwicklungsteams betrieben. Durch diese sollen unterschiedliche Datenbanken verwendet werden. Der Grund dafür ist, dass Test- und Produktivsystem in Bezug auf ihre technische Umsetzung möglichst identisch sein, gleichzeitig aber voneinander getrennt existieren sollen [Aug21]. Daraus folgt die Erstellung einer eigenständigen Datenbank für jedes System. Diese sind unter Angabe verschiedener Verbindungseigenschaften zu erreichen.

Der Prototyp nutzt eine Datenbank, welche ausschließlich für den Zweck CTD zu speichern, erstellt wurde. Ergebnis dieser Arbeit soll eine Erweiterung sein, die flexibel auch in anderen SAP Commerce Systemen verwendet werden kann. In diesen ist es möglich, dass die Datenbankinfrastruktur auf eine andere Weise realisiert wird. In dem Fall müssten einige Teile, wie die Verbindung zur Datenbank, angepasst werden. Um die Wiederverwendbarkeit der Implementierung zu vereinfachen, sollte ein anderer Lösungsansatz betrachtet werden.

3.2.1 Definition einer Datenstruktur im Kontext der Plattform SAP Commerce

Durch die Plattform SAP Commerce werden verschiedene Daten gespeichert. Dazu gehören Entitäten wie Produkte, Kundeninformationen oder Bestellungen. Für diese wird während des Build-Prozesses ein entsprechendes Datenbankschema erstellt [Fah20b]. Es ist möglich, weitere Strukturen zu definieren [SAP22c]. Somit kann unter Nutzung bestehender Logiken der E-Commerce Plattform das Datenbankmodell erweitert werden. Dabei kommt das Object-Relational Mapping[5] (ORM) zum Einsatz [SAP22f]. Diese Technik ermöglicht die Nutzung einer relationalen Datenbank in einem Programm, das in einer objektorientierten Programmiersprache geschrieben worden ist. Mittels einer Abstraktion werden Objekte in Tabellen des relationalen Schemas übersetzt [Lia22].

Dieses Vorgehen ist auch für Click-Through-Daten möglich. Hierfür ist die Definition des Datentyps „Interaktion“ im Kontext von SAP Commerce notwendig. Weiterhin muss die Abfrage der Daten umgestellt werden, sodass bestehende Logiken der Plattform verwendet werden.

Dieser Lösungsansatz ist unabhängig vom verwendeten System nutzbar. Der Grund dafür ist die automatische Erstellung der Datenbank während des Build-Prozesses. Dieser erfolgt spezifisch für das entsprechende System und gewährleistet die Verwendung einer eigenständigen Datenbank. Zusätzlich vereinfacht diese Lösung

[5] https://de.wikipedia.org/wiki/Objektrelationale_Abbildung

die Implementierung der Erweiterung in anderen SAP Commerce Systemen, weil kein Aufwand für das Aufsetzen einer Datenbank entsteht.

3.2.2 Dynamische Ermittlung von Verbindungsinformationen

Im Abschnitt „2.3.1 Datenbank zur Speicherung von Click-Through-Daten“ wird beschrieben, dass der Prototyp einen JDBC-Treiber nutzt, um Verbindungen zur Datenbank herzustellen. Dafür wird ein Uniform Resource Locator (URL) konfiguriert [BS15], S. 513 f.. Dieser enthält notwendige Verbindungseigenschaften, wie den Servernamen oder die Portnummer, über den der MySQL-Server erreichbar ist [Eng22].

```
1  private static final String USERNAME = „root“;
2  private static final String PASSWORD = /**/;
3  private static final String CONNECTION_STRING =
4      „jdbc:mysql://[host]/hybris?useConfigs=maxPerformance
5          &characterEncoding=utf8&nullCatalogMeansCurrent=true“;
```

Abbildung 3.1 - Datenbankverbindung mittels statischen URLs (Prototyp)

Wie in der Abbildung 3.1 gezeigt, werden durch den Prototypen Konstanten für die Datenbankverbindung verwendet. Grund dafür ist, dass dieser nur auf einem lokalen Rechner entwickelt wurde und auf diesem System zum Beispiel ein statischer Anfrage-URL verwendet werden konnte. Aufgrund der Architektur mit mehreren Systemen ist der URL in Abhängigkeit zu diesen zu verwenden. Dafür muss eine entspreche Konfiguration vorgenommen werden. Es unterscheidet sich der Teil „[host]“ (siehe Zeile 4) in den verschiedenen Systemen. Das bedeutet, dass unter Verwendung des Strings für die lokalen Systeme der beteiligten Entwickler keine Verbindung zur Datenbank des Produktiv- oder Testsystems möglich ist. Weiterhin müssen Benutzername und Passwort angegeben werden. Diese stimmen für die Systeme des betrachteten Projekts nicht überein, können sich in Zukunft ändern und unterscheiden sich bei der Implementierung bei anderen Unternehmen.

Auf jedem System der SAP Commerce Plattform gibt es Dateien, die systemspezifische Informationen enthalten [Fah21]. Um einen Aufwand für die

Konfiguration der Datenbankverbindung zu verhindern, können bestehende und durch die Standardlogik verwendete Einstellungen genutzt werden.

3.3 Zusammenführung gleichartiger Interaktionen

Beim Click-Through-Scoring werden die Anzahlen von Klicks und Impressionen eines Elementes analysiert und für die Bewertung der Relevanz in Abhängigkeit zu einem Suchbegriff verwendet [OMT21]. Daraus ergibt sich die Definition einer Interaktion durch ihre Produktidentifikation, ihren Interaktionstyp und den eingegebenen Suchbegriff. Der bestehende Prototyp erfasst mit der Zeit ein weiteres Merkmal. Dabei ist es nicht notwendig die exakte Uhrzeit zu erfassen. Der Grund dafür ist, dass Click-Through-Daten über einen längeren Zeitraum gesammelt werden, durch den eine genaue Angabe der Uhrzeit an Bedeutung verliert. Es ist Anforderung, dass lediglich ein Datum angegeben wird, um ältere Elemente zu entfernen. Da somit ausschließlich der Tag der Interaktion gespeichert wird, ist es möglich, dass gleichartige Einträge in die Datenbank übernommen werden. Bereits zwei Suchanfragen an einem Tag mit demselben Suchbegriff sorgen dafür, dass zwei identische Impressionen gespeichert werden. Um diese Doppelung zu vermeiden, sollte auf geeignete Weise eine Zusammenführung stattfinden. Das sorgt dafür, dass Speicherplatz gespart wird und bei der Abfrage weniger Datenbankeinträge durchsucht werden.
Eine Vereinigung ist durch Nutzung eines Zählers möglich. Dieser gibt an, wie viele Interaktionen mit den entsprechenden Eigenschaften erfasst worden sind. Dafür ist es notwendig, dass die Datenstruktur für Interaktion um eine Spalte erweitert wird.

3.4 Website Traffic auf der betrachteten Online-Präsenz

Auf Online-Präsenzen von Unternehmen hat die Ladezeit einen Einfluss auf den Gewinn, der erzielt werden kann [Web19]. Es ist ein Ziel der Unternehmen, eine geringe Seitenladezeit auch bei einer hohen Anzahl von Benutzern zu gewährleisten. Im Folgenden wird der Website Traffic des von der dotSource GmbH betreuten E-Commerce Systems thematisiert. Abbildung 3.2 visualisiert wie viele Volltextsuchen auf dem Produktivsystem abgesendet werden.

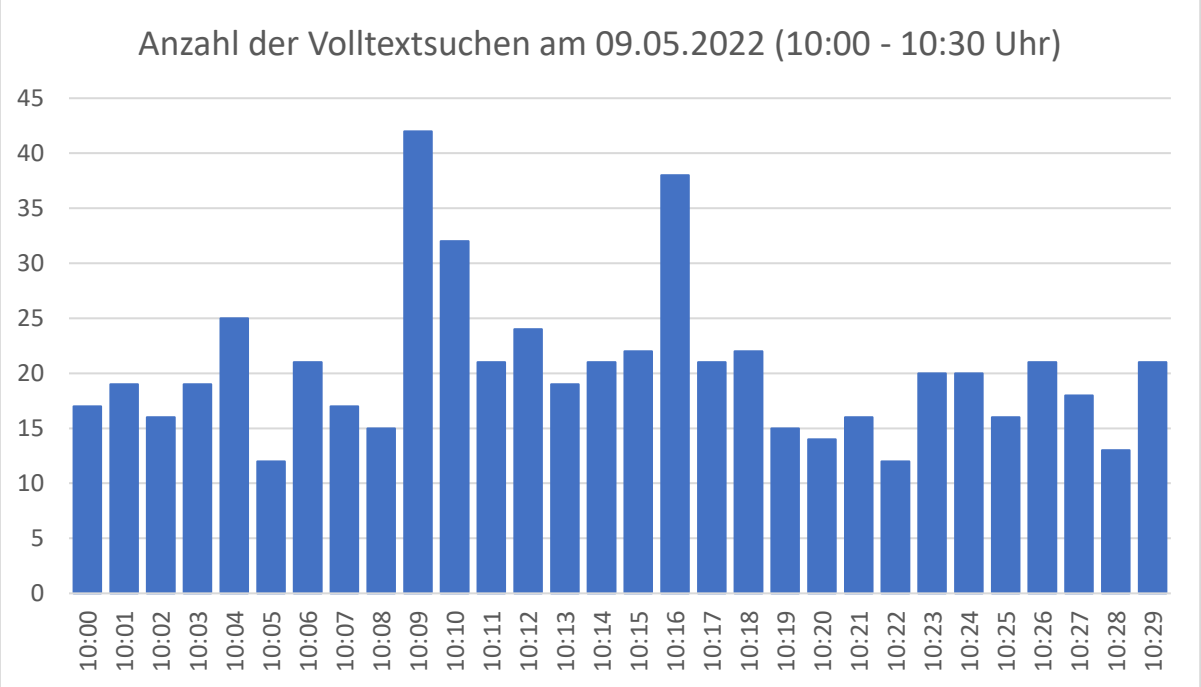

Abbildung 3.2 - Anzahl der Volltextsuchen am 09.05.2022

Das Diagramm zeigt einen Bereich von 30 Minuten am neunten Mai im Jahr 2022. Zur Auswertung dieser Informationen ist das Tool Kibana[6] genutzt worden. Dieses wird zur Visualisierung und Analyse von Daten verwendet [Ela22]. Im angegebenen Bereich wurden insgesamt 609 Aufrufe erfasst. Das entspricht einem Durchschnitt von 20,3 Volltextsuchen pro Minute. Zwischen 10:09 und 10:10 erfolgten sogar 42 Suchanfragen.

3.4.1 Nutzung eines Puffer-Algorithmus

Eine Volltextsuche liefert im bestehenden Projekt bis zu 24 Produkte. Das entspricht der konfigurierten maximalen Anzahl von Elementen auf einer Seite. Mit der Implementierung des Prototyps würde für jedes der geladenen Produkte ein Schreibvorgang an der Datenbank initiiert werden. Dabei nutzt jede Verbindung zur Datenbank Ressourcen des Servers [Goo22]. Unter der Voraussetzung, dass auf jede Anfrage ein Suchergebnis mit bis zu 24 Produkten folgt, wären pro Sekunde

[6] https://www.elastic.co/de/what-is/kibana

durchschnittlich 8,12 schreibende SQL-Befehle notwendig. Es ist zu beachten, dass dabei ausschließlich die Erfassung von Impressionen betrachtet wurde. Das Speichern von Klicks beim Aufruf von Produktdetailseiten erfordert zusätzliche Schreibvorgänge. Um auch bei hoher Auslastung des Webshops die Kommunikation mit der Datenbank performant zu gestalten, sollte die Anzahl der Datenbank-Interaktionen verringert werden. Ein Lösungsansatz, der dieses Ziel verfolgt, ist die Zwischenspeicherung von Interaktionen. Das bedeutet, dass nicht sofort versucht wird, erfasste Interaktionen zu persistieren. Unter Verwendung des Arbeitsspeichers des Servers können die Informationen temporär gespeichert werden. Nach Ansammlung einer bestimmten Menge, werden diese durch einen Vorgang in die Datenbank übernommen. Dadurch wird verhindert, dass ein für jede Suchanfrage mehrere Datenbank-Interaktionen ausgeführt werden müssen.

3.4.2 Nutzung eines Cache-Algorithmus

Die Anzahl der Volltextsuchen hat sowohl auf das Erfassen von Interaktionen als auch auf die Performance des Click-Through-Scorings Einfluss. Bei durchschnittlich 20,3 Suchabfragen pro Minute ist es möglich, dass innerhalb eines kurzen Zeitraums vor allem beliebte Suchbegriffe mehrfach verwendet werden. Mit Implementierung des Prototyps ist für jede Abfrage eine Datenbankinteraktion notwendig. Um dies zu verhindern, kann ein Cache-Algorithmus verwendet werden. Dieser speichert die zu einem Suchbegriff zugehörigen CTD im Arbeitsspeicher. Bei weiteren Abfragen mit diesem Suchbegriff wird die Ladezeit verringert, indem die zuvor zurückgegebenen Daten verwendet werden. Dabei ist keine Interaktion mit der Datenbank notwendig.

3.5 Nutzung einer weiteren Datenbanktabelle

Wie in Abbildung 3.2 (siehe Abschnitt 3.4) gezeigt, ist auf dem Produktivsystem mit vielen Suchanfragen zu rechnen. Trotz der Zusammenführung gleichartiger Interaktionen mit Hilfe eines Zählers gibt es viele Kombinationsmöglichkeiten aus Suchbegriff, Produkt, Datum und der Unterscheidung zwischen Klick und Impression. Für jede erfasste Variation wird ein Eintrag in die Tabelle für Interaktionen vorgenommen.

Bei der Berechnung eines neuen Scores sollen durch den SOLR-Server zugehörige Klickdaten verwendet werden. Hierfür müssen zur Suche passende Interaktionen gefiltert und aus diesen CTD gebildet werden. Das Ergebnis dieses Vorgangs ist eine Liste von Daten, bestehend aus:

- Produktidentifikation,
- Anzahl von Klicks und
- Anzahl von Impressionen.

Die Bildung, deren Aufwand mit zunehmender Anzahl von Datenbankeinträgen steigt, erfolgt bei jeder Suchabfrage und hat damit einen Einfluss auf die Antwortzeit des SOLR-Servers.

Um den Zeitaufwand bei der Suchabfrage zu vermeiden, sollten die Berechnung der CTD zuvor durchgeführt werden. Der hierfür betrachtete Lösungsansatz ist die regelmäßige Bildung in definierten Intervallen, zum Beispiel einmal täglich. Das betrifft die CTD aller Suchbegriffe. Anschließend erfolgt die Persistierung in einer separaten Datenbanktabelle. Nachteil dieser Lösung ist, dass neu erfasste Interaktionen erst zeitversetzt einen Einfluss haben. Allerdings können bereits vor der Aktivierung des Click-Through-Scorings Daten gesammelt werden, sodass bei der Berechnung eine angemessene Anzahl von CTD verfügbar sein kann. Weiterhin wird einmalig im gewählten Intervall ein höherer Aufwand im Vergleich zur bisherigen Lösung benötigt. Dennoch kann die Bildung zu einem Zeitpunkt geringer Auslastung gewählt werden. Der Vorteil ist, dass die zu durchsuchende Menge und damit der Zeitaufwand minimiert wird. Zusätzlich kann bei der Berechnung eine Mindestanzahl von Klicks und Impressionen festgelegt werden. So erfolgt die Filterung von kleinen Datensätzen.

4 Optimierung des bestehenden Prototyps

In diesem Kapitel wird die Umsetzung der im Kapitel 3 beschriebenen Verbesserungsmöglichkeiten thematisiert. Damit soll der bestehende Prototyp optimiert werden. Einige Änderungen sind von einem Inhouse Entwickler vorgenommen worden. Nicht durch den Autor der vorliegenden Arbeit implementierte Teile werden entsprechend gekennzeichnet.

4.1 Umsetzung zweier Feature Toggle

„[...] Feature Toggling beschreibt das Vorgehen innerhalb der Softwareentwicklung, Funktionen einer Software ein- oder auszuschalten bzw. zur Laufzeit zu verändern“ [Tak22]. Mit dieser Technik soll im bestehenden Projekt eine Möglichkeit geschaffen werden, die beiden Teile der Erweiterung zu aktivieren oder zu deaktivieren. Somit kann entschieden werden, ob die Erfassung von Benutzerinteraktionen stattfindet und das Click-Through-Scoring angewendet wird. Dabei ist darauf zu achten, dass diese voneinander unabhängig sind.

Zu Beginn erfolgt die Deklaration der Feature Toggle. Sie werden durch Variablen des Datentyps Boolean implementiert. Entsprechend der Abbildung 4.1 werden diese einer bestehenden Auflistung hinzugefügt.

```
public enum Feature {
      [...]
      /**
       * Enables the click-through-scoring extension for solr
       */
     ENABLE_CLICK_THROUGH_SCORING(false),
     /**
       * Enable collecting click-through-data for the click-through-scoring
         extension for solr
       */
     ENABLE_CLICK_THROUGH_DATA_COLLECTING(false);
}
```

Abbildung 4.1 - Deklaration der Feature Toggle

Sie werden mit dem Wert „false“ initialisiert. Das bedeutet, dass die Funktionen im Backoffice Administration Cockpit[7] der Plattform aktiviert werden müssen. Dies ist im laufenden Betrieb möglich.
Unter Verwendung dieser Variablen werden im Quellcode Verzweigungen erzeugt, die bestimmen, ob die beiden Module der Erweiterung ausgeführt werden. In der Abbildung 4.2 ist die Bedingung für das Sammeln von Impressionen präsentiert. Dabei wird die Methode „textSearch“ (siehe Zeile 1) bei jeder Volltextsuche aufgerufen. Entspricht der in Abbildung 4.1 definierte Feature Toggle dem Wert „true“, werden die Interaktionen erfasst. Anlage C und Anlage D zeigen die entsprechenden Logiken zur Erfassung von Klicks, beziehungsweise zur Anpassung der Suchergebnisse.

```
1 public String textSearch([...]) {
2     [...]
3     if (getFeatureToggleService().isFeatureActive(
4         Feature.ENABLE_CLICK_THROUGH_DATA_COLLECTING))
5     {
6         //save impressions
7     }
8     [...]
9 }
```

Abbildung 4.2 - Feature Toggle für das Sammeln von Impressionen

Durch diese Änderungen kann flexibel auf Probleme reagiert werden. Beispielsweise kann der Shopbetreiber das Click-Through-Scoring deaktivieren, wenn durch diese Erweiterung ein Fehler bei der Suche nach Produkten auftritt.
Weiterhin ist es möglich, zunächst nur den Feature Toggle für das Sammeln der CTD zu aktivieren. Der Kunde der dotSource GmbH kann nach eigenem Ermessen entscheiden, wann eine ausreichende Menge von Daten erfasst worden ist. Nach Erreichen dieses Grenzwertes wird das Click-Through-Scoring aktiviert.

[7] https://s.dotsource.de/backoffice

4.2 Integration der Datenhaltung in die Logik des Systems

4.2.1 Definition der Datenstruktur „ProductInteraction“

Zur Steigerung der Wiederverwendbarkeit der Erweiterung sollen Interaktionen in die Datenbankstruktur des bestehenden Systems aufgenommen werden. Hierfür wird die Konfigurationsdatei „items.xml“ angepasst. Diese dient der Definition von sogenannten Typen [SAP22c]. Sie sind die Grundlage für Objekte, welche mittels des ORMs in die Datenbank übernommen werden.

Damit beim Build-Prozess die automatische Erzeugung einer weiteren Datenbanktabelle stattfindet, wird zuerst ein Typ mit dem Code „ProductInteraction“ ergänzt. Dies wird in Abbildung 4.3 dargestellt. Wie in Zeile 3 gezeigt, wird der Name der Tabelle und eine Identifikation des Typs angegeben. Beide müssen global eindeutig sein [SAP22b]. Es folgt die Definition der Attribute und Indizes des Typs.

```
 1 <itemtype code="ProductInteraction">
 2     <description>Product interatction logging entry</description>
 3     <deployment table="ProductInteraction" typecode="16109"/>
 4     <attributes>
 5         [...]
 6     </attributes>
 7     <indexes>
 8         [...]
 9     </indexes>
10 </itemtype>
```

Abbildung 4.3 - items.xml: Definition der Datenbanktabelle

Unter dem Tag „attributes“ (siehe Zeile 4 in Abbildung 4.3) werden die Eigenschaften des Objektes, beziehungsweise die Spalten der Datenbank bestimmt. Gemäß der Abbildung 4.4 wird für jedes Attribut ein Tag „attribute“ angegeben. Dieses beinhaltet:

- den Namen des Attributs,
- den Typ des Attributs,
- eine Beschreibung (dient nur der Dokumentation),
- die Art der Speicherung und
- erweiterte Einstellungen.

Letzteres wird im Tag „modifiers“ angegeben. Unter anderem kann bestimmt werden, ob eine Eigenschaft optional ist. Die hier definierten Attribute werden als Pflichtfelder markiert. Die Art der Speicherung (siehe „<persistence />“ in Abbildung 4.4) wird auf „property“ gesetzt. Das entspricht einer persistenten Speicherung der Entität [SAP22b]. Bei Angabe von „dynamic“ würde keine Persistierung in die Datenbank vorgenommen werden. Wie in Abschnitt 3.3 beschrieben, sollen gleichartige Interaktionen durch Nutzung eines Zählers zusammengeführt werden. Die Umsetzung dieses Attributs wird in Abschnitt 4.3 vorgenommen.

```
 1 <attributes>
 2     <attribute qualifier="productCode" type="java.lang.String">
 3         <description>Produkt-/Artikelnummer</description>
 4         <persistence type="property"/>
 5         <modifiers optional="false"/>
 6     </attribute>
 7     <attribute qualifier="searchTerm" type="java.lang.String">
 8         <description>Suchbegriff</description>
 9         <persistence type="property"/>
10         <modifiers optional="false"/>
11     </attribute>
12     <attribute qualifier="interactionType"
13         type="ProductInteractionType">
14         <description>Art der Interaktion</description>
15         <persistence type="property"/>
16         <modifiers optional="false"/>
17     </attribute>
18     <attribute qualifier="interactionDate" type="java.util.Date">
19         <description>Datum (ohne Uhrzeit) der Interaktion</description>
20         <persistence type="property"/>
21         <modifiers optional="false"/>
22     </attribute>
23 </attributes>
```

Abbildung 4.4 - items.xml: Definition der Datenbankspalten

Eine Besonderheit bei der Definition der Struktur stellt der Datentyp des Attributes „interactionType“ dar (siehe Zeile 12 in Abbildung 4.4). Es wird „ProductInteractionType“ angegeben. Dies ist eine Enumeration, welche zuvor in der Datei „items.xml“ deklariert wird. Wie in Abbildung 4.5 gezeigt, werden die zwei möglichen Werte:

- „CLICK“ und

- „IMPRESSION“

definiert.

```
1  <enumtype code="ProductInteractionType">
2      <value code="CLICK"/>
3      <value code="IMPRESSION"/>
4  </enumtype>
```

Abbildung 4.5 - items.xml: Definition der Enumeration „ProductInteractionType“

Diese Enumeration wird im Build-Prozess angelegt. Durch die Logik der E-Commerce Plattform wird für sie ein Eintrag in einer weiteren Tabelle angelegt. Sie beinhaltet Enumerationswerte mit einem zugehörigen Schlüssel, welcher durch den MySQL-Datentyp „bigint“ repräsentiert wird. In der Spalte „interactionType“ (siehe Zeile 12 in Abbildung 4.4) kommen lediglich diese Schlüssel zum Einsatz.

Um die Performance von Datenbankabfragen zu optimieren, können Indizes verwendet werden [aco22]. Sie bestehen aus Verweisen auf bestimmte Speicherbereiche. Damit ermöglichen sie eine schnellere Ermittlung der gesuchten Datensätze im Vergleich zur sequenziellen Suche [Beg19]. Es wurden zwei Indizes implementiert. Sie werden in Abbildung 4.6 gezeigt.

```
 1 <indexes>
 2     <index name="prodSearchTermIDX">
 3         <key attribute="productCode"/>
 4         <key attribute="searchTerm"/>
 5     </index>
 6     <index name="prodSearchInterDateIDX" unique="true">
 7         <key attribute="productCode"/>
 8         <key attribute="searchTerm"/>
 9         <key attribute="interactionType"/>
10         <key attribute="interactionDate"/>
11     </index>
12 </indexes>
```

Abbildung 4.6 - items.xml: Definition der Indizes

Der erste Index mit dem Namen „prodSearchTermIDX“ (siehe Zeile 2 in Abbildung 4.6) dient der Optimierung von lesenden Befehlen, die bei der Abfrage von CTD verwendet werden. Dabei wird in Abhängigkeit zu den Datenbankspalten „productCode“ und „searchTerm“ gesucht.

Als zweiter Index wird „prodSearchInterDateIDX“ (siehe Zeile 6 in Abbildung 4.6) definiert. Er beinhaltet die vier Attribute, die eine Interaktion charakterisieren. Dies wird bei Schreibvorgängen verwendet, da anhand dieser Merkmale festgestellt wird, ob eine Interaktion bereits in der Datenbank vorhanden ist. In diesem Fall soll der Zähler dieses Eintrags erhöht werden.
Die in diesem Kapitel beschriebenen Änderungen wurden von einem Inhouse Entwickler umgesetzt. Anlage E zeigt die vollständige Definition des Typs „ProductInteraction“.

4.2.2 Schnittstelle zur Persistierung von Interaktionen

Nach Umsetzung der im vorherigen Abschnitt beschriebenen Änderungen kann eine Standardlogik von SAP Commerce genutzt werden, um auf die Datenbank des Systems zuzugreifen. Dafür kommt die Klasse „ModelService“ zum Einsatz. Sie dient der Verwaltung von Datenstrukturen und stellt Funktionalitäten wie das:

- Erstellen,
- Speichern,
- Aktualisieren oder
- Löschen

von Datenbankeinträgen zur Verfügung [Ali18]. Abbildung 4.7 zeigt die Methode zur Persistierung einer Liste von Benutzerinteraktionen. Sie sind vom Typ „ProductInteractionModel“ (siehe Zeile 4). Diese Klasse wird auf Basis der Definition des Typs „ProductInteraction“ in der Datei „items.xml“ (siehe Abbildung 4.3) automatisch erstellt.
Unter Verwendung der vordefinierten Methode „saveAll“ (siehe Zeile 7 in Abbildung 4.7) werden die Interaktionen dem „ModelService“ übergeben. Die Klasse speichert alle Elemente der Auflistung. Schlägt dieser Vorgang, zum Beispiel aufgrund eines fehlenden Pflichtfeldes, fehl, erfolgt eine entsprechende Fehlerbehandlung. Dabei wird keines der angegebenen Elemente persistiert [SAP22a]. Die vollständige Implementierung dieser Schnittstelle wird in Anlage G aufgeführt. Diese beinhaltet Methoden zur Löschung und zum Auslesen von Datenbankeinträgen.

```
1  private ModelService modelService;
2
3  public boolean addInteractionsToDatabase(final
4      Collection<ProductInteractionModel> interactions)
5  {
6      try {
7          modelService.saveAll(interactions);
8          return true;
9      }
10     catch (final ModelSavingException e) {
11         LOG.error(e.getMessage(), e);
12         return false;
13     }
14 }
```

Abbildung 4.7 - Methode zum Hinzufügen von Interaktionen zur Datenbank

4.2.3 Dynamische Ermittlung des URLs zur Datenbankverbindung

Die Plattform SAP Commerce verwendet zwei Arten von Konfigurationsdateien zur Definition von Einstellungen. Diese sind „project.properties“ und „local.properties“ [Fah21]. Sie bestehen aus Schlüssel-Wert-Paaren. Es gibt eine globale „project.properties“-Datei, die Standardeinstellungen für die Plattform beinhaltet. Dazu gehören unter anderem:

- der Datenbanktreiber,
- Benutzername und Passwort für die Datenbank und
- der URL zur Verbindung mit der Datenbank [Raj22].

Unter Nutzung systemspezifischer „local.properties“-Dateien können die in den projektspezifischen „project.properties“-Dateien hinterlegten Standardkonfigurationen überschrieben werden. Um unter Verwendung der Schlüssel auf die Konfigurationswerte zuzugreifen, kommt die Klasse „ConfigurationService“, eine Standard-Logik der Plattform, zum Einsatz [Tri21]. Um die Datenbankverbindung in Abhängigkeit zum System herzustellen, wird der statische Anfrage-URL durch die konfigurierten Werte der „local.properties“-Dateien ersetzt. Die Umsetzung wird im Folgenden beschrieben.

Um der Erweiterung notwendige Verbindungseigenschaften bekannt zu machen, werden sie beim Aufruf übermittelt. Dies wird in Abbildung 4.8 gezeigt. Es kommen

drei Schlüssel für die Anfrage-URL, den Benutzernamen und das Passwort zum Einsatz (siehe Zeilen 1 - 3). Diese sind für alle auf SAP Commerce basierenden Systeme identisch. Mit ihnen werden die zugehörigen Konfigurationswerte ermittelt. Zuletzt werden sie als Parameter beim Aufruf des SOLR-Plugins übergeben (siehe Zeile 16), sodass die Erweiterung korrekte Verbindungseigenschaften nutzen kann.

```
1  private static final String MYSQL_DB_URL = „db.url“;
2  private static final String MYSQL_DB_USERNAME= „db.username“;
3  private static final String MYSQL_DB_PASSWORD= „db.password“;
4  [...]
5  if(getFeatureToggleService()
6      .isFeatureActive(Feature.ENABLE_CLICK_THROUGH_SCORING))
7  {
8      String paramDBUrl = „ dbURL=“ + configurationService
9          .getConfiguration().getString(MYSQL_DB_URL);
10     String paramDBUsername = „ dbUsername=“ + configurationService
11         .getConfiguration().getString(MYSQL_DB_USERNAME);
12     String paramDBPassword = „ dbPassword=“ + configurationService
13         .getConfiguration().getString(MYSQL_DB_PASSWORD);
14
15     target.add(„rq“, „{!ctrParser“
16         + paramDBUrl + paramDBUsername + paramDBPassword + „}“);
17 }
```

Abbildung 4.8 - Aufruf der Erweiterung in der SOLR-Query

4.3 Erweiterung der Datenstruktur „Interaktion“

Ziel der in diesem Abschnitt thematisierten Optimierung ist es, gleichartige Interaktionen zusammenzuführen. Damit soll die Komplexität bei Datenbankabfragen und die Anzahl der zu speichernden Einträge verringert werden.

Im Abschnitt 4.2.1 wird beschrieben, wie Interaktionen als Typ definiert und automatisch als Datenbanktabelle erstellt werden. Diese Struktur wird um einen Zähler erweitert. Unter dem Tag „<attributes>“ (siehe Abbildung 4.4) wird ein Attribut mit dem Namen „counter“ hinzugefügt. Dieses ist in Abbildung 4.9 dargestellt. Wie die vier bereits angegebenen Eigenschaften ist dieses ein Pflichtfeld und wird persistent in einer Datenbank abgelegt.

```
1 <attribute qualifier="counter" type="java.lang.Integer">
2     <description>Anzahl Interaktionen</description>
3     <persistence type="property"/>
4     <modifiers optional="false"/>
5 </attribute>
```

Abbildung 4.9 - items.xml: Ergänzung eines Zählers

Dieser Zähler kommt an zwei Stellen der Implementierung zum Einsatz. Diese sind das Speichern von Interaktionen und das Auslesen von CTD. Diese werden im Folgenden beschrieben.

4.3.1 Nutzung des Zählers beim Speichern von Interaktionen

Die in diesem Abschnitt beschriebene Implementierung wird in Abbildung 4.10 dargestellt. Sollen Interaktionen in die Datenbank übernommen werden, wird zunächst überprüft, ob die zu speichernde Interaktion bereits vorliegt. Dafür wird nach einem Eintrag mit übereinstimmenden Merkmalen gesucht (siehe Zeilen 4 bis 9). Ist das Element bereits vorhanden, wird dessen Zähler angepasst. Wie in den Zeilen 12 und 13 umgesetzt, ist es möglich, dass mehrere identische Benutzerinteraktionen gleichzeitig übernommen werden. Das bedeutet zum Beispiel, dass zwei Klicks auf ein Produkt für denselben Suchbegriff persistiert werden sollen. In diesem Fall wird der Zähler des Elements um die entsprechende Anzahl erhöht.
Ist das zu speichernde Element noch nicht vorhanden, wird dieses neu angelegt und der Datenbank hinzugefügt. Dabei wird der Zähler auf die Anzahl der zu persistierenden Interaktionen gesetzt. Durch dieses Vorgehen wird gewährleistet, dass jede mögliche Interaktion nur einmal in die Datenbank aufgenommen wird.

```
1  private ProductInteractionModel readOrCreateNewInteractionModel(
2      final Entry<ProductInteractionKey, Integer> setEntry)
3  {
4      final Optional<ProductInteractionModel> entry =
5          productInteractionDao.findInteractionByKeyFields(
6                  setEntry.getKey().getProductCode(),
7                  setEntry.getKey().getSearchTerm(),
8                  setEntry.getKey().getInteractionType(),
9                  setEntry.getKey().getInteractionDate());
10     if (entry.isPresent())
11     {
12         entry.get().setCounter(entry.get().getCounter().intValue() +
13             setEntry.getValue());
14         return entry.get();
15     }
16     return createNewInteractionModel(setEntry);
17 }
```

Abbildung 4.10 - Verwendung des Zählers der Tabelle "ProductInteraction"

4.3.2 Nutzung des Zählers beim Auslesen von Click-Through-Daten

Um die CTR eines Produktes für einen Suchbegriff zu berechnen, wird die Anzahl von Klicks und Impressionen benötigt. Dafür werden die in der Datenbank gespeicherten Interaktionen analysiert. Dabei ist der Zähler der Einträge zu verwenden.

Der Befehl zur Bildung der CTD wird in Abbildung 4.11 dargestellt. Dabei kommt die sogenannte Flexible Search zum Einsatz. Diese ist eine Logik der Plattform SAP-Commerce und ermöglicht die Suche nach Typen, die in der Datei „items.xml“ definiert werden [Fah20a].

Durch den Befehl werden alle Einträge der Tabelle selektiert und nach Produktidentifikation und Suchbegriff gruppiert (siehe Zeile 22). Das ermöglicht die Ermittlung der Anzahl von Impressionen und Klicks. Wie in den Zeilen 2 bis 8 gezeigt, wird die Summe aus den Zählern der Einträge bestimmt, deren Interaktionstyp „Click“ ist. Dies erfolgt auch für Impressionen.

```
SELECT {p.productcode}, {p.searchterm},
   SUM(
      CASE
         WHEN {enum.code} = 'CLICK'
         THEN {p.counter}
         ELSE 0
      END)
   AS clicks,
   SUM(
      CASE
         WHEN {enum.code} = 'IMPRESSION'
         THEN {p.counter}
         ELSE 0
      END)
   AS impressions
FROM
{
   ProductInteraction AS p
   JOIN EnumerationValue AS enum
   ON {p.interactiontype} = {enum.pk}
}
GROUP BY {p.productcode},{p.searchterm}
HAVING clicks > 0 AND impressions > 4
ORDER BY {p.searchterm};
```

Abbildung 4.11 - Flexible Search-Befehl zur Abfrage von Click-Through-Daten

4.4 Puffer für aufgenommene Interaktionen

Um die Anzahl der Interaktionen mit der Datenbank zu verringern, sollen erfasste CTD im Arbeitsspeicher des Servers zwischengespeichert werden. Nach Ansammlung einer definierten Menge erfolgt die Persistierung in der Datenbank. Dabei ist anzumerken, dass zum Beispiel auf dem Produktivsystem des betrachteten Projekts mehrere Threads verwendet werden. Dadurch werden mehrere Suchanfragen, die zur selben Zeit erfolgen, gleichzeitig verarbeitet. Es ist erforderlich, dass die erfassten Interaktionen beider Anfragen korrekt gespeichert werden. Aufgrund der parallelen Abarbeitung ist auf sogenannte Thread-Sicherheit zu achten. Das bedeutet, dass durch Multithreading weder fehlerhaftes Verhalten noch unvorhersehbare Ergebnisse verursacht werden [BS20], S. 367.

4.4.1 Komponente zur Zwischenspeicherung von Interaktionen

```
1  public class DefaultProductInteractionService
2      implements ProductInteractionService {
3
4      protected static final String
5          KEY_CONFIG_BUFFER_SIZE = "interaction.buffer.size";
6      protected static final int
7          DEFAULT_BUFFER_SIZE = 5000;
8      private static final Map<ProductInteractionKey, Integer>
9          BUFFER_MAP = new ConcurrentHashMap<>(DEFAULT_BUFFER_SIZE);
10     private int bufferSize;
11
12     /**/
13
14     public void updateBufferSize()
15     {
16         this.bufferSize = configurationService.getConfiguration()
17             .getInt(KEY_CONFIG_BUFFER_SIZE, DEFAULT_BUFFER_SIZE);
18     }
19 }
```

Abbildung 4.12 - Klasse "DefaultProductInteractionService"

Es wird die in Abbildung 4.12 gezeigte Klasse mit dem Namen „DefaultProductInteractionService“ umgesetzt. Sie beinhaltet eine Variable des Objekttyps Map zur Zwischenspeicherung von Interaktionen (siehe Zeilen 8 und 9). In dieser werden als Schlüssel Objekte der Klasse „ProductInteractionkey“ verwendet. Diese überschreibt die Methode „hashCode“, sodass sich die Hashwerte aus den vier Attributen ergeben, die eine Interaktion charakterisieren. Die vollständige Implementierung dieser Entität ist in Anlage H aufgeführt. Als Wert der Einträge der Map fungiert eine Zählervariable. Die „BUFFER_MAP“, fortlaufend als Puffer bezeichnet, wird als ConcurrentHashMap instantiiert. Bei Operationen auf dieser Instanz werden sowohl Thread-Sicherheit als auch Konsistenz und Atomizität gewährleistet [BS20], S. 382 f.. Weiterhin ist das Attribut „bufferSize“ (siehe Zeile 10) entscheidend. Diese Variable entspricht der Anzahl zwischengespeicherter Interaktionen, ab der die Übernahme in die Datenbank erfolgt. Um diesen Grenzwert zu konfigurieren, wird eine Methode implementiert, die den „ConfigurationService“ nutzt (siehe Zeilen 14 - 18). Dieser ist eine Standardlogik der Plattform SAP Commerce, die Konfigurationen im laufenden Betrieb ermöglicht [Tri21].

4.4.2 Hinzufügen von Interaktionen zum Puffer

```
1   public void addInteraction(final ProductInteractionKey interaction)
2   {
3       BUFFER_MAP.compute(interaction, (key, value) ->
4       {
5           if (Objects.isNull(value))
6           {
7               return 1;
8           }
9           return value + 1;
10      });
11      if (BUFFER_MAP.size() >= bufferSize)
12      {
13          flushToDBAtOnce();
14          // read buffer size again, in case config has changed...
15          updateBufferSize();
16      }
17  }
```

Abbildung 4.13 - DefaultProductInteractionService: Hinzufügen einer Interaktion

Werden auf der Storefront der Online-Präsenz Benutzerinteraktionen erfasst, wird die in Abbildung 4.13 gezeigte Methode aufgerufen. Diese ist für die Zwischenspeicherung der Elemente im Puffer verantwortlich. Es kommt die Methode „compute“ zum Einsatz. Diese überprüft, ob zum gegebenen Schlüssel bereits ein Eintrag in der Map vorhanden ist. Ist dies der Fall, wird der zugehörige Wert um eins inkrementiert. Das bedeutet, dass der Zähler dieses Elements erhöht wird. Für den Fall, dass der Schlüssel nicht existiert, wird dieser zusammen mit dem Wert eins hinzugefügt. Aufgrund der Verwendung der Klasse „ConcurrentHashMap“ erfolgt die Ausführung des Algorithmus Thread-sicher. Auch bei mehreren parallelen Aufrufen werden alle Interaktionen konsistent persistiert.

Es folgt die Überprüfung der Größe des Puffers. Wurde der aktuell konfigurierte Grenzwert erreicht, sollen alle zwischengespeicherten Elemente in die Datenbank übernommen werden. Der entsprechende Algorithmus wird im folgenden Abschnitt thematisiert.

4.4.3 Übernahme des Puffers in die Datenbank

```
1  public void flushToDBAtOnce()
2  {
3      final Set<Entry<ProductInteractionKey, Integer>>
4          toSave = Set.copyOf(BUFFER_MAP.entrySet());
5      BUFFER_MAP.clear();
6      productInteractionDao.addInteractionsToDatabase(
7          toSave.stream().map(this::readOrCreateNewInteractionModel)
8              .collect(Collectors.toList()));
9  }
```

Abbildung 4.14 - DefaultProductInteractionService: Persistieren des Puffer-Inhalts

Die in Abbildung 4.14 dargestellte Methode persistiert aktuell im Puffer vorhandene Elemente. Dabei muss beachtet werden, dass während dieses Prozesses weitere Interaktionen erfasst werden könnten.

Zuerst erfolgt die Erstellung einer Kopie der Map. Wird der Puffer während des Kopiervorgangs angepasst, werden die Änderungen in das sich ergebende Set übernommen [Ora22]. Anschließend wird der Puffer geleert. Die Komponente kann dann bis zum wiederholten Erreichen des Grenzwertes mit Benutzerinteraktionen gefüllt werden. Weiterhin wird über die in der Kopie enthaltenen Elemente iteriert und die Methode „readOrCreateNewInteractionModel“ (siehe Zeile 7) aufgerufen. Sie wird in Abschnitt 4.3.1 beschrieben und in Abbildung 4.10 dargestellt. Die Aufgabe der Methode ist das Zurückgeben in der Datenbank vorhandener Interaktionen oder das Anlegen eines neuen Objektes dieser Klasse, sofern kein Eintrag gefunden worden ist. Dabei wird der Zähler der Elemente erhöht oder initialisiert. Es ergibt sich eine Liste von Interaktionen, die gespeichert werden sollen. Die dafür verwendete Methode (siehe Zeile 6) ist eine Schnittstelle zur Datenbank und wird in Abschnitt 4.2.2 erklärt.

4.4.4 Verhalten des Puffers beim Neustart des Servers

Während eines Deployments oder durch manuelles Eingreifen bei Problemen auf dem Produktivsystem könnte der Server neu gestartet werden. Aufgrund dessen, dass der Inhalt des Puffers erst bei Erreichen des konfigurierten Grenzwertes persistiert wird, würden im RAM zwischengespeicherte Interaktionen gelöscht werden.

Um dies zu verhindern, wird die in Abbildung 4.15 gezeigte Methode implementiert. Auf diese wird die Annotation „@predestroy“ angewandt. Dadurch erfolgt der Aufruf dieser Methode, wenn die Instanz, in der der Puffer definiert ist, entfernt wird. So zum Beispiel beim Herunterfahren des Servers. Unabhängig von der Anzahl zwischengespeicherter Elemente wird die Persistierung in die Datenbank gestartet. Dadurch wird gewährleistet, dass alle erfassten Interaktionen gespeichert werden.
Diese Methode wurde von einem Inhouse Entwickler umgesetzt. Die vollständige Implementierung der Klasse „DefaultProductInteractionService“ wird in Anlage I aufgeführt.

```
1   /**
2   * In pre destroy we ensure, all buffered models are written to
3   database.
4   */
5   @PreDestroy
6   public void preDestroy() {
7       flushToDBAtOnce();
8   }
```

Abbildung 4.15 - DefaultProductInteractionService: Methode "preDestroy"

4.5 Caching von Click-Through-Daten

Um zu verhindern, dass bei Suchanfragen mit demselben Begriff wiederholt Verbindungen zur Datenbank aufgebaut werden müssen, werden Click-Through-Daten in einen Cache aufgenommen. Hierfür kommt eine Bibliothek zum Einsatz[8]. Dieses enthält eine abstrakte Klasse „LoadingCache“. Deren Aufgabe ist es, Elemente automatisch dem Cache hinzuzufügen, wenn diese nicht vorhanden sind.
Die Datenstruktur wird beim ersten Aufruf der Erweiterung, also bei der ersten Suchabfrage nach dem Start des SOLR-Servers, initialisiert. Die entsprechende Methode ist in Anlage J aufgeführt. Es wird definiert, dass die maximale Anzahl von Elementen 1000 und die maximale Zeit, die ein Element gespeichert wird, 20 Minuten beträgt. Entsprechend der Vorgaben des Shopbetreibers könnten diese Werte geändert und an die Auslastung des Systems angepasst werden.

[8] https://mvnrepository.com/artifact/com.google.guava/guava/30.0-jre

4.6 Nutzung einer Datenbanktabelle für Click-Through-Daten

Die Erstellung der Tabelle für berechnete Click-Through-Daten erfolgt entsprechend der Vorgehensweise beim Typ „Interaction“ in der Datei „items.xml“. Die Definition wird in Anlage F gezeigt. Um die erzeugte Tabelle mit CTD zu füllen, wird ein Cronjob implementiert. Dieser ist eine Aufgabe, die automatisiert oder manuell ausgeführt werden kann [ION22]. Die Technik wird vor allem für wiederkehrende Aufgaben verwendet. Durch den Cronjob wird die Datenbanktabelle zunächst geleert. Anschließend erfolgt die Berechnung der CTD anhand der bis zu dem Zeitpunkt erfassten Interaktionen. Die dafür aufgerufene Methode zeigt Abbildung 4.16.

```
1   private void updateClickThroughDataTable()
2   {
3       FlexibleSearchQuery query =
4           new FlexibleSearchQuery(QUERY_CLICK_THROUGH_DATA);
5       [...]
6       final SearchResult<ArrayList<Object>> searchResult =
7           flexibleSearchService.search(query);
8       List<ArrayList<Object>> resultList = searchResult.getResult();
9       ArrayList<ClickThroughDataModel> ctdList = new ArrayList<>();
10      resultList.forEach(element ->
11      {
12          ClickThroughDataModel ctd =
13              modelService.create(ClickThroughDataModel.class);
14          ctd.setProductCode((String) element.get(0));
15          ctd.setSearchTerm((String) element.get(1));
16          ctd.setClicks((Integer) element.get(2));
17          ctd.setImpressions((Integer) element.get(3));
18          ctdList.add(ctd);
19      });
20      modelService.saveAll(ctdList);
21  }
```

Abbildung 4.16 - Aktualisierung der Tabelle für CTD

Es kommt die in Abbildung 4.11 beschriebene Query zum Einsatz (siehe Abschnitt 4.3.2). Mit dieser werden die Daten aus der Tabelle für Interaktionen abgefragt. Beim Selektieren erfolgt eine Gruppierung nach Suchbegriff und Produktidentifikation. Weiterhin erfolgt die Filterung nach Ergebnissen mit einer ausreichenden Anzahl von Klicks und Impressionen. Aus der zurückgegebenen Liste werden CTD gebildet und in die Datenbank übernommen. Die vollständige Implementierung des Cronjobs wird in Anlage K aufgeführt.

5 Testmethodik

5.1 Nutzung von Originaldaten auf einem Testsystem

Durch den Betreiber des betrachteten E-Commerce-Systems ist vorgegeben, dass ausschließlich getestete Software einen Einfluss auf die Suchergebnisse haben darf. Das bedeutet, dass durch den Nutzer keine Änderungen wahrgenommen werden dürfen. Durch das manuelle Pflegen von CTD oder durch Suchabfragen und Klicks auf dem Testsystem werden synthetische Daten gepflegt. Da sie nicht das Verhalten der Kunden widerspiegeln, ist der Test auf Grundlage dieser unzweckmäßig. Um die Nutzung von Originaldaten auf einem Testsystem zu ermöglichen, wird das folgende Vorgehen angewendet.

Zuerst wird der Feature Toggle „ENABLE_CLICK_THROUGH_DATA_COLLECTING“ (siehe Abbildung 4.1) auf dem Produktivsystem aktiviert. Dadurch erfolgt die Erfassung von Interaktionen der Kunden auf der Storefront. Aufgrund der Umsetzung des zweiten Feature Toggle bleibt das Scoring im deaktivierten Zustand unverändert. Daher ist das Sammeln der Daten auf dem Produktivsystem möglich.

Weiterhin sollen die durch das Click-Through-Scoring angepassten Suchergebnisse getestet werden. Hierfür erfolgt die Migration der Originaldaten auf ein Testsystem. Das hierfür verwendete Skript wird in Anlage N aufgeführt. Auf dem System wird auch der zweite Feature Toggle aktiviert und somit das Suchergebnis auf Basis der gesammelten Interaktionen verändert. Mit dieser Vorgehensweise werden durch den Kunden keine Änderungen wahrgenommen, da Anpassungen nur auf einem internen System erfolgen.

5.2 Lösungsansätze zur Evaluierung der Relevanz

5.2.1 Formulierung von User Storys

Eine User Story ist eine Technik zur Formulierung eines Ziels bei der Softwareentwicklung und bietet Vorteile wie eine leichte Verständlichkeit und schnelle Durchführung [t2i17]. Dabei werden Anforderungen aus Sicht des Endbenutzers erklärt und das gewünschte Ergebnis dargestellt [Reh22].

Unter Verwendung dieser Vorgehensweise könnte für einige Suchbegriffe ein selbst definiertes Ziel aufgestellt werden. Dabei wird durch den Autor der vorliegenden Arbeit die gewünschte Anordnung der Suchergebnisse festgelegt. Es wird versucht nachzuvollziehen, welches Ergebnis der Nutzer bei der Verwendung eines bestimmten Terms erwartet. Da dies nicht für alle Suchbegriffe möglich ist, ist eine Stichprobe mit den in der Vergangenheit häufigsten Suchen denkbar. Anschließend erfolgt unter Verwendung der User Story ein Vergleich mit dem ursprünglichen und dem durch das Click-Through-Scoring angepassten Ergebnis. Durch dieses Vorgehen wird erklärt, warum ein Suchergebnis aus Sicht des Kunden als ungeeignet oder passend erachtet werden könnte.

Nachteil dieser Vorgehensweise ist, dass keine exakte Definition des Ziels vorgenommen wird. Grund dafür ist, dass lediglich für ausgewählte Suchbegriffe die Analyse des Ergebnisses erfolgt. Weiterhin würde beim Aufstellen von User Storys durch den Autor der vorliegenden Arbeit eine subjektive Bewertung stattfinden.

5.2.2 Bewertung anhand von Kennzahlen

Eine weitere Möglichkeit eine Software zu bewerten, ist die Verwendung von Kennzahlen. Mit diesen kann der Erfolg durch quantifizierbare Messwerte hergeleitet werden [Gre21]. Zu den Kennzahlen, welche betrachtet werden könnten, gehören:

- Umsatz,
- Gewinn,
- Konversionsrate von Besuchern der Webseite in Bestellungen und
- Kundenbewertungen bezüglich des Einkaufserlebnisses.

Diese Werte zu verbessern, kann für den Shopbetreiber die Motivation sein, eine bestimmte Implementierung vorzunehmen. Daher ist der Vorteil dieser Vorgehensweise, dass auf das Erreichen des eigentlichen Ziels geprüft wird. Weiterhin ist es möglich, den Wert der Implementierung festzustellen, da dieser zum Beispiel eine Umsatzsteigerung von 20% zugesprochen werden kann. Dabei sind Werbekampagnen oder andere Marketing-Aktionen, die einen Einfluss auf die Kennzahlen haben könnten, zu beachten.

Es ist anzumerken, dass dieser Test erst nach Durchführung des Deployments der Erweiterung auf dem Produktivsystem möglich ist. Aufgrund dessen, dass die Ermittlung der Kennzahlen nicht unter Verwendung synthetischer Daten des Testsystems stattfinden kann, wird dieser Lösungsansatz verworfen. Wird durch das stattdessen gewählte Verfahren ein positives Ergebnis festgestellt, könnte mit dieser Variante retrospektiv die Eignung belegt werden.

5.2.3 Evaluierung anhand von Expertenmeinung

Eine weitere Möglichkeit der Evaluierung der Relevanz ist die Bewertung durch Personen, welche für die Online-Präsenz oder für deren Inhalte verantwortlich sind. Dazu gehören Produktpfleger, weitere Angestellte des Unternehmens und Entwickler, die das E-Commerce System betreuen. Diese haben einen maßgeblichen Anteil am Erfolg des Projekts. Aufgrund der langjährigen Erfahrung und umfangreichen Sortimentskenntnisse dieser Personen, wird ihnen die Fähigkeit, möglichst objektiv zwischen geeigneten und unpassenden Suchergebnissen zu unterscheiden, zugesprochen. Sie werden im Folgenden als Experten bezeichnet.

Das Ziel ist es, eine möglichst objektive Evaluierung der Relevanz vorzunehmen. Dabei soll entschieden werden, ob sich die Suchergebnisse qualitativ verbessert haben. Im Vergleich zur Erstellung von User Storys durch eine Person, wird bei der Auswertung verschiedener Expertenmeinungen ein weniger subjektives Ergebnis erwartet. Grund dafür ist, dass mehrere Rückmeldungen beachtet werden.

Im Gegensatz zur Bewertung von Kennzahlen ist die Durchführung dieses Lösungsansatzes auf einem Testsystem realisierbar. Entsprechend des im Abschnitt 5.1 beschriebenen Vorgehens könnte die Erweiterung des Scorings anhand von Originaldaten bewertet werden. Sind auf dem Testsystem dieselben Produktdaten gepflegt, entsprechen die Suchergebnisse den Ergebnissen, die unter Anwendung des Scorings auf dem Produktivsystem ermittelt werden würden. Aufgrund der Durchführbarkeit auf dem Testsystem und der möglichst objektiven Bewertung wird dieser Lösungsansatz gewählt.

Mit Aktivierung beziehungsweise Deaktivierung des Scorings können das vorherige und das angepasste Suchergebnis gegenübergestellt werden. Unter Verwendung dieser Anordnungen soll ein Vergleich stattfinden, welchen die Experten unabhängig voneinander durchführen. Als Methode, um deren Meinungen zu erfassen, wird die Umfrage gewählt.

5.3 Umfrage zur Erfassung von Expertenmeinungen

Mit der Umfrage unter den Experten wird ein Vergleich zwischen den Treffern durchgeführt, die ohne und mit Verwendung von Click-Through-Daten gezeigt werden. Damit soll erfasst werden, ob sich die Qualität des Suchergebnisses verbessert, verschlechtert oder nicht bemerkbar unterscheidet.

In der Umfrage werden die Ergebnisse von fünf Suchabfragen betrachtet. Die Auswahl der zugehörigen Suchbegriffe erfolgt anhand von zwei Kriterien:

Zunächst soll möglichst einfach erkannt werden, welche Treffer erwartet werden. Beispielsweise ist es bei der Eingabe des Markennamens „Kerbl“ nicht möglich, ohne Kontext die Suchintention zu definieren. Die Marke ist ein Hersteller von mehreren unterschiedlichen Produktarten. Dabei können die Experten beispielsweise nicht feststellen, ob ein Desinfektionsmittel oder ein Besen besser zur Abfrage „Kerbl“ passt. Im Gegensatz dazu ist die Erwartung beim Suchbegriff „Besen“ einfacher vorherzusagen. Mit erhöhter Wahrscheinlichkeit wird ein Universalbesen relevanter als eine Kehrmaschine erachtet. Als Zweites ist die bisherige Einschätzung des Suchergebnis entscheidend. Es sollen Begriffe mit einer aktuell schlechten Sortierung analysiert werden. Grund dafür ist, dass für diese eine Verbesserung erfolgen soll. Die Auswahl erfolgt subjektiv durch den Autor der vorliegenden Arbeit.

5.4 Vorgehensweise beim Test nicht-funktionaler Anforderungen

Um einen erfolgreichen Livegang der Erweiterung zu gewährleisten, soll auf die Einhaltung von nicht-funktionalen Anforderungen geprüft werden. Es werden der Ressourcenverbrauch und das Zeitverhalten betrachtet. Folgende Aspekte werden getestet:

- Antwortzeit des SOLR-Servers bei Verwendung der CTD.
- Durch das Persistieren von Interaktionen und CTD erforderlicher Speicherplatz in der Datenbank.
- Benötigte Zeit zur Erfassung von Interaktionen.

Für diese muss entschieden werden, auf welchem System der Test stattfindet.

Die Ermittlung der Antwortzeit des SOLR-Servers, kann nicht in der Produktivumgebung erfolgen. Auf dieser ist der Teil zur Anpassung der Suchergebnisse deaktiviert. Zum Test dieses Aspekts wird ein Testsystem gewählt.

Um die Größe einer Datenbanktabelle festzustellen, können Metadaten verwendet werden, die durch das DBMS MySQL bereitgestellt werden [Wel16]. Da unter Verwendung dieser Methode keine Veränderungen vorgenommen werden, kann die Bestimmung des Speicherbedarfs auf der Produktivumgebung stattfinden. Es ist anzumerken, dass bei diesem Aspekt der benötigte Platz in der Datenbank thematisiert wird. Der Cache und der Puffer werden nicht betrachtet, da deren Speicherbedarf durch eine Begrenzung der enthaltenen Elemente eingeschränkt wird.

Beim Erfassen von Interaktionen erfolgt die Zwischenspeicherung in einen Puffer und bei Erreichen eines konfigurierten Grenzwertes die Übernahme in die Datenbank. Letzteres geschieht nebenläufig und hat aus diesem Grund keinen Einfluss auf die Seitenladezeit. Es wird die Übernahme der Interaktionen in den entsprechenden Puffer betrachtet. Dabei ist anzumerken, dass in diesem Teil Logiken des Standards der Programmiersprache Java verwendet werden (siehe Abschnitt 4.2.2). Daher wird keine bemerkbare Veränderung der Seitenladezeit erwartet. Um dies zu belegen, wird die benötigte Zeit auf dem lokalen System des Autors der vorliegenden Arbeit ermittelt.

6 Evaluierung der Implementierung

6.1 Test der Relevanz der Suchergebnisse

In diesem Abschnitt wird die durch das Click-Through-Scoring erreichte Änderung der Qualität der Suchergebnisse thematisiert. Die Bewertung erfolgt anhand der Meinungen von 17 Experten. In der nachfolgenden Abbildung 6.1 werden die fünf gewählten Suchbegriffe und die Verteilung der zugehörigen Stimmen dargestellt. Es konnte zwischen einem der beiden Suchergebnisse, einer Tendenz zu einer der Anordnungen oder keiner Tendenz gewählt werden. Aufgrund der Verwendung von Originaldaten auf dem Testsystem, entsprechen die gezeigten Suchergebnisse den Anordnungen auf der Produktivumgebung.

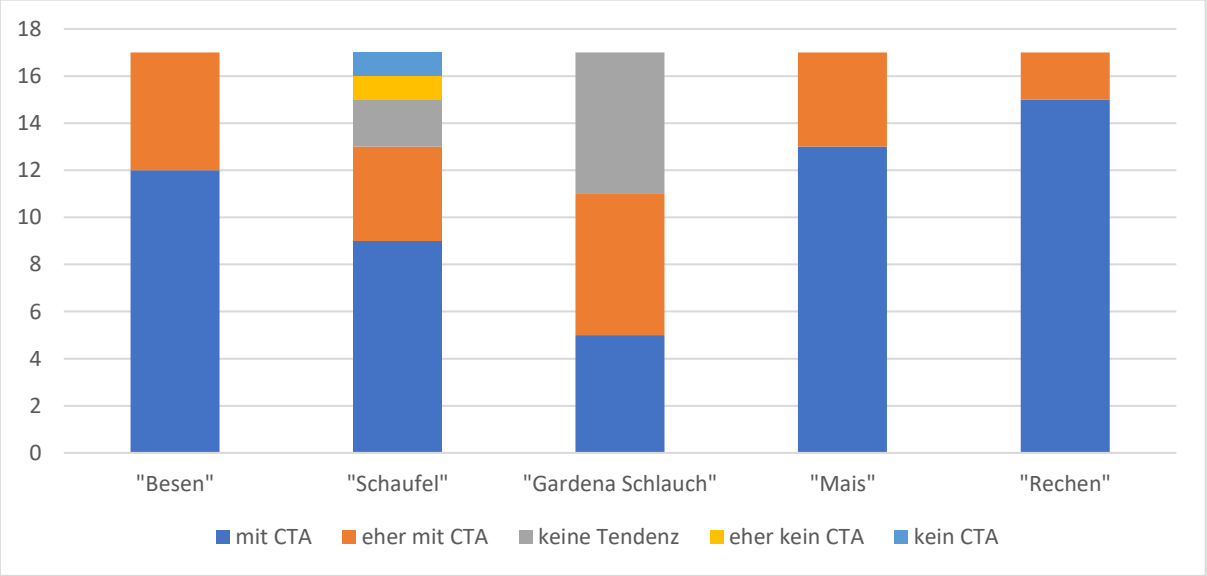

Abbildung 6.1 - Verteilung der Bewertung zur Relevanz nach Suchbegriff

Bei drei Suchbegriffen wird ausschließlich das Suchergebnis unter Verwendung der CTD oder eine Tendenz zu diesem angegeben. „Schaufel“ ist die einzige Abfrage mit Stimmen gegen die Anpassung durch die Erweiterung. Dabei wurde jeweils eine Stimme für das ursprüngliche Ergebnis und für die Tendenz zu diesem angegeben. Der Grund hierfür könnte die im Vergleich zu den anderen Begriffen schwerer erkennbare Suchintention sein. Zu den Treffern gehören sowohl Baggerschaufeln als auch Schaufeln als Gartenwerkzeug. Letztere werden durch den Click-Through-

Algorithmus an vorderster Stelle gezeigt. Dass dies mit 13 von 17 Stimmen der Erwartung der meisten Experten entspricht, zeigt Abbildung 6.1. Der Suchbegriff, bei dem am häufigsten keine Tendenz angegeben wurde, ist „Gardena Schlauch“[9]. Der Grund für dieses Ergebnis könnte das bereits ohne Anpassung angemessenere Suchergebnis sein. Dennoch empfanden 11 Experten die veränderte Anordnung der Treffer als besser.

In Abbildung 6.2 wird die Verteilung aller Stimmen unabhängig vom Suchbegriff gezeigt. Während unter drei Prozent für das ursprüngliche Ergebnis sprechen, erreichten die durch CTD angepassten Suchergebnisse 88 Prozent der Stimmen. Es ist anzumerken, dass im Vergleich zur Größe des Sortiments nur eine kleine Stichprobe von Suchbegriffen beachtet wurde. Um die Funktionsweise anhand von objektiven Kriterien zu belegen, sollte auf dem Produktivsystem ein vollständiger Test erfolgen.

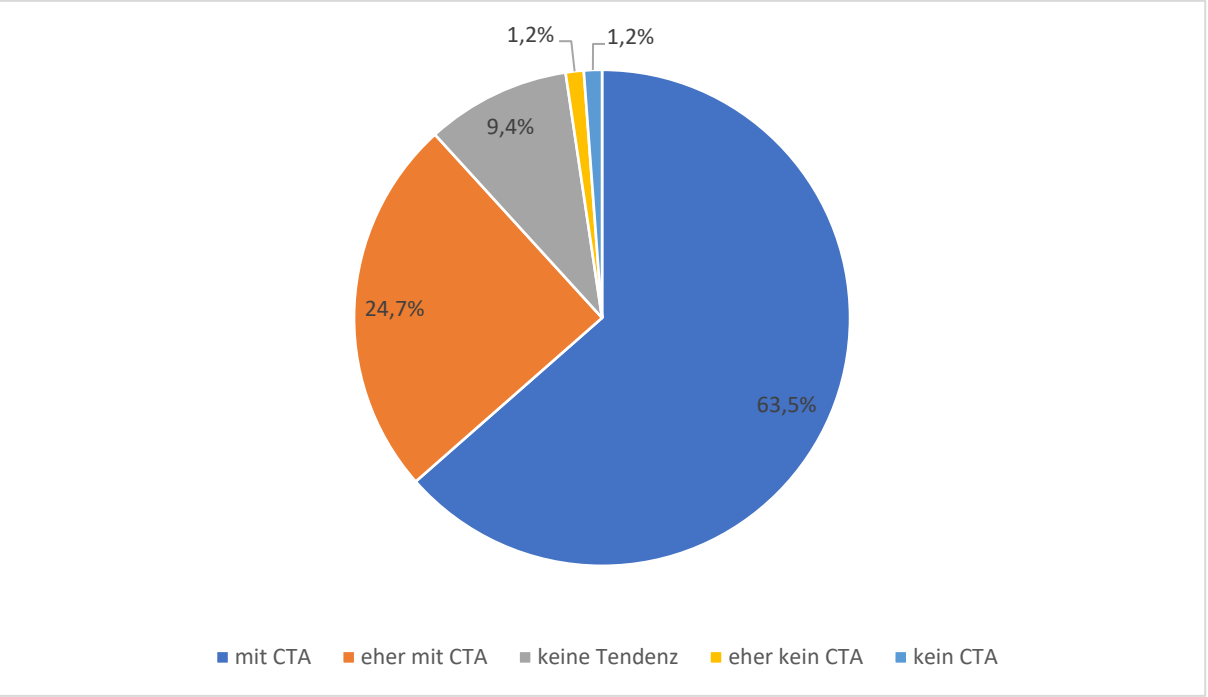

Abbildung 6.2 - Anteilige Verteilung aller Bewertungen

[9] „Gardena“ ist eine Marke für Gartengeräte

6.2 Test der Performance der Erweiterung

In diesem Abschnitt wird thematisiert, welchen Einfluss der Prototyp nach dem Livegang auf die Ressourcen des Produktivsystems hätte. Im Folgenden wird der Speicherplatzbedarf, die Antwortzeit des SOLR-Servers und der Aufwand beim Speichern von Interaktionen festgestellt.

6.2.1 Feststellung des Speicherplatzbedarfs in der Datenbank

Der Speicherplatzbedarf der Datenbanktabellen für Interaktionen und CTD wird mittels SQL-Befehl auf dem Produktivsystem des betrachteten Unternehmens ermittelt. Dabei werden Metadaten verwendet, die das DBMS MySQL bereitstellt. Es kommt der in der folgenden Abbildung dargestellte Befehl zum Einsatz. Das Resultat dieses Befehls beinhaltet den aktuell benötigten Speicherplatz. Dabei wird sowohl die Größe des Indexes als auch der Daten berücksichtigt.

```
1  SELECT
2    TABLE_NAME AS `Table`,
3    ROUND((DATA_LENGTH + INDEX_LENGTH) / 1024 / 1024) AS `Size (MB)`
4  FROM
5    information_schema.TABLES
6  WHERE
7      TABLE_SCHEMA = [database name]
8    AND
9      TABLE_NAME = `productinteraction`
10   OR
11     TABLE_NAME = `clickthroughdata`
```

Abbildung 6.3 - SQL-Befehl zur Ermittlung des Speicherplatzbedarfs

Es ergibt sich ein Speicherplatzbedarf von insgesamt 293 Megabytes bei einem Erfassungszeitraum von 18 Tagen. Es ist anzumerken, dass der Zeitraum durch den Shopbetreiber bestimmt werden kann. Beispielsweise könnten Interaktionen für zwei Wochen als relevant erachtet werden. Bei gleichbleibendem Website Traffic wird der notwendige Speicherplatz somit dauerhaft auf ungefähr 228 Megabytes beschränkt. Das wird erreicht, indem regelmäßig Interaktionen anhand ihres Erfassungszeitpunktes entfernt werden.

Im Vergleich zur Kapazität der Produktivumgebung entspricht der gemessene Speicherbedarf ungefähr 0,1 Prozent. Weiterhin gibt es bereits 27 Datenbanktabellen,

welche einen höheren Bedarf aufweisen. Dazu gehören Bestellungen, Adressen oder Produkte. Aus diesem Grund wird der gemessene Anteil als gering eingeschätzt.

6.2.2 Feststellung der Antwortzeit des SOLR-Servers

Um den Einfluss des Scorings auf die Seitenladezeit festzustellen, wird die Antwortzeit des SOLR-Servers auf dem Testsystem analysiert. Hierfür sind einige Vorbereitungen notwendig. Vor dem Test ist die Migration der Originaldaten des Produktivsystems entsprechend der in Abschnitt 5.1 beschriebenen Vorgehensweise erfolgt. Dadurch wird die Zeit, die für die Abfrage der CTD benötigt wird, nicht durch einen kleineren Datensatz verringert.

Als Zweites wird der Cache des SOLR-Servers für Suchergebnisse deaktiviert. Dadurch wird gewährleistet, dass das Suchergebnis bei jeder Abfrage erneut geladen wird. Andernfalls würde das im Cache vorliegende Ergebnis ohne Verwendung des Rescorings zurückgegeben werden.

Weiterhin ist anzumerken, dass die Antwortzeit in Abhängigkeit zur Suchanfrage und der Anzahl der zugehörigen Treffer steht. Dadurch ist ein Vergleich zwischen verschiedenen Abfragen zwecklos. Aus diesem Grund werden fünf geeignete Suchbegriffe gewählt, die als Stichprobe dienen. Zur Auswahl gehören Suchbegriffe mit:

- vielen erfassten Impressionen,
- vielen erfassten Klicks,
- vielen gefundenen Treffern.

Zur Messung wird eine Standardlogik der Implementierung des SOLRs verwendet. Dabei wird der Wert eines Feldes „elapsedTime“ des Suchergebnisses mit der Antwortzeit gesetzt. Die entsprechende Logik zur Ermittlung der Zeit zeigt Anlage M. Durch Auslesen dieses Feldes ergibt sich die Antwortzeit für die entsprechende Suche. Insgesamt sind 342 Suchabfragen durchgeführt worden. Diese sind in die drei Kategorien:

- Suche ohne Verwendung der CTD (Kategorie 1),
- Suche unter Verwendung der CTD (Kategorie 2) und

- Suche unter Verwendung der CTD mit leerem Cache (Kategorie 3)

unterteilt. Letzteres meint den in Abschnitt 4.5 umgesetzten Cache für Click-Through-Daten. Für die Suchen dieser Kategorie gilt, dass sie die erste Abfrage mit dem Suchbegriff innerhalb der letzten 20 Minuten sind. Nach dieser Zeit werden Elemente aus dem Cache entfernt. Abbildung 6.4 visualisiert das Ergebnis.

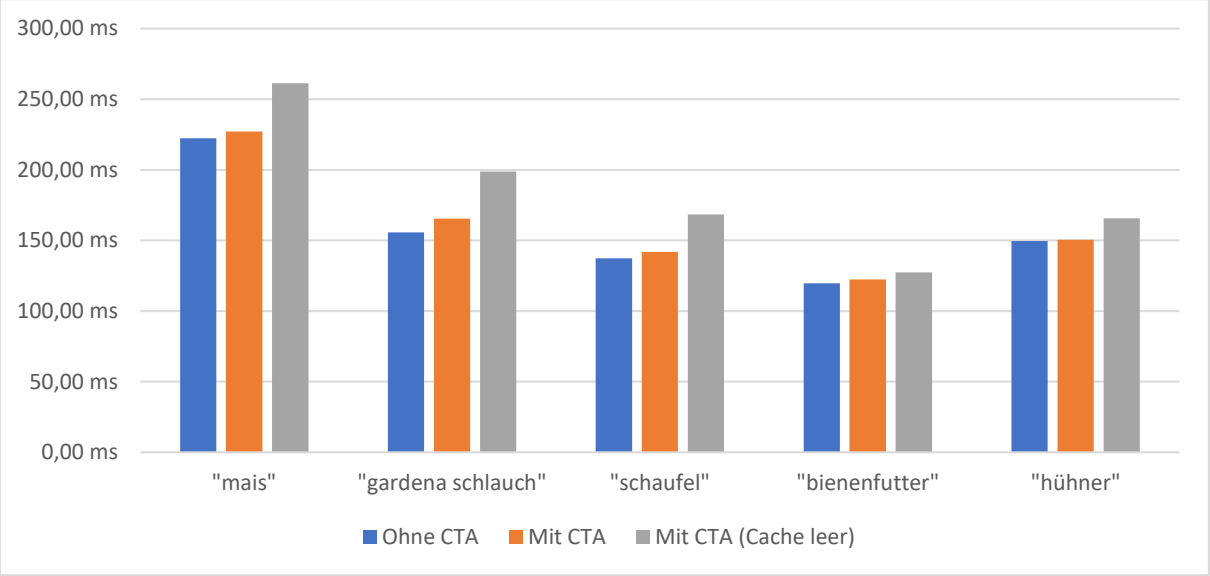

Abbildung 6.4 - Zeitaufwand des Click-Through-Scorings

Für alle Suchbegriffe ist die Abfrage ohne CTD am schnellsten und dauert unter Verwendung des CTA mit leerem Cache am längsten. So dauert die Suche nach „Mais“ im Normalfall durchschnittlich 222,3 Millisekunden. Bei der ersten Abfrage und aktiviertem Feature Toggle für das Click-Through-Scoring beträgt der Wert 261,3 Millisekunden. Es ergeben sich durchschnittliche Differenzen von:

- 4,6 Millisekunden zwischen den Kategorien 1 und 2,
- 27,4 Millisekunden zwischen den Kategorien 1 und 3 und
- 22,8 Millisekunden zwischen den Kategorien 2 und 3.

Der dritten Zeit kann der Aufwand für das Abfragen der CTD aus der Datenbank zugewiesen werden. Da innerhalb von 20 Minuten viele Suchanfragen nur einmal durchgeführt werden, ist der zweite Wert von 27,4 Millisekunden ausschlaggebend.

Diese Zeit ist durch Nutzer nur schwer zu bemerken. Sie entspricht ungefähr 17,5 %[10] der Antwortzeit des SOLR-Servers beziehungsweise 1,4 %[11] der Seitenladezeit. Dieser Aufwand für das Click-Through-Scoring wird als angemessen für die im Abschnitt 6.1 festgestellte Verbesserung der Qualität der Suchergebnisse erachtet. Grund dafür sind der möglicherweise steigende Umsatz und Gewinn des Unternehmens.

6.2.3 Zeitaufwand für das Zwischenspeichern

Beim Messen des Zeitaufwands für das Zwischenspeichern von Interaktionen ist zwischen Impressionen und Klicks zu unterscheiden. Letztere werden beim Aufruf einer Produktdetailseite einzeln in den Puffer übernommen. Im Gegensatz dazu werden beim Laden eines Suchergebnisses mehrere Impressionen gleichzeitig erfasst. Für beide Teile wird auf dem lokalen System des Autors der vorliegenden Arbeit ein Algorithmus umgesetzt, der in Anlage L aufgeführt ist. Dieser bestimmt die Zeit vor und nach Ausführung der Methode, die zur Zwischenspeicherung genutzt wird. Aus der Differenz dieser Werte ergibt sich der Zeitaufwand.

Für beide Teile wird die benötigte Zeit mehrfach gemessen. Dafür werden Suchergebnisse abgefragt und Produktdetailseiten aufgerufen. Die zugrundliegenden Suchen sind voneinander verschieden. Es unterscheidet sich:

- der Suchbegriff,
- die Anzahl der Teilterme,
- die Verwendung von Groß- und Kleinschreibung,
- die Länge der Suchbegriffe und
- die Anzahl der zurückgegeben Treffer.

[10] Referenzwert 156,9 Millisekunden; durchschnittliche Antwortzeit ohne Verwendung von CTD
[11] Referenzwert 2 Sekunden; einmalige Messung mit dem Tool Google Lighthouse

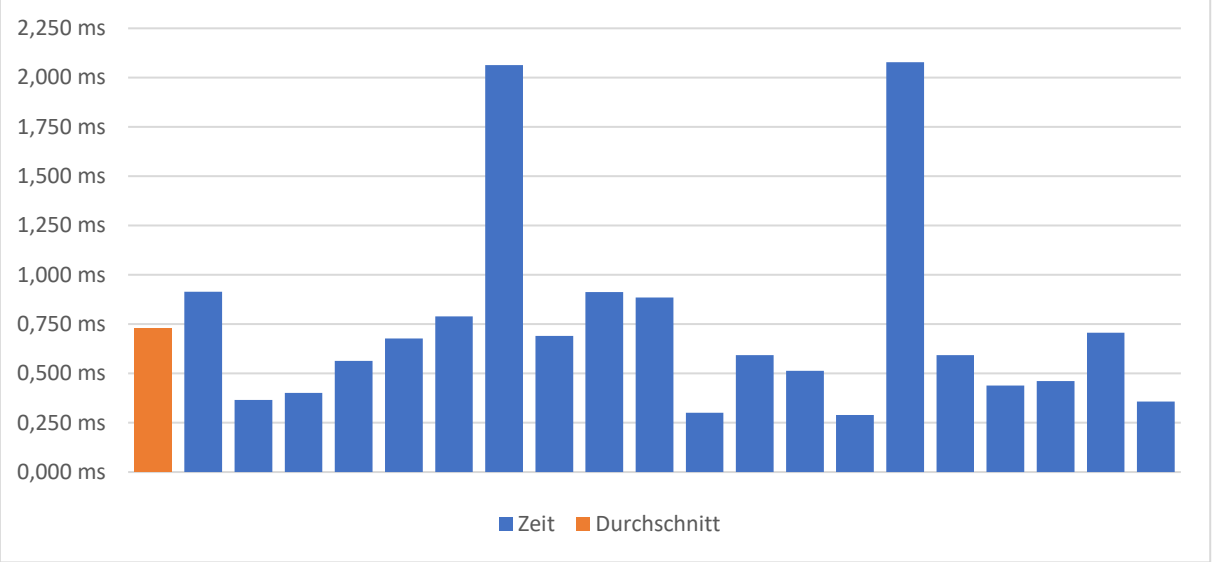

Abbildung 6.5 - Zeitaufwand für das Zwischenspeichern von Impressionen

Abbildung 6.5 visualisiert die Dauer, die für das Zwischenspeichern von Impressionen benötigt wird. Es sind 20 Werte erfasst worden, welche zwischen einem Minimum von 0,3 und einem Maximum von 2,08 Millisekunden liegen. Aus diesen ergibt sich eine durchschnittliche Zeit von ungefähr 0,73 Millisekunden. Unterschiede zwischen den einzelnen Werten sind durch den Ressourcenverbrauch nebenläufiger Prozesse und der Anzahl der hinzuzufügenden Elemente zu begründen. Weiterhin haben bereits im Puffer vorhandene Interaktionen einen Einfluss auf die Zeit. Durch sie wird bestimmt, wie viele Elemente durchsucht werden und ob ein neuer Eintrag angelegt oder lediglich ein Zähler erhöht werden muss.

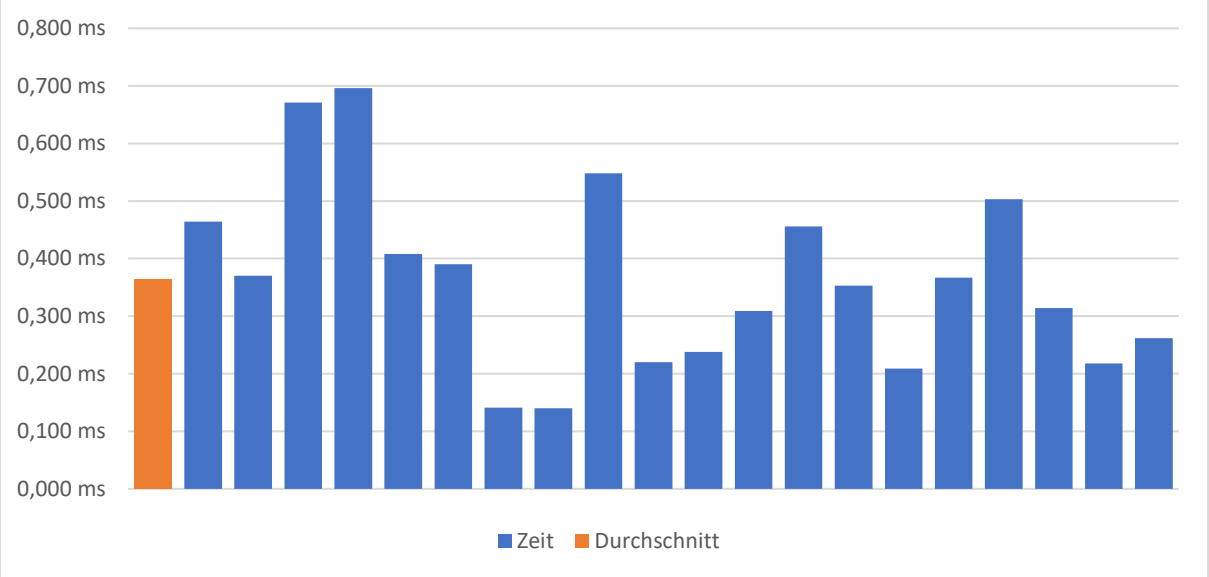

Abbildung 6.6 - Zeitaufwand für das Zwischenspeichern von Klicks

In der Abbildung 6.6 wird gezeigt, wie lange die Übernahme von Klicks in den Puffer dauert. Dabei ist eine durchschnittliche Zeit von ungefähr 0,36 Millisekunden gemessen worden. Im Vergleich zum Durchschnittswert von der Zwischenspeicherung von Impressionen ist diese Dauer geringer. Grund dafür ist, dass jeweils nur eine Interaktion erfasst wird.

Mit den Mittelwerten 0,36 und 0,73 Millisekunden sind für die Zwischenspeicherung von Klicks oder mehreren Impressionen Zeiten unter einer tausendstel Sekunde gemessen worden. Im Vergleich zu einer Seitenladezeit von ungefähr zwei Sekunden[12] auf dem Produktivsystem sind die Werte vernachlässigbar und nicht durch den Nutzer bemerkbar.

[12] Einmalige Messung mit dem Tool Google Lighthouse als Referenzwert

7 Zusammenfassung

Grundlage dieser Bachelorarbeit war ein Prototyp mit dem Klickdaten in das Scoring-Verfahren einer SAP Commerce Plattform einbezogen werden können. Dabei sollen im Suchergebnis häufig geklickte Treffer weiter vorne angezeigt werden. Bei dieser Vorgehensweise werden Benutzerinteraktionen zur Berechnung einer Click-Through-Rate genutzt. Ziel der vorliegenden Arbeit war die Optimierung und Evaluierung dieses Prototyps. Schließlich sollte festgestellt werden, ob ein erfolgreiches Deployment in einer Produktumgebung gewährleistet wird.

Im ersten Arbeitsschritt erfolgte die Analyse der bestehenden Implementierung. Es sind verschiedene Verbesserungspotenziale festgestellt worden. Deren Umsetzung erfolgte in einem weiteren Kapitel. Um flexibel auf Probleme reagieren zu können, wurden zwei Feature Toggle für das Sammeln von Interaktionen und das Anpassen der Suchergebnisse durch das Click-Through-Scoring eingeführt. Zur Steigerung der Performance wurde die Anzahl der Datenbankeinträge verringert. Hierfür erfolgte die Zusammenführung von gleichartigen Interaktionen durch einen Zähler. Weiterhin fand die Integration der Struktur in die Datenhaltung des Systems statt, indem sie als Typ im Kontext der Plattform SAP Commerce definiert worden ist. Dies wurde auch für einen weiteren Typ für Click-Through-Daten vorgenommen. Sie werden aus den Interaktionen gebildet und dienen bei Suchabfragen der Festlegung eines neuen Scores. Durch den Prototyp erfolgte die Berechnung dieser Daten erst bei der Suchanfrage. Die damit verbundene Erhöhung der Seitenladezeit konnte verhindert werden, indem ein Cronjob für deren regelmäßige Berechnung und Persistierung implementiert wurde. Die letzte Optimierung beinhaltet die Umsetzung eines Puffers und eines Cache. Durch sie wird die Anzahl der benötigten Datenbankinteraktionen verringert. Die Aufgabe des Puffers ist die Zwischenspeicherung von erfassten Benutzerinteraktionen, sodass die Übernahme in die Datenbank stoßweise erfolgt. Der Cache erfasst die zu einem Suchbegriff zugehörigen Click-Through-Daten. Bei erneuter Abfrage mit diesem Begriff innerhalb eines bestimmten Zeitraums werden die erfassten Click-Through-Daten nochmals verwendet.

Zur Bewertung der Relevanz der Suchergebnisse ist die Evaluierung durch Experten gewählt worden. Zu diesen Personen gehören Produktpfleger, weitere Angestellte des Unternehmens und Entwickler, die das E-Commerce System betreuen. Sie führten einen Vergleich zwischen den sich ergebenden Suchergebnissen mit den ursprünglichen Anordnungen durch. Für alle betrachteten Suchbegriffe wurde überwiegend das angepasste Ergebnis gewählt, wobei insgesamt nur 2,4 Prozent der Stimmen für die unveränderte Anordnung gesammelt worden sind.

Weiterhin ist der Einfluss auf die Performance und Ressourcen gemessen worden. Zuerst wurde der Speicherbedarf für Interaktionen und Click-Through-Daten in der Datenbank ermittelt. Bei einem Erfassungszeitraum von zwei Wochen liegt der benötigte Speicherplatz bei 228 Megabytes. Aufgrund der Kapazität der Datenbank und der Größe weiterer Tabellen ist dieser Wert als gering eingeschätzt worden. Die Messung der Antwortzeit des SOLR-Servers ergab eine Erhöhung um durchschnittlich 27,4 Millisekunden. Im Vergleich zur Seitenladezeit und ursprünglichen Antwortzeit des SOLR-Servers ist dieser Anteil angemessen für den möglichen Nutzen der Erweiterung. Zuletzt wurde die notwendige Zeit für die Aufnahme von Interaktionen in den Puffer betrachtet. Diese liegt bei unter einer Millisekunde und ist somit vernachlässigbar.

8 Fazit und Ausblick

Entsprechend der Anforderung einen bestehenden Prototyp zu optimieren, wurde zunächst eine Analyse der Implementierung mit anschließender Umsetzung der Verbesserungsmöglichkeiten durchgeführt. Außerdem wurde das Ziel, die Erweiterung zu evaluieren erreicht. Schließlich konnte die Frage, ob eine erfolgreiche Nutzung auf einer Produktivumgebung gewährleistet wird, beantwortet werden. Aufgrund der umgesetzten Sicherheitsmechanismen, der Überprüfung des Einflusses auf die Ressourcen und die Evaluierung der Relevanz wird ein positives Ergebnis erwartet. Im Folgenden werden Kritikpunkte und mögliche weitere Schritte, die nicht Teil dieser Arbeit waren, betrachtet.

Als Kritikpunkt ist anzumerken, dass der Speicherbedarf der Tabelle für Interaktionen verringert werden kann. Durch Anpassung der Definition des Typs, könnten anstelle eines Zählers zwei separate Attribute für die Anzahlen der Klicks und Impressionen eingeführt werden. Somit wird auch bei unterschiedlichen Interaktionsarten kein zusätzlicher Eintrag benötigt und der Speicherplatz gespart.

In Zukunft kann die Implementierung als Grundlage für weitere Entwicklungen genutzt werden. Ein Beispiel ist die Einführung der Konversionsrate als zusätzliches Merkmal im Scoring-Verfahren. Weiterhin muss die Nutzung des Click-Through-Scorings nicht auf Volltextsuchen beschränkt werden. Es ist denkbar, die Entwicklung auf Kategorieseiten oder Produktempfehlungen auszuweiten.

Ein weiterer Punkt ist die Zusammenfassung von Kundenmeinungen. Da bei der Bildung von Click-Through-Daten die durch alle Kunden indizierte Relevanz als eine Klickrate dargestellt wird, erfolgt keine Personalisierung. Beispielsweise ist es möglich, dass Privatpersonen überwiegend andere Produkte als Kunden, die für ein Unternehmen agieren, erwarten. Dennoch wird für beide Gruppen derselbe Datensatz verwendet. Durch die Einführung einer Personalisierung könnte ein besseres Ergebnis erzielt werden.

Aufgrund der Anforderung, auf dem Produktivsystem keine durch den Kunden bemerkbaren Änderungen vorzunehmen, wurde die Evaluierung größtenteils auf Testsystemen durchgeführt. Nach der positiven Bewertung in dieser Arbeit, sollte ein

vollständiger Test auf der Produktumgebung stattfinden. Dadurch kann auf die Bewertung der Relevanz anhand von subjektiven Empfindungen verzichtet und geeignete Kennzahlen verwendet werden. Außerdem ist so die Feststellung des tatsächlichen Nutzens beziehungsweise Wertes der Erweiterung möglich. Mit dem Deployment der Implementierung auf das Produktivsystem, wird ein positiver Einfluss erwartet. Dazu gehört die Steigerung des Umsatzes oder eine erhöhte Kundenzufriedenheit aufgrund des Kauferlebnisses.

V Literaturverzeichnis

[aco22] a coding project GmbH (Hg.): „Arbeiten mit Datenbank Indizes | a coding project“, 23.05.2022, https://www.a-coding-project.de/ratgeber/mysql/arbeiten-mit-datenbank-indizes, Abgerufen am: 23.05.2022.

[Ali18] Aliev, R: „Essential Hybris Services“, 26.07.2018, https://hybrismart.com/2018/07/26/essential-hybris-services/, Abgerufen am: 25.05.2022.

[Apa19a] Apache Software Foundation (Hg.): „Other Schema Elements | Apache Solr Reference Guide 8.4“, 30.12.2019, https://solr.apache.org/guide/8_4/other-schema-elements.html, Abgerufen am: 24.05.2022.

[Apa19b] Apache Software Foundation (Hg.): „SolrRequest (Solr 8.4.0 API)“, 2019, https://solr.apache.org/docs/8_4_0/solr-solrj/org/apache/solr/client/solrj/SolrRequest.html, Abgerufen am: 18.07.2022.

[Aug21] Augsten, S: „Was ist eine Testumgebung?“, 15.06.2021, https://www.dev-insider.de/was-ist-eine-testumgebung-a-1026323/, Abgerufen am: 10.05.2022.

[Beg19] Begerow Beratungsgesellschaft mbH & Co. KG (Hg.): „Datenbank Index - Indizes in einer Datenbank | Datenmodellierung Grundlagen“, 22.12.2019, https://datenbanken-verstehen.de/datenmodellierung/datenbank-index/, Abgerufen am: 23.05.2022.

[Bez22] Jeff Bezos, zit. nach Clarion Tech (Hg.): „Why Digital Transformation Is Easy To Start But Difficult To Finish“, 03.05.2022, https://www.clariontech.com/blog/why-digital-

transformation-is-easy-to-start-but-difficult-to-finish, Abgerufen am: 05.05.2022.

[BS15] Boyarsky, J.; Selikoff, S.: „OCP Oracle Certified Professional Java SE 8 Programmer II Study Guide“, Wiley, o. O., 2015.

[BS20] Boyarsky, J.; Selikoff, S.: „OCP Oracle Certified Professional Java SE 11 Programmer II Study Guide“, 2nd ed., John Wiley & Sons Incorporated, Newark, 2020.

[BSW18] Boron, B; Schreiber, K; Wendt, M: „Fokus auf das Produktivsystem“, 2018, https://www.maibornwolff.de/blog/fokus-auf-das-produktivsystem, Abgerufen am: 10.05.2022.

[Dau22] Daus, K: „Die Vor- und Nachteile von E-Commerce“, 23.03.2022, https://www.lightspeedhq.de/blog/die-vor-und-nachteile-von-e-commerce/, Abgerufen am: 04.05.2022.

[Ebb15] Ebbers, H: „Jede Änderung ein Feature“, 2015, https://entwickler.de/agile/jede-anderung-ein-feature, Abgerufen am: 10.05.2022.

[Ela22] Elastic (Hg.): „Was ist Kibana?“, 12.05.2022, https://www.elastic.co/de/what-is/kibana, Abgerufen am: 12.05.2022.

[Eng22] Engel, D: „Erstellen der Verbindungs-URL - JDBC Driver for SQL Server“, 13.05.2022, https://docs.microsoft.com/de-de/sql/connect/jdbc/building-the-connection-url?view=sql-server-ver15, Abgerufen am: 13.05.2022.

[Erd22] Erdt, P.: „Einbeziehung von Click-Through-Daten im SOLR-Scoring“, Praxisarbeit (IV), Duale Hochschule Gera-Eisenach, Februar 2022.

[Fah20a] Fahri, N: „Flexible Search and Restrictions in SAP Hybris(Commerce)“, 13.06.2020, https://nurayfahri.medium.com/flexible-search-and-restrictions-in-sap-hybris-commerce-57fbf65e8a9a, Abgerufen am: 18.07.2022.

[Fah20b] Fahri, N: „Data Modeling in SAP Commerce(Hybris) - Nuray Fahri - Medium“, 22.11.2020, https://nurayfahri.medium.com/data-modeling-in-sap-commerce-hybris-cc9272fbaf, Abgerufen am: 13.05.2022.

[Fah21] Fahri, N: „Configuration properties and customizing platform code in SAP Commerce(Hybris)“, 06.06.2021, https://nurayfahri.medium.com/configuration-properties-and-customizing-platform-code-in-sap-commerce-hybris-2c5caf52ed63, Abgerufen am: 23.05.2022.

[Goo22] Google Cloud (Hg.): „Datenbankverbindungen verwalten | Cloud SQL for SQL Server | Google Cloud“, 29.04.2022, https://cloud.google.com/sql/docs/sqlserver/manage-connections?hl=de, Abgerufen am: 11.05.2022.

[Gre21] Green, D.: „Die 10 wichtigsten IT-Kennzahlen und KPIs - Apptio“, Apptio, Inc, 25.02.2021, https://www.apptio.com/de/solutions/it-metrics-kpis/.

[ION22] IONOS Digitalguide (Hg.): „CronJob – Aufgaben automatisch ausführen lassen“, 15.07.2022, https://www.ionos.de/digitalguide/hosting/hosting-technik/cronjob/, Abgerufen am: 15.07.2022.

[Lia22] Liang, M: „Understanding Object-Relational Mapping: Pros, Cons, and Types“, 23.05.2022, https://www.altexsoft.com/blog/object-relational-mapping/, Abgerufen am: 23.05.2022.

[Mar18] Marcedo Gesellschaft für Unternehmensverkäufe mbH (Hg.): „Ratgeber: Intelligente Produktsuche in Online-Shops – Wer nichts findet, kauft auch nichts… - Blog für den Onlinehandel“, 21.02.2018, https://www.shopanbieter.de/knowhow-produktsuche-2, Abgerufen am: 04.05.2022.

[OMT21] OMT GmbH (Hg.): „Was ist die Click-Through-Rate (CTR)? | Finde es jetzt raus! | OMT“, 08.07.2021, https://www.omt.de/lexikon/click-through-rate/, Abgerufen am: 16.05.2022.

[Ora22] Oracle (Hg.): „Map (Java Platform SE 8)“, 2022, https://docs.oracle.com/javase/8/docs/api/java/util/Map.html#entrySet--, Abgerufen am: 02.06.2022.

[Pro22] Progress Software Corporation (Hg.): „What Is a JDBC Driver?“, 13.05.2022, https://www.progress.com/faqs/datadirect-jdbc-faqs/what-is-a-jdbc-driver, Abgerufen am: 13.05.2022.

[Raj22] Raj (Hg.): „Configuring Custom database in Hybris | Javainsimpleway“, 23.05.2022, http://javainsimpleway.com/configuring-custom-database-in-hybris/, Abgerufen am: 23.05.2022.

[Reh22] Rehkopf, M: „User Storys | Beispiele und Vorlage | Atlassian“, 10.06.2022, https://www.atlassian.com/de/agile/project-management/user-stories, Abgerufen am: 10.06.2022.

[SAP22a] SAP (Hg.): „ModelService (hybris Commerce Suite 1905)“, 25.05.2022, https://help.sap.com/doc/02d5152884b34821a06408495ba0b771/1905/en-US/de/hybris/platform/servicelayer/model/ModelService.html, Abgerufen am: 25.05.2022.

[SAP22b] SAP (Hg.): „items.xml Element Reference | SAP Help Portal“, 20.05.2022, https://help.sap.com/docs/SAP_COMMERCE/d0224eca81e249cb821f2cdf45a82ace/8bff7a568669101488a5e40cb7bbd0b9.html?locale=en-US&version=1905, Abgerufen am: 23.05.2022.

[SAP22c] SAP (Hg.): „items.xml | SAP Help Portal“, 12.05.2022, https://help.sap.com/docs/SAP_COMMERCE/d0224eca81e249cb821f2cdf45a82ace/8bffa9cc86691014bb70ac2d012708bc.html?version=1905&locale=en-US, Abgerufen am: 13.05.2022.

[SAP22d] SAP (Hg.): „SOLR Score Calculation - SAP Help Portal“, 06.05.2022, https://help.sap.com/docs/SAP_SUCCESSFACTORS_RECRUITING/8477193265ea4172a1dda118505ca631/4f698829534d4787ab31b70b86d89702.html?version=2105&locale=en-US, Abgerufen am: 10.05.2022.

[SAP22e] SAP (Hg.): „Choosing a Database - SAP Help Portal“, 05.05.2022, https://help.sap.com/docs/SAP_COMMERCE/a74589c3a81a4a95bf51d87258c0ab15/388323c93e624b079f9c1bb515979ec1.html?version=1905&locale=en-US, Abgerufen am: 05.05.2022.

[SAP22f] SAP (Hg.): „Database Mapping | SAP Help Portal“, 20.05.2022, https://help.sap.com/docs/SAP_NETWEAVER_703/db5b67e2690e4c0f8254813e9b1e05fb/02c9383eac02561ee10000000a114084.html?version=7.03.29&locale=en-US, Abgerufen am: 23.05.2022.

[SAP22g] SAP (Hg.): „Search and Navigation in SAP Commerce Cloud - An In-depth Series“, 05.05.2022, https://www.sap.com/cxworks/article/2589632416/search_and_navigation_in_sap_commerce_cloud_an_in_depth_series, Abgerufen am: 05.05.2022.

[SEO22] SEO Revolution GmbH (Hg.): „Die Click Through Rate | Definition, Berechnung & Optimierung“, 10.05.2022, https://seo-revolution.com/glossar/click-through-rate, Abgerufen am: 10.05.2022.

[Sha19] Shahapure, N: „Apache Solr: Introduction and Advantages“, 18.12.2019, https://www.xtivia.com/blog/apache-solr-introduction-and-advantages/, Abgerufen am: 05.05.2022.

[Sta22] Statista (Hg.): „E-Commerce - Entwicklung des Umsatzes 2020 | Statista“, 04.05.2022, https://de.statista.com/statistik/daten/studie/3979/umfrage/e-commerce-umsatz-in-deutschland-seit-1999/, Abgerufen am: 04.05.2022.

[t2i17] t2informatik GmbH (Hg.): „Was ist eine User Story? - Wissen kompakt - t2informatik“, 01.11.2017, https://t2informatik.de/wissen-kompakt/user-story/, Abgerufen am: 10.06.2022.

[Tak22] Takso GmbH (Hg.): „Feature Flags - die deutsche Info Seite | Feature Flags“, 16.05.2022, https://featureflags.de/docs/getting-started-with-feature-flags/, Abgerufen am: 16.05.2022.

[Tri21] Trivedi, M.: „We know that the behavior of SAP Hybris to a large extent is cutomizable. One can specify specific value of“, LinkedIn, 17.09.2021, https://www.linkedin.com/pulse/how-configure-read-static-dynamic-properties-sap-hybris-mohit-trivedi.

[Vis21] Visitor Analytics GmbH (Hg.): „Click through Rate (CTR) (Durchklickrate) - Visitor Analytics“, 2021, https://www.visitor-analytics.io/de/glossar/c/click-through-rate-ctr-durchklickrate/, Abgerufen am: 17.05.2022.

[Web19] Webering, J: „So wichtig ist die Seitenladezeit für Online-Shops“, 20.09.2019, https://www.onlinehaendler-news.de/online-handel/praxistipps/131717-wichtig-seitenladezeit-online-shops, Abgerufen am: 11.05.2022.

[Wel16] Welch, A: „How to Get the Size of a Table in MySQL“, 01.01.2016, https://chartio.com/resources/tutorials/how-to-get-the-size-of-a-table-in-mysql/, Abgerufen am: 27.06.2022.

[Yue21] Yuece, H: „Wie Online-Shops die Absprungrate senken“, 2021, https://uptain.de/blog/absprungrate-onlineshop, Abgerufen am: 17.05.2022.

VI Anlagen

A Prototyp: SQL-Befehl zur Erstellung der Tabelle "Interaction"

```
CREATE TABLE Interaction
(
    ID                  int AUTO_INCREMENT,
    interaction_type    varchar(10),
    productID           varchar(20),
    search_term         varchar(255),
    time_of_interaction datetime DEFAULT NOW(),
    PRIMARY KEY (ID)
)
```

B Prototyp: Implementierung des Rescorers

```
package solr.clickthrough.custom;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.apache.lucene.search.*;
import solr.clickthrough.entity.DatabaseConnectionInfo;

import java.io.IOException;
import java.util.Arrays;

public class ClickThroughDataQueryRescorer extends Rescorer
{
    public static final String CODE_STRING = "code_string";
    private static final Logger logger =
        LogManager.getLogger(ClickThroughDataQueryRescorer.class);
    private final double reRankWeight;
    private final ClickThroughDataFetcher clickThroughDataFetcher;

    public ClickThroughDataQueryRescorer(double reRankWeight,
        DatabaseConnectionInfo dbInfo, String queryString)
    {
        logger.debug("Initialising ClickThroughDataQueryRescorer...");
        this.reRankWeight = reRankWeight;
        this.clickThroughDataFetcher =
            new ClickThroughDataFetcher(queryString, dbInfo);
    }

    @Override
    public TopDocs rescore(IndexSearcher searcher, TopDocs
        firstPassTopDocs, int topN) throws IOException
    {
        logger.debug("Rescoring Docs in "+this.getClass()+
            "with topN = "+topN);
        if (clickThroughDataFetcher.isEmpty())
        {
            logger.debug("ClickThroughDataFetcher gets an empty result,
                returning firstPassDocs...");
            return firstPassTopDocs;
        }
        ScoreDoc[] hits = firstPassTopDocs.scoreDocs.clone();
        if(topN != 0 && topN < hits.length)
        {
            //rescore first topN docs
            ScoreDoc[] subset = new ScoreDoc[topN];
            System.arraycopy(hits,0,subset,0,topN);
            rescoreDocs(searcher, subset, clickThroughDataFetcher);
            sortDocsByScore(subset);
            System.arraycopy(subset,0,hits,0,topN);
```

```
        } else
        {
            //rescore all docs
            rescoreDocs(searcher, hits, clickThroughDataFetcher);
            sortDocsByScore(hits); //sort Docs after rescoring
        }
        return new TopDocs(firstPassTopDocs.totalHits, hits);
    }

    @Override
    public Explanation explain(IndexSearcher searcher, Explanation
        firstPassExplanation, int docID) throws IOException
    {
        String documentCode = searcher.doc(docID).get(CODE_STRING);
        double multiplicativeBoost =
            clickThroughDataFetcher.readBoost(documentCode);
        float firstPassScore =
            firstPassExplanation.getValue().floatValue();
        double secondPassScore =
            combine(firstPassScore, multiplicativeBoost);
        Explanation weightExplanation = Explanation.match(reRankWeight,
            "configured weight of click-through-scoring");
        Explanation boostExplanation = Explanation.match(
            multiplicativeBoost, "multiplicative boost, computed as 1 +
                click-through-rate");
        Explanation firstPassScoreExplanation = Explanation.match(
            firstPassScore,
                "firstPassScore, computed as ", firstPassExplanation);
        return Explanation.match(secondPassScore,
            "secondPassScore based on click-through-scoring, computed as
                firstPassScore * boost ^ (weight) from:", boostExplanation,
                    weightExplanation, firstPassScoreExplanation);
    }

    private float combine(float firstPassScore, double boost)
    {
        float score = firstPassScore;
        score *= Math.pow(boost, reRankWeight); //Boost >= 1
        return score;
    }

    private void rescoreDocs(IndexSearcher searcher, ScoreDoc[] hits,
        ClickThroughDataFetcher clickThroughDataFetcher) throws IOException
    {
        for (ScoreDoc currentDoc : hits)
        {
            String documentCode = searcher.doc(currentDoc.doc)
                .get(CODE_STRING);
            double boost = clickThroughDataFetcher.readBoost(documentCode);
            currentDoc.score = combine(currentDoc.score, boost);
        }
```

```
    }

    private void sortDocsByScore(ScoreDoc[] hits)
    {
        Arrays.sort(hits,
                (a, b) -> Float.compare(b.score, a.score));
    }
}
```

C Feature Toggle zur Erfassung von Klicks

```
protected String productDetail(final String productCode,
    final HttpServletRequest request, [...])
{
    [...]
    final String searchTerm = extractSearchTermFromURL(request);
    if (featureToggleService.isFeatureActive(
        Feature.ENABLE_CLICK_THROUGH_DATA_COLLECTING))
    {
        productSolrTrackingFacade.addInteraction(productCode,
            searchTerm, ProductInteractionType.CLICK);
    }
    [...]
}
```

D Feature Toggle zur Anpassung des Suchergebnisses

```
public class AnonymousFacetSearchQueryBoostPopulator
    extends AbstractFacetSearchQueryPopulator
{
    private FeatureToggleService featureToggleService;

    public void populate(SearchQueryConverterData source,
        SolrQuery target)
    {
        [...]
        if(getFeatureToggleService().isFeatureActive(
            AnonymousFeature.ENABLE_CLICK_THROUGH_SCORING)
            && target.get("sort").equals("score desc"))
        {
            target.add("rq", "{!ctrParser}");
        }
    }
    [...]
}
```

E items.xml: Definition des Typs „ProductInteraction“

```
<enumtype code="ProductInteractionType">
    <value code="CLICK"/>
    <value code="IMPRESSION"/>
</enumtype>

<itemtype code="ProductInteraction">
    <description>Product interatction logging entry</description>
    <deployment table="ProductInteraction" typecode="16109"/>
    <attributes>
        <attribute qualifier="productCode" type="java.lang.String">
            <description>Produkt-/Artikelnummer</description>
            <persistence type="property"/>
            <modifiers optional="false" initial="true"/>
        </attribute>
        <attribute qualifier="searchTerm" type="java.lang.String">
             <description>Suchbegriff</description>
             <persistence type="property"/>
             <modifiers optional="false" initial="true"/>
        </attribute>
        <attribute qualifier="interactionType"
            type="ProductInteractionType">
            <description>Art der Interaktion</description>
            <persistence type="property"/>
            <modifiers optional="false" initial="true"/>
        </attribute>
        <attribute qualifier="interactionDate" type="java.util.Date">
            <description>Datum (ohne Uhrzeit) der
                Interaktion</description>
            <persistence type="property"/>
            <modifiers optional="false" initial="true"/>
        </attribute>
    </attributes>
    <indexes>
        <index name="prodSearchTermIDX">
            <key attribute="productCode"/>
            <key attribute="searchTerm"/>
        </index>
        <index name="prodSearchInterDateIDX" unique="true">
            <key attribute="productCode"/>
            <key attribute="searchTerm"/>
            <key attribute="interactionType"/>
            <key attribute="interactionDate"/>
        </index>
    </indexes>
</itemtype>
```

F items.xml: Definition des Typs „ClickThroughData“

```
<itemtype code="ClickThroughData">
    <description>
       Calculated ClickThroughData based on ProductInteraction
    </description>
    <deployment table="ClickThroughData" typecode="16110"/>
    <attributes>
        <attribute qualifier="productCode" type="java.lang.String">
            <description>Produkt-/Artikelnummer</description>
            <persistence type="property"/>
            <modifiers optional="false" initial="true"/>
        </attribute>
        <attribute qualifier="searchTerm" type="java.lang.String">
            <description>Suchbegriff</description>
            <persistence type="property"/>
            <modifiers optional="false" initial="true"/>
        </attribute>
        <attribute qualifier="clicks" type="java.lang.Integer">
            <description>Anzahl Klicks</description>
            <persistence type="property"/>
            <modifiers optional="false" initial="true"/>
        </attribute>
        <attribute qualifier="impressions" type="java.lang.Integer">
            <description>Anzahl Impressionen</description>
            <persistence type="property"/>
            <modifiers optional="false" initial="true"/>
        </attribute>
    </attributes>
    <indexes>
        <index name="prodSearchTermIDX" unique="true">
            <key attribute="productCode"/>
            <key attribute="searchTerm"/>
        </index>
    </indexes>
</itemtype>
```

G Implementierung einer Schnittstelle für Datenbankinteraktionen

```
//imports

public class DefaultProductInteractionDao extends
    DefaultGenericDao<ProductInteractionModel> implements
        ProductInteractionDao {
    //logger
    private static final String PARAM_TIMESTAMP = "timestamp";
    private static final String QUERY_GET_ALL_INTERACTIONS_BEFORE_DATE =
        select(ProductInteractionModel._TYPECODE,ItemModel.PK)
            .from(ProductInteractionModel._TYPECODE)
            .where(Condition.of(ProductInteractionModel._TYPECODE,
                ProductInteractionModel.INTERACTIONDATE,
                COMPARATOR_LESS_OR_EQUAL + param(PARAM_TIMESTAMP)))
                    .toString();

    private ModelService modelService;

    public DefaultProductInteractionDao(final ModelService modelService,
            final FlexibleSearchService flexibleSearchService)
    {
        super(ProductInteractionModel._TYPECODE);
        this.modelService = modelService;
        setFlexibleSearchService(flexibleSearchService);
    }

    @Override
    public Optional<ProductInteractionModel> findInteractionByKeyFields(
        String productCode, String searchTerm, ProductInteractionType
            interactionType, LocalDate interactionDate)
    {
        final Map<String, Object> params = Map.of(
                ProductInteractionModel.PRODUCTCODE, productCode,
                ProductInteractionModel.SEARCHTERM, searchTerm,
                ProductInteractionModel.INTERACTIONTYPE, interactionType,
                ProductInteractionModel.INTERACTIONDATE,
                    DateUtils.localDateToDate(interactionDate));
        return find(params).stream().findAny();
    }

    @Override
    public boolean addInteractionsToDatabase(final
        Collection<ProductInteractionModel> interactions)
    {
        try
        {
            modelService.saveAll(interactions);
            return true;
        }
```

```
        catch (final ModelSavingException e)
        {
            LOG.error(e.getMessage(), e);
            return false;
        }
    }

    @Override
    public boolean deleteByDateIsBefore(final LocalDate timestamp)
    {
        final FlexibleSearchQuery query = new
            FlexibleSearchQuery(QUERY_GET_ALL_INTERACTIONS_BEFORE_DATE);
        query.addQueryParameter(PARAM_TIMESTAMP,
            DateUtils.localDateToDate(timestamp));
        query.setResultClassList(Collections.singletonList(PK.class));
        try
        {
            modelService.removeAll(getFlexibleSearchService().<PK>
                search(query).getResult());
            return true;
        }
        catch (final ModelRemovalException e)
        {
            LOG.error(e.getMessage(), e);
            return false;
        }
    }
}
```

H „ProductInteractionKey“ als Schlüssel des Puffers

```
package de.dotsource.core.services.impl;

import java.time.LocalDate;
import java.util.Objects;

import de.dotsource.core.enums.ProductInteractionType;

/**
 * Product interaction, used as key for caching.
 */
public class ProductInteractionKey {

    private final String productCode;
    private final String searchTerm;
    private final ProductInteractionType interactionType;
    private final LocalDate interactionDate;

    public ProductInteractionKey(final String productCode, final String
        searchTerm, final ProductInteractionType interactionType)
    {
        super();
        this.productCode = productCode;
        this.searchTerm = searchTerm;
        this.interactionType = interactionType;
        interactionDate = LocalDate.now();
    }

    public static ProductInteractionKey of(final String productCode, final
        String searchTerm, final ProductInteractionType interactionType)
    {
        return new ProductInteractionKey(productCode, searchTerm,
            interactionType);
    }

    public String getProductCode()
    {
        return productCode;
    }

    public String getSearchTerm()
    {
        return searchTerm;
    }

    public ProductInteractionType getInteractionType()
    {
        return interactionType;
    }
```

```
    public LocalDate getInteractionDate()
    {
        return interactionDate;
    }

    @Override
    public int hashCode()
    {
        return Objects.hash(interactionDate, interactionType,
            productCode, searchTerm);
    }

    @Override
    public boolean equals(final Object obj)
    {
        if (this == obj)
        {
            return true;
        }
        if ((obj == null) || (getClass() != obj.getClass()))
        {
            return false;
        }
        ProductInteractionKey other =
            (ProductInteractionKey) obj;
        return Objects.equals(interactionDate, other.interactionDate)
            && interactionType == other.interactionType
            && Objects.equals(productCode, other.productCode)
            && Objects.equals(searchTerm, other.searchTerm);
    }
}
```

I Komponente zur Zwischenspeicherung von Interaktionen

```
package de.dotsource.core.services.impl;

import de.hybris.platform.servicelayer.config.ConfigurationService;
import de.hybris.platform.servicelayer.model.ModelService;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Objects;
import java.util.Optional;
import java.util.Set;
import java.util.concurrent.ConcurrentHashMap;
import java.util.stream.Collectors;
import javax.annotation.PreDestroy;
import de.dotsource.core.daos.ProductInteractionDao;
import de.dotsource.core.enums.ProductInteractionType;
import de.dotsource.core.model.ProductInteractionModel;
import de.dotsource.core.services.ProductInteractionService;
import de.dotsource.core.util.DateUtils;

public class DefaultProductInteractionService
    implements ProductInteractionService
{
    protected static final String
       KEY_CONFIG_BUFFER_SIZE = "interaction.buffer.size";
    protected static final int DEFAULT_BUFFER_SIZE = 5000;
    private static final Map<ProductInteractionKey, Integer>
        BUFFER_MAP = new ConcurrentHashMap<>(DEFAULT_BUFFER_SIZE);
    private final ProductInteractionDao productInteractionDao;
    private final ModelService modelService;
    private final ConfigurationService configurationService;
    private int bufferSize;

    public DefaultProductInteractionService(
        final ProductInteractionDao productInteractionDao,
        final ModelService modelService,
        final ConfigurationService configurationService)
    {
        super();
        this.productInteractionDao = productInteractionDao;
        this.modelService = modelService;
        this.configurationService = configurationService;
        bufferSize = DEFAULT_BUFFER_SIZE;
    }

    /**
     * In pre destroy we ensure, all buffered models are written to
       database.
     */
    @PreDestroy
```

```
public void preDestroy() {
    flushToDBAtOnce();
}

@Override
public void addInteraction(
     final String productCode,
     final String searchTerm,
     final ProductInteractionType interactionType)
{
    addInteraction(ProductInteractionKey.of(productCode, searchTerm,
        interactionType));
}

@Override
public void addInteraction(final ProductInteractionKey interaction)
{
    BUFFER_MAP.compute(interaction, (key, value) ->
    {
        if (Objects.isNull(value)) {
            return 1;
        }
        return value + 1;
    });

    if (BUFFER_MAP.size() >= bufferSize)
    {
        flushToDBAtOnce();
        // read buffer size again, in case config has changed...
        updateBufferSize();
    }
}

public void flushToDBAtOnce()
{
    final Set<Entry<ProductInteractionKey, Integer>>
        toSave = Set.copyOf(BUFFER_MAP.entrySet());
    BUFFER_MAP.clear();
    productInteractionDao.addInteractionsToDatabase(
            toSave.stream()
                .map(this::readOrCreateNewInteractionModel)
                .collect(Collectors.toList()));
}

public void updateBufferSize() {
    this.bufferSize = configurationService.getConfiguration()
        .getInt(KEY_CONFIG_BUFFER_SIZE, DEFAULT_BUFFER_SIZE);
}

private ProductInteractionModel readOrCreateNewInteractionModel(
        final Entry<ProductInteractionKey, Integer> setEntry)
```

```
    {
        final Optional<ProductInteractionModel> entry =
            productInteractionDao.findInteractionByKeyFields(
                setEntry.getKey().getProductCode(),
                setEntry.getKey().getSearchTerm(),
                setEntry.getKey().getInteractionType(),
                setEntry.getKey().getInteractionDate());
        if (entry.isPresent())
        {
            entry.get().setCounter(
                entry.get()
                    .getCounter()
                    .intValue()
                     + setEntry.getValue());
            return entry.get();
        }
        return createNewInteractionModel(setEntry);
    }

    private ProductInteractionModel createNewInteractionModel(
        final Entry<ProductInteractionKey, Integer> setEntry)
    {
        final ProductInteractionModel model =
            modelService.create(ProductInteractionModel.class);
        model.setProductCode(setEntry.getKey().getProductCode());
        model.setSearchTerm(setEntry.getKey().getSearchTerm());
        model.setInteractionType(setEntry.getKey().getInteractionType());
        model.setInteractionDate(DateUtils.localDateToDate(
            setEntry.getKey().getInteractionDate()));
        model.setCounter(setEntry.getValue());
        return model;
    }
}
```

J Initialisierung des Caches fürs Click-Through-Daten

```
private static volatile
    LoadingCache<String, HashMap<String, ClickThroughData>>
        clickThroughDataCache;

private static synchronized void initializeClickThroughDataCache()
{
    if (Objects.isNull(clickThroughDataCache))
    {
        clickThroughDataCache = CacheBuilder.newBuilder()
            .maximumSize(1000L)
            .expireAfterWrite(20, TimeUnit.MINUTES)
            .build(new CacheLoader<>()
            {
                @Override
                public HashMap<String, ClickThroughData>
                    load(String searchTerm)
                {
                    return getClickThroughData(searchTerm);
                }
            });
    }
}
```

K Implementierung des Cronjobs zur Aktualisierung der CTD

```
package de.dotsource.aDotsourceCustomer.core.jobs;

import de.dotsource.aDotsourceCustomer.core.model.ClickThroughDataModel;
import de.dotsource.hybris.dsconfiguration.service.DSConfigurationService;
import
de.dotsource.hybris.dsutil.threads.service.DsThreadPoolManagementService;
import de.hybris.platform.cronjob.enums.CronJobResult;
import de.hybris.platform.cronjob.enums.CronJobStatus;
import de.hybris.platform.cronjob.model.CronJobModel;
import de.hybris.platform.impex.constants.ImpExConstants;
import de.hybris.platform.impex.jalo.ImpExManager;
import de.hybris.platform.impex.jalo.ImpExMedia;
import de.hybris.platform.impex.jalo.cronjob.ImpExImportCronJob;
import de.hybris.platform.impex.model.cronjob.ImpExImportCronJobModel;
import de.hybris.platform.servicelayer.cronjob.AbstractJobPerformable;
import de.hybris.platform.servicelayer.cronjob.CronJobService;
import de.hybris.platform.servicelayer.cronjob.PerformResult;
import de.hybris.platform.servicelayer.model.ModelService;
import de.hybris.platform.servicelayer.search.FlexibleSearchQuery;
import de.hybris.platform.servicelayer.search.SearchResult;
import java.io.ByteArrayInputStream;
import java.io.InputStream;
import java.io.UnsupportedEncodingException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;

public class CalculateClickThroughDataJob extends
    AbstractJobPerformable<CronJobModel>
{
    private static final org.apache.logging.log4j.Logger LOG =
        org.apache.logging.log4j.LogManager.getLogger(
            CalculateClickThroughDataJob.class);

    private static final int CLICK_THROUGH_DATA_PAGINATION_RANGE = 1000;
    private static final String CLICK_THROUGH_DATA_THREAD_POOL_SIZE =
        "click.through.data.thread.pool.size";
    private static final int ABORT_POLLING_INTERVAL_MS = 1000;
    private static final String DEFAULT_ENCODING = "UTF-8";
    private static final String IMPEX_EXTENSION = ".impex";
    private static final String CSV_FILE_NAME = "click-through-data.csv";
    private static final String IMPEX_FILE_NAME =
        "cleanup_click_through_data_table_";

    private ImpExManager impexManager;
    private final CronJobService cronJobService;
    private final ModelService modelService;
    private final DSConfigurationService dsConfigurationService;

    private static final String QUERY_CLICK_THROUGH_DATA =
        "SELECT {p.productcode}, {p.searchterm}, " +
```

```
        "sum(CASE WHEN {enum.code} = 'CLICK' THEN {p.counter} ELSE 0 END)
            AS clicks, " +
        "sum(CASE WHEN {enum.code} = 'IMPRESSION' THEN {p.counter} ELSE 0
            END) AS impressions " +
        "FROM {ProductInteraction AS p " +
        "JOIN EnumerationValue AS enum " +
        "ON {p.interactiontype} = {enum.pk} "+
        "} GROUP BY {p.productcode},{p.searchterm} " +
        "HAVING clicks > 0 AND impressions > 4 " +
        "ORDER BY {p.searchterm}";

    public CalculateClickThroughDataJob(
        final ModelService modelService,
        final CronJobService cronJobService,
        final DSConfigurationService dsConfigurationService)
    {
        this.cronJobService = cronJobService;
        this.modelService = modelService;
        this.dsConfigurationService = dsConfigurationService;
    }

    @Override
    public PerformResult perform(final CronJobModel cronJobModel)
    {
        impexManager = ImpExManager.getInstance();
        try
        {
            clearClickThroughDataTable();
            updateClickThroughDataTable();
        }
        catch (Exception e)
        {
            LOG.error(e, e);
            return new PerformResult(CronJobResult.FAILURE,
                CronJobStatus.ABORTED);
        }
        return new PerformResult(CronJobResult.SUCCESS,
            CronJobStatus.FINISHED);
    }

    private void updateClickThroughDataTable()
    {
        FlexibleSearchQuery query = new
            FlexibleSearchQuery(QUERY_CLICK_THROUGH_DATA);
        LOG.debug("Started calculating click-through-data with query = {}",
            QUERY_CLICK_THROUGH_DATA);
        query.setResultClassList(Arrays.asList(String.class, String.class,
            Integer.class, Integer.class));
        int start = 0;
        int range = CLICK_THROUGH_DATA_PAGINATION_RANGE;
        int total;
        query.setCount(range);
        query.setNeedTotal(true);
        do
        {
```

```
            LOG.debug("Processing batch of click-through-data. Starting at
                {} with range of {}", start, range);
            query.setStart(start);
            final SearchResult<ArrayList<Object>> searchResult =
                flexibleSearchService.search(query);
            List<ArrayList<Object>> resultList = searchResult.getResult();
            ArrayList<ClickThroughDataModel> ctdList = new ArrayList<>();
            resultList.forEach(element ->
            {
                ClickThroughDataModel ctd =
                    modelService.create(ClickThroughDataModel.class);
                ctd.setProductCode((String) element.get(0));
                ctd.setSearchTerm((String) element.get(1));
                ctd.setClicks((Integer) element.get(2));
                ctd.setImpressions((Integer) element.get(3));
                ctdList.add(ctd);
            });
            modelService.saveAll(ctdList);
            total = searchResult.getTotalCount();
            start += range;
        }
        while (start < total);
    }

    private void clearClickThroughDataTable() throws
        UnsupportedEncodingException, InterruptedException
    {
        // create stream for header:
        final String impexHeader = "REMOVE
            ClickThroughData[batchmode=true];itemtype(code)[unique=true]\n
            ;ClickThroughData";
        InputStream impexHeaderStream = new
            ByteArrayInputStream(impexHeader.getBytes());

        // create impex media:
        final String impexMediaCode = IMPEX_FILE_NAME +
            System.currentTimeMillis() + IMPEX_EXTENSION;
        final ImpExMedia jobMedia = getImpexManager()
            .createImpExMedia(impexMediaCode, DEFAULT_ENCODING);
        jobMedia.setFieldSeparator(';');
        jobMedia.setData(impexHeaderStream, jobMedia.getCode() +
            IMPEX_EXTENSION, ImpExConstants.File.MIME_TYPE_IMPEX);

        // create cronjob for impex import
        final int threads = getDsConfigurationService()
            .getValue(CLICK_THROUGH_DATA_THREAD_POOL_SIZE,
                Runtime.getRuntime().availableProcessors());
        final ImpExImportCronJob cronJob =
            getImpexManager().createDefaultImpExImportCronJob();
        cronJob.setJobMedia(jobMedia);
        cronJob.setMaxThreads(threads);
        cronJob.setExternalDataCollection(Collections.singletonList(
            getImpexManager().createImpExMedia(CSV_FILE_NAME,
                DEFAULT_ENCODING, impexHeaderStream)));
```

```
        // start cronjob
        final ImpExImportCronJobModel cronJobModel =
            getModelService().toModelLayer(cronJob);
        final Thread importThread =
            DsThreadPoolManagementService.getTenantAwareThread(
                () -> getCronJobService().performCronJob(cronJobModel,
                    true));

        //check status and wait until finished
        for (importThread.start(); importThread.isAlive(); )
        {
            try
            {
                Thread.sleep(ABORT_POLLING_INTERVAL_MS);
            }
            catch (final InterruptedException e)
            {
                // request abort
                getModelService().refresh(cronJobModel);
                cronJobModel.setRequestAbort(true);
                getModelService().save(cronJobModel);
                Thread.currentThread()
                        .interrupt();
                throw e;
            }
        }
    }

    protected ModelService getModelService() {
        return modelService;
    }

    protected DSConfigurationService getDsConfigurationService() {
        return dsConfigurationService;
    }

    protected ImpExManager getImpexManager() {
        return impexManager;
    }

    protected CronJobService getCronJobService() {
        return cronJobService;
    }
}
```

L Algorithmus zur Berechnung der Dauer eines Methodenaufrufs

```
long startTime = System.nanoTime();

[method call]

long endTime = System.nanoTime();

long duration = (endTime - startTime);
    //divide by 1000000 to get milliseconds
```

M Logik zur Ermittlung der Antwortzeit des SOLR-Servers

```
public abstract class SolrRequest<T extends SolrResponse>
    implements Serializable
{
    public final T process(SolrClient client, String collection)
        throws SolrServerException, IOException
    {
        long startNanos = System.nanoTime();
        T res = this.createResponse(client);
        res.setResponse(client.request(this, collection));
        long endNanos = System.nanoTime();
        res.setElapsedTime(TimeUnit.NANOSECONDS.toMillis(
           endNanos - startNanos));
        return res;
    }
}
```

[Apa19b]

N Impex-Script zur Migration von CTD

```
"#% impex.setTargetFile( ""CTD_Export.csv"" );"
insert_update ProductInteraction;
    counter;
    interactionDate[unique=true];
    interactionType(code)[unique=true];
    productCode[unique=true];
    searchTerm[unique=true];
"#% impex.exportItemsFlexibleSearch(""
    SELECT {productinteraction:pk}
    FROM {ProductInteraction AS productinteraction}"" ,
    Collections.EMPTY_MAP, Collections.singletonList( Item.class ), true,
        true, -1, -1 );"
```

dotSource®

Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich,

1. dass ich meine Bachelorarbeit mit dem Thema:

   Optimierung und Evaluierung einer Erweiterung zur Integration von Click-Through-Daten im Apache SOLR Scoring-Algorithmus

   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 und
3. dass ich meine Studienarbeit bei keiner anderen Prüfung vorgelegt habe.

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

Ort, Datum