
Architektur für ein Qualitätsmanagementsystem zur Verbesserung der Relevanz von Suchergebnissen

Masterarbeit im Studiengang Wirtschaftsinformatik (*Master of Science*)
an der Fakultät für Informatik und Ingenieurwissenschaften
der Technischen Hochschule Köln

vorgelegt von: Simon Schneider
eingereicht bei: Prof. Dr. Faeskorn-Woyke
Zweitprüferin: Prof. Dr. Birgit Bertelsmeier

Köln, 24. Januar 2023

Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich während der Anfertigung dieser Masterarbeit unterstützt und motiviert haben.

Zuerst gebührt mein Dank Frau Prof. Dr. Faeskorn-Woyke und Frau Prof. Dr. Birgit Bertelsmeier, die meine Masterarbeit betreut und begutachtet haben. Für die hilfreichen Anregungen und die konstruktive Kritik bei der Erstellung dieser Arbeit möchte ich mich herzlich bedanken.

Ich bedanke mich bei meinem Vorgesetzten bei Brose, Herrn Max Telgkamp, für die Unterstützung und die Möglichkeit, meine Tätigkeit bei Brose flexibel an die Bedürfnisse meines Studium anzupassen.

Ebenfalls möchte ich mich bei meinen Kollegen, Freunden und Kommilitonen bedanken, die mir mit viel Geduld, Interesse und Hilfsbereitschaft zur Seite standen. Ein großes Danke für die zahlreichen interessanten Debatten und Ideen, die maßgeblich dazu beigetragen haben, dass diese Masterarbeit in dieser Form vorliegt, an Lisa Effertz, Fabian Kotschenreuther und Niclas Raabe.

Außerdem möchte ich Verena Dreyer für das Korrekturlesen meiner Masterarbeit danken.

Abschließend möchte ich mich bei meinen Eltern bedanken, die mir mein Studium durch ihre Unterstützung ermöglicht haben und stets ein offenes Ohr für mich hatten.

Köln, 24. Januar 2023

Abstract

Die Suchfunktion ist in vielen Softwareprodukten eine wichtige Komponente, die häufig zur Navigation in der Anwendung dient. Gerade, wenn große Datenmengen bereitgestellt werden, wie es bei Streamingdiensten (Netflix, Spotify) oder bei E-Commerce-Plattformen (Amazon, Zalando) der Fall ist, ist es wichtig, dass die Suchergebnisse für den Nutzer relevant sind. Eine für den Nutzer effektive Navigation mit der Suchfunktion ist nur möglich, wenn die Suchergebnisse eine ausreichend große Relevanz für den Nutzer bieten. Die Organisationen, welche die oben genannten Dienste betreiben, versuchen daher, die Relevanz ihrer Suchergebnisse zu optimieren. Eine Optimierung auf Relevanz ist zwar für eine spezielle Suche einfach, jedoch können dabei häufig Seiteneffekte auftreten, welche die Relevanz über alle Suchen verschlechtern. In einem E-Commerce-Shop kann das Einführen des Synonyms „Birne -> Glühbirne“ dafür sorgen, dass Nutzer, die Glühbirnen kaufen wollen und nach „Birne“ suchen nun auch Glühbirnen finden. Falls Nutzer aber das Obst Birne kaufen möchten, sind die Ergebnisse für diese Gruppe irrelevant. Bei einer Optimierung der Relevanz über alle Suchen können Qualitätsmanagementsysteme unterstützen. Ein Qualitätsmanagementsystem für die Relevanz von Suchergebnissen muss nicht nur fachliche und technische, sondern auch organisatorische Anforderungen beachten, um die Optimierungspotenziale vollständig auszuschöpfen. Diese Arbeit erläutert diese Anforderungen und stellt eine Architektur für ein Qualitätsmanagementsystem vor. Die Architektur wird hinsichtlich der Erfüllung der erläuterten Anforderungen analysiert. Desweiteren werden die Vor- und Nachteile für die jeweiligen Architekturentscheidungen unter Betrachtung der Anforderungen diskutiert. Das Ziel der Arbeit ist es, die Architektur entsprechend zu erläutern, sodass eine Organisation diese für sich angepasst implementieren kann.

Inhaltsverzeichnis

1	Einleitung	1
1.1	Problemstellung	1
1.2	Motivation	1
1.3	Ziel dieser Arbeit	1
2	Organisatorische Voraussetzungen für die Nutzung eines Relevance Quality Management Systems	2
2.1	Einsatzbereiche von Suchfunktionalität in Organisationen	2
2.1.1	Streamingdienst - Netflix	2
2.1.2	Streamingdienst - Spotify	2
2.1.3	E-Commerce - Zalando	3
2.1.4	Dokumentationsplattform - Shopify Support	5
2.2	Relevance Driven Organizations	6
2.2.1	Reifestufen von Organisationen	7
2.2.2	Einordnung der vorgestellten Organisationen	8
2.3	Einführung eines Relevance Quality Management Systems	10
2.3.1	Voraussetzungen	10
2.3.2	Change Management nach Kotter	10
3	Anforderungen an ein Relevance Quality Management System	14
3.1	Allgemeine Anforderungen	14
3.1.1	Definitionen der Domäne	14
3.1.2	Nutzungskontext	15
3.2	Benutzermodellierung	17
3.2.1	Stakeholder	17
3.2.2	User Profiles	19
3.2.3	Personae	22
3.3	Aufgabenmodellierung	29
3.3.1	Szenarien	29
3.3.2	Use Cases	31
3.4	Bestehende Anwendungen	48
3.4.1	Quepid	48
3.4.2	Elastic App Search	49
3.4.3	Lucidworks Connected Search	51
3.5	Technische Schnittstellen	52
3.5.1	Search Engines	52
3.5.2	Nutzerdatenanalyse	52

3.5.3	Erweiterbarkeit durch Plugins	53
4	Architekturvorschlag für ein Relevance Quality Management System	54
4.1	Domain-Driven Design	54
4.1.1	Ubiquitous Language	54
4.1.2	Model-Driven Design	55
4.1.3	Layered Architecture	58
4.1.4	Bounded Context	60
4.2	Ubiquitous Language	60
4.2.1	Domain Model Begriffe	60
4.2.2	Bounded Contexts	62
4.2.3	Organisatorische Begriffe	62
4.2.4	Technische Begriffe	63
4.3	Architekturschichten	64
4.3.1	Präsentationsschicht	65
4.3.2	Anwendungsschicht	65
4.3.3	Domainschicht	66
4.3.4	Infrastrukturschicht	66
4.4	Bounded Contexts - Domainschicht	68
4.4.1	Domain Model - Offline	68
4.4.2	Domain Model - Online	76
4.4.3	Domain Model - Integration	84
5	Referenzimplementierung des Architekturvorschlags	87
5.1	Allgemeines	87
5.1.1	Programmiersprachen	87
5.1.2	Framework	87
5.1.3	Build Tools	88
5.1.4	Bibliotheken	89
5.1.5	Source Code	90
5.1.6	Packaging	90
5.1.7	Deployment	90
5.2	Umsetzung der Architekturschichten	91
5.2.1	Präsentationsschicht	91
5.2.2	Anwendungsschicht	92
5.2.3	Domainschicht	93
5.2.4	Infrastrukturschicht	94
5.3	Umsetzung der Use Cases	95
5.3.1	Technische Use Cases	95
5.3.2	Relevance Test Management	96
5.3.3	Online-Metriken	99
6	Fazit	101
6.1	Allgemein	101

6.2	Technisch	101
6.3	Organisatorisch	102
6.4	Ausblick	102
7	Verzeichnisse	103
7.1	Abkürzungsverzeichnis	103
7.2	Glossar	103
7.3	Abbildungsverzeichnis	110
7.4	Tabellenverzeichnis	111
7.5	Literaturverzeichnis	112

1 Einleitung

1.1 Problemstellung

In einer Welt, in welcher immer mehr Daten zu Verfügung stehen, wird es immer wichtiger, in diesen die relevanten Daten zu finden. Für diese Aufgabe können Suchmaschinen genutzt werden. Eine Suchmaschine zu entwickeln, welche nur relevante Ergebnisse erzeugt, ist sehr herausfordernd[KS22]. Für ein Produkt mit Suchfunktion kann die Steigerung der Relevanz von Suchergebnissen einen erheblichen Anteil der gesamten Entwicklungsressourcen betragen. Wenn ein Produkt Suchergebnisse anzeigen soll, welche aus dynamischen Inhalten generiert werden, ist dieser Aufwand nicht nur während der Entwicklung, sondern auch im Betrieb des Systems vorhanden. Das ist insbesondere in Dokumentationssystemen und im E-Commerce-Bereich der Fall, da sich hier die relevanten Ergebnisse stetig ändern. Wenn im Sommer frische Erdbeeren gefragter sind als im Winter, muss die Suchmaschine die Produkte entsprechend der Jahreszeit sortieren. Diese Abhängigkeit von externen Faktoren führt zu einer großen Dynamik in den Ergebnissen und somit auch zu einem hohen Entwicklungsaufwand, um die hohe Relevanz der Ergebnisse dauerhaft zu gewährleisten.

1.2 Motivation

Organisationen, die sich stark mit der Relevanz von Suchergebnissen beschäftigen, haben häufig verschiedene Rollen und Teams, welche miteinander interagieren müssen, um die Relevanz der Suchergebnisse zu verbessern. Bei der Vereinfachung dieser Prozesse und der Verbesserung der Kommunikation zwischen allen Stakeholdern kann ein Qualitätsmanagementsystem genutzt werden. Dieses System soll zum einen die benötigten Daten liefern, um die Relevanz der Suchergebnisse bewerten zu können und zum anderen die Stakeholder dabei unterstützen, Relevanzverbesserungen am Produkt durchzuführen. Ein Qualitätsmanagementsystem zur Verbesserung der Relevanz von Suchergebnissen wird im Folgenden als Relevance Quality Management System (RQMS) bezeichnet.

1.3 Ziel dieser Arbeit

Diese Arbeit stellt eine mögliche Systemarchitektur für ein RQMS vor. Die vorgestellte Architektur bildet die Prozesse zur Verbesserung der Relevanz von Suchergebnissen ab. Der Architekturvorschlag wird in einer Referenzimplementierung umgesetzt und anhand dieser Implementierung wird die Architektur in Bezug auf ihre Qualität beurteilt. Zu jedem Modul werden die Designentscheidungen erläutert und es wird erklärt, welche Prozesse wie abgebildet wurden. Auf der Grundlage des Architekturvorschlags soll eine Organisation in der Lage sein selbst ein RQMS zu entwickeln und die für die Organisation notwendigen Anpassungen vorzunehmen.

2 Organisatorische Voraussetzungen für die Nutzung eines Relevance Quality Management Systems

2.1 Einsatzbereiche von Suchfunktionalität in Organisationen

Der organisatorische Kontext einer Suchfunktion ist eine wichtige Quelle zur Ermittlung von Anforderungen für ein RQMS. Im Folgenden werden verschiedene Nutzungskontexte von Suchfunktionen beschrieben, welche als Quellen für Anforderungen an ein RQMS dienen. Diese Nutzungskontexte zeigen auf, welche Rolle die Suchfunktion in Organisationen einnehmen kann und welche Geschäftsprozesse damit verbunden sind.

2.1.1 Streamingdienst - Netflix

Netflix ist ein Streamingdienst für Filme und Serien. Das wichtigste Feature der Plattform ist die Empfehlungsfunktion, welche von der Suchfunktion ergänzt wird. Laut Netflix hat die Verbesserung der Suchfunktion dazu geführt, dass weniger Abonnements gekündigt wurden und so \$1 Milliarde mehr Umsatz generiert werden konnte [siehe Col17]. Für die Suchfunktion bei Netflix werden drei Use Cases vorgestellt [siehe LD19, S.1]. Der erste Use Case wird als „fetch“ bezeichnet und beschreibt, dass ein Nutzer einen einzigen, spezifischen Eintrag sucht, indem er eine entsprechende Anfrage formuliert. Als Beispiel wird hier die Anfrage für „Stranger Things“ gezeigt, bei welcher der Nutzer die gleichnamige Serie anschauen möchte. Der zweite Use Case wird als „find“ bezeichnet. Im Gegensatz zum ersten Use Case erwartet der Nutzer hier keinen spezifischen Eintrag, sondern relevante Einträge, die seine Vorstellungen erfüllen. Ein Beispiel hierfür ist die Suche nach dem Namen einer Schauspielerin. Im dritten Use Case, welcher als „explore“ bezeichnet wird, hat der Nutzer nur eine sehr grobe Vorstellung, welche Ergebnisse er erwartet. Dabei sucht der Nutzer zum Beispiel nach einem Genre wie „Horror“. Allgemein gilt, je unspezifischer das erwartete Ergebnis ist, desto wichtiger wird die Empfehlungsfunktion, um dem Nutzer relevante Einträge bereitzustellen. Die Relevanz eines Eintrags kann sehr individuell sein. Wenn ein Nutzer gerade einen Film mit einer Schauspielerin gesehen hat und nach deren Namen sucht, wird er sich wahrscheinlich nicht nochmals denselben Film anschauen. Daher ist es wichtig, diesen individuellen Faktor in die Relevanz einfließen zu lassen.

2.1.2 Streamingdienst - Spotify

Spotify ist ein Musik-Streaming-Dienst, welcher 2006 gegründet wurde [siehe MK22]. Die Spotify-Anwendung stellt den Nutzern eine Suchfunktionalität bereit, mit welcher die Nutzer

nach Titeln, Alben, Titellisten, Genres und weiteren Objekten suchen können. In einer Studie[siehe Hos+19, S. 6] hat Spotify drei Typen von Suchen identifiziert: Bei der fokussierten Suche wissen die Nutzer, welches Objekt sie finden möchten, bevor sie die Suche beginnen. Das ist häufig der Fall, wenn die Nutzer ein bestimmtes Lied oder Lieder eines bestimmten Künstlers hören möchten. Der zweite Typ von Suchen wird als „Offene-Suche“ bezeichnet. Hierbei suchen die Nutzer nach Musik ohne den Wunsch, spezielle Titel oder Künstler hören zu wollen. Diese Suchen werden häufig dazu genutzt um Musik passend zu einer Situation zu finden. Beispiele hierfür sind Workout-Musik, Sommer Hits oder auch Liebeslieder. Der dritte Typ an Suchen wird als „Explorative Suche“ bezeichnet und kommt eher selten vor. Hierbei sucht der Nutzer nach einem Einstieg in ein Thema oder Genre und erwartet weiterführende Ergebnisse. In der Studie wird das Beispiel eines Teilnehmers aufgeführt: „Ich hab mich für Französischen Rap interessiert [...] Und ich kannte den Künstler Stromea. Ich habe einen Titel gefunden und wusste nicht, wie ich von hier aus weiterkomme.“ (Frei übersetzt aus [siehe Hos+19, S. 6]).

Für Spotify ist es wichtig, den Nutzern in allen Suchtypen relevante Ergebnisse zu liefern. Dazu wurde, unter anderem auf Grundlage dieser Studie, die Suche angepasst und um neue Metriken erweitert[siehe Hos+19, S. 10].

2.1.3 E-Commerce - Zalando

Zalando beschreibt sich selbst als digitale Modeplattform für Mode-Marken, Mode-Blogger, Kunden, lokale Geschäfte und alle Gruppen, die mit Mode arbeiten. Dabei vertreibt Zalando nicht nur die eigene Mode, sondern bietet Händlern die Möglichkeit, ihre Produkte über den Zalando Onlineshop zu vertreiben. Das Herzstück des Zalando Onlineshops ist die Suche. Diese wurde nicht als alleinstehende „Search Engine“ entwickelt, sondern als „Search platform“, welche alle anderen Abteilungen bei Zalando nutzen können[EB17, siehe]. Die

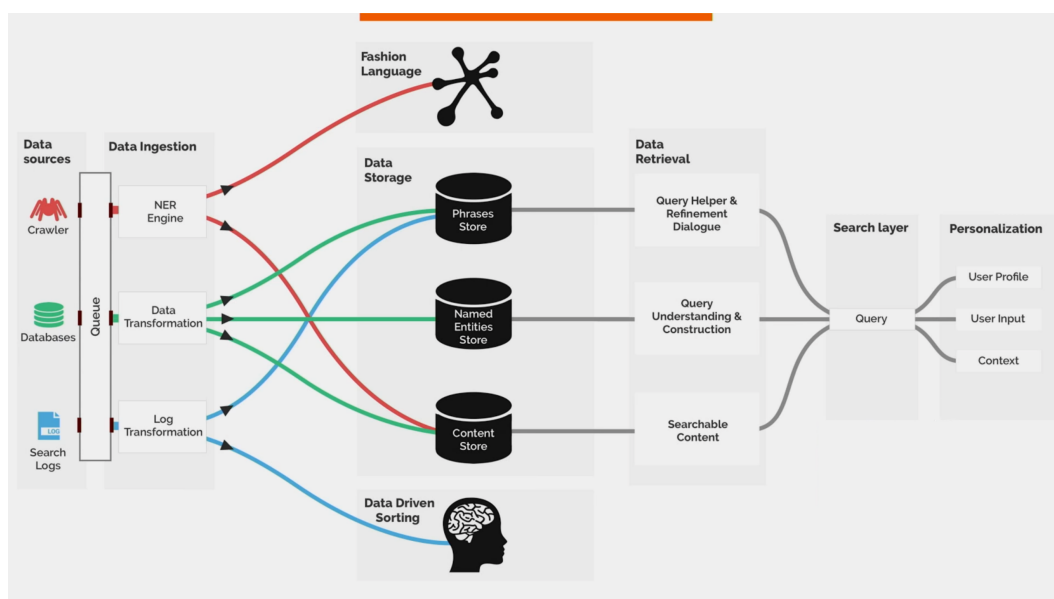


Abbildung 2.1: Search platform Architektur bei Zalando, aus „Berlin Buzzwords 2017“[WW20]

Architektur der „Search platform“ ist in Abbildung 2.1 dargestellt. Diese Architektur ist in mehrere Schichten aufgeteilt, welche von links nach rechts dargestellt sind.

Data sources

Die erste Schicht „Data sources“ stellt alle genutzten Datenquellen dar. Bei Zalando werden neben der Produktdatenbank auch die Logs der Suchanfragen als Datenquellen genutzt. Zusätzlich werden auch Internet-Blogs und andere Mode-Webseiten mit Crawlern eingelesen, um später deren Inhalt als Grundlage zur Erzeugung einer „Fashion Language“ zu nutzen.

Data Ingestion

Die „Data Ingestion“-Schicht bereitet diese Rohdaten auf und persistiert diese in den entsprechenden Datenspeichern. Diese Aufbereitung ist notwendig, damit alle Daten im gleichen Format in den Datenspeichern vorliegen.

Fashion Language

Die dritte Schicht besteht aus mehreren Komponenten. Die „Fashion Language“-Komponente definiert selbstständig eine Sprache, um Gegenstände in der Modewelt zu beschreiben. Ein Beispiel ist die automatische Erstellung von Synonymdefinitionen aus Mode-Blogs, bei welcher für Begriffe wie „Sneaker“ ein Synonym auf den Begriff „Turnschuh“ erstellt wird.

Data Storage

Die „Data Storage“-Komponenten bestehen wiederum aus drei Datenspeichern[siehe EB17]:

1. Der „Phrases Store“ enthält Text-Bausteine, die dem Kunden vorgeschlagen werden, um bessere Suchen zu schreiben.
2. Der „Named Entities Store“ enthält die Relationen zwischen Begriffen und deren Attribut Kategorien. Dabei werden Begriffe wie zum Beispiel „Leder“, der Kategorie „Material“ zugeordnet, oder „Blau“ der Kategorie „Farbe“.
3. Der „Content Store“ enthält die totale Ergebnismenge der Suche.

Data Driven Sorting

Der letzte Bestandteil der dritten Schicht ist das „Data Driven Sorting“. Hierbei wird mithilfe der Logs von Suchanfragen die Sortierung der Suchergebnisse automatisiert angepasst[siehe EB17].

Data Retrieval

Die „Data Retrieval“-Schicht bietet drei Funktionalitäten an. Diese werden bei einer Suchanfrage der Reihe nach ausgeführt, um dem Nutzer den gewünschten Inhalt bereitzustellen. Der „Query Helper & Refinement Dialogue“ soll den Nutzer dabei unterstützen, möglichst spezifische Anfragen zu stellen. Dafür werden dem Kunden in einer Auswahlliste Vorschläge für

die Formulierung der Anfrage präsentiert. Außerdem werden auch persönliche Präferenzen des Kunden miteinbezogen, sodass Markennamen, Farben oder die Kategorie aus den bisherigen Käufen dem Kunden an oberster Stelle präsentiert werden. Die „Query Understanding & Construction“-Funktionalität erstellt aus der Nutzereingabe eine strukturierte Abfrage. Neben den Standardfunktionen, wie der Rechtschreibprüfung, werden dabei die Bestandteile den entsprechenden Kategorien zugeordnet und zueinander gewichtet. Die „Searchable Content“-Funktionalität sucht letztendlich mit der vorab definierten strukturierten Abfrage in der Ergebnismenge und gibt diese nach Relevanz sortiert zurück[siehe EB17].

Search layer & personalization

Die letzten beiden Schichten der Architektur zeigen die Informationen, welche vom Kunden bereitgestellt werden. Dabei sind neben der Anfrage des Kunden auch der Kontext des Kunden und seine bisherigen Interaktionen wichtig, da diese für die Relevanzsortierung und die Unterstützung bei der Formulierung der Anfragen genutzt werden[siehe EB17].

Organisationsstruktur

Die „Search platform“ bei Zalando besteht aus vielen Teams. Für jede Komponente gibt es dedizierte Entwicklungsteams. Außerdem gibt es ein eigenes Architekturteam, welches die Gesamtarchitektur der Suche im Blick behält. Um die „Fashion Language“ umsetzen zu können, gibt es ein eigenes Team, welches sich nur mit dem automatischen Aufbau von Wörterbüchern beschäftigt[siehe EB17]. Daran lässt sich erkennen, dass Zalando im Bereich Suche eine strukturierte Organisation aufgebaut hat[WW20].

2.1.4 Dokumentationsplattform - Shopify Support

Shopify bietet „E-Commerce as a Service“ an. Kleine und mittelständische Händler können dort Onlineshops erstellen[siehe 22b] und ihre Logistik-Prozesse auslagern[siehe 22c]. Um die Fragen der Kunden möglichst einfach beantworten zu können, bietet Shopify eine ausführliche Online-Dokumentation an. Eine wichtige Funktion dieser Dokumentationsplattform ist die Suche. Shopify hat deshalb untersucht, wie sie die Qualität der Suche verbessern können. Dazu wurde eine dreiteilige Strategie entwickelt, mit der die Qualität der Suche gemessen werden sollte[Slo21].

Datensammlung

Für jede Suche wird bei Shopify ein „Search Fact“ erzeugt. Dieser enthält alle Anfragen einer Suche. Falls der Nutzer bei der ersten Anfrage nicht das gewünschte Ergebnis findet, müssen alle weiteren Anfragen einem „Search Fact“ zugeordnet werden, um das Nutzerverhalten vollständig zu erfassen. Außerdem werden alle Interaktionen mit den Suchergebnissen aufgezeichnet. Zusätzlich wird dokumentiert, wenn der Kunde den Support kontaktiert hat. Diese Daten werden über Kafka anderen Systemen zur Verfügung gestellt. Als zweiten Schritt in der Datensammlung markieren Mitarbeiter des Support-Teams die Suchergebnisse für vordefinierte Suchanfragen. Dadurch wird ein qualitativ hochwertiger Datensatz erzeugt, welcher

später zur Verbesserung der Suche genutzt werden kann[Slo21].

Offline-Metriken

Bevor eine verbesserte Suchfunktion den Nutzern zur Verfügung gestellt werden kann, soll bei Shopify sichergestellt werden, dass diese einem festgelegten Qualitätsanspruch genügt. Zur Überprüfung der Qualität von Verbesserungen der Suche werden bei Shopify "Offline-Metriken" verwendet. Dabei werden Suchen auf einer vordefinierten Ergebnismenge durchgeführt und mit den erwarteten Ergebnissen verglichen. Hierzu werden unter anderem die vom Supportteam markierten Suchergebnisse genutzt. Zur Auswertung werden auch die Metriken "Mean Average Precision"[CL06, S. 538] und "Normalized Discounted Cumulative Gain"[Cla+08, S. 663] genutzt. Wenn die Metriken die Grenzwerte überschreiten, kann eine Verbesserung der Suchfunktion für die Kunden freigeschaltet werden[Slo21].

Online-Metriken

Das Nutzerverhalten der Suche wird bei Shopify in Echtzeit überwacht, um herauszufinden, wie oft Nutzer mit den Suchergebnissen interagieren, wie weit die Nutzer auf der Seite scrollen müssen, um das für sie beste Ergebnis zu finden und ob ein Nutzer den Support kontaktieren musste, um sein Problem zu lösen. Neue Funktionalitäten können in Form eines A/B-Tests eingeführt werden und so nur einem Teil der Nutzer zur Verfügung gestellt werden. Die erfassten Metriken beider Gruppen können dann miteinander verglichen werden, um zu entscheiden ob die neue Funktionalität auch für alle Nutzer eine Verbesserung zur bisherigen Funktionalität bietet[Slo21].

Ergebnis

Mithilfe dieser Strategie konnte Shopify für seine Dokumentationsplattform einen neuen Sortierungsalgorithmus entwickeln, welcher den vorherigen Algorithmus in allen Metriken an Leistung übertroffen hat. Hierbei wird auch erwähnt, dass diese Strategie zur Analyse genutzt werden kann, falls neue Funktionalitäten wesentlich schlechter arbeiten als Bisherige. Dann kann eine Segmentierung der Anfragen helfen, um Kundenverhalten besser nachvollziehen zu können und die Funktionalität dann auf jedes Segment einzeln zu optimieren[Slo21].

2.2 Relevance Driven Organizations

Wenn eine Suchfunktion neu entwickelt werden soll, ist die Messung der Relevanz von Suchergebnissen selten eine Anforderung. Daher implementieren die wenigsten Softwareentwickler Relevanzmetriken. Die Anforderung, Metriken zur Messung der Relevanz zu implementieren, muss aus der Organisation kommen[Sch20, S. 12]. Unter anderem deshalb haben die organisatorischen Gegebenheiten einen starken Einfluss auf die Relevanz der Suchergebnisse. Im Folgenden werden die Reifestufen einer Organisation in Bezug auf die Fähigkeit, relevante Suchergebnisse in ihrem Produkt bereitzustellen, erläutert und die bisher vorgestellten Organisationen diesen Reifestufen zugeordnet.

2.2.1 Reifestufen von Organisationen

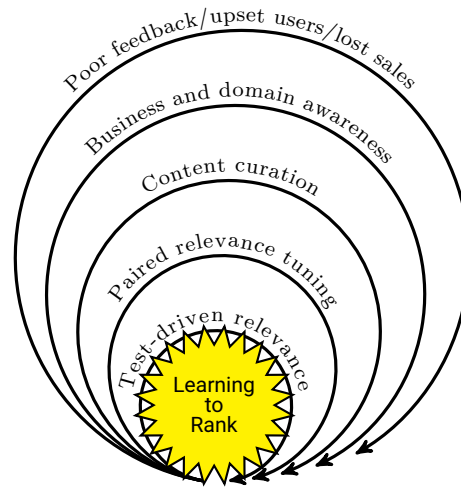


Abbildung 2.2: Learning to rank als Zentrum einer Relevanzgesteuerte Organisation, aus *Relevant search: with applications for Solr and Elasticsearch*, S. 277[TB16]

Die Reife einer Organisation in Bezug auf die Fähigkeit, relevante Suchergebnisse in ihrem Produkt bereitzustellen, lässt sich in sechs Stufen einteilen[siehe TB16, S. 276ff]. Bis auf die erste Stufe sind diese in Abbildung 2.2 dargestellt und werden in den folgenden Abschnitten erläutert. In „Quality assurance for search results“[Sch20, S. 12ff] werden diese Stufen ausführlich beschrieben.

Relevanzblindflug

Eine Organisation, die eine Suchfunktionalität bereitstellt, aber keine Metriken besitzt, um die Relevanz der Suchergebnisse zu beurteilen, kann dieser Stufe zugeordnet werden. Da die Relevanz der Suchergebnisse unbekannt ist, bleiben die sich daraus ergebenden Konsequenzen meistens unentdeckt[siehe TB16, S. 263].

Geschäfts- und Domänenbewusstsein

Für ein Bewusstsein der Geschäftsprozesse und der Domäne wird eine interne Feedbackschleife benötigt, welche einem Relevance Engineer die benötigten Informationen liefert. Die wichtigste Information ist die Antwort der verschiedenen Abteilungen auf die Frage „Was möchte der Nutzer?“. Wenn eine Organisation die interne Feedbackschleife umgesetzt hat und der Relevance Engineer alle Barrieren zwischen den Abteilungen aufgebrochen hat, damit die Informationen von und zwischen den Abteilungen geteilt werden, kann die Organisation dieser Stufe zugeordnet werden[siehe TB16, S. 265].

Kuratieren des Inhalts

Wenn die Informationen bereits zwischen Abteilungen geteilt werden, ist der nächste Schritt die Sicherstellung einer ausreichend hohen Qualität der Suchergebnisse. Dazu kann eine Organisation die Rolle des Content Curators einführen. Ein Content Curator vereinigt die Anforderungen der verschiedenen Abteilungen und definiert, welche Ergebnisse für die Nutzer

relevant sind. Dabei ist die direkte Kommunikation zwischen Content Curator und Relevance Engineer essentiell. Eine Organisation, welche die Rolle des Content Curator eingeführt hat, kann in diese Stufe eingeordnet werden[siehe TB16, S. 268ff].

Paarweise Relevanzoptimierung

In der nächsten Reifestufe arbeiten Content Curator und Relevance Engineer paarweise daran, Relevanzverbesserungen zu implementieren. Ähnlich wie beim Pair Programming im Software Engineering, arbeiten beide Personen gemeinsam an einem Rechner. Dadurch wird die Kommunikation gefördert und die Anforderungen des Content Curator können so spezifisch wie möglich umgesetzt werden[siehe TB16, S. 270ff].

Test-gesteuerte Relevanzoptimierung

Zur Automatisierung der Relevanzoptimierung braucht es Testszenarios. Wenn diese durch den Content Curator und Relevance Engineer aufgebaut wurden, können damit neue Funktionalitäten getestet und bestehende Algorithmen optimiert werden. Die große Herausforderung für Organisationen ist hierbei die stetige Wartung der Testszenarios. Nur wenn diese gepflegt sind, kann sich die Organisation auch auf die Ergebnisse der Tests verlassen. Organisationen, die diese Herausforderungen meistern, können in die Stufe der „Test-gesteuerten Relevanzoptimierung“ eingeordnet werden[siehe TB16, S. 272ff].

Learning to rank

Die höchste Stufe der Relevanzoptimierung ist „Learning to rank“. Dabei wird die Relevanz der Suchergebnisse automatisch optimiert. Diese Optimierung geschieht auf Basis der Daten, welche die Nutzer produzieren. Damit eine Organisation diese Stufe erreichen kann, reicht es nicht aus, den Prozess der Relevanzoptimierung zu automatisieren, sondern alle vorherigen Stufen müssen ebenfalls erfüllt sein. Eine automatische Relevanzoptimierung benötigt Feedbackschleifen, Content Curators, Relevance Engineers und Testszenarien, damit sichergestellt werden kann, dass die automatische Optimierung alle Anforderungen erfüllt. Desweiteren muss auch sichergestellt sein, dass die Relevanz durch automatische Anpassungen in keinem Segment zu stark negativ beeinträchtigt wird[siehe TB16, S. 276ff]. Die Feedback-Kultur der Organisation ist weiterhin ein wichtiger Bestandteil, da nur so alle Anforderungen und Optimierungen implementiert werden können[Sch20, S. 16f].

2.2.2 Einordnung der vorgestellten Organisationen

Die in Abschnitt 2.1 vorgestellten Organisationen können auch in die Reifestufen eingeordnet werden. Die im Folgenden dargestellte Einordnung basiert nur auf den in dieser Arbeit verwendeten Informationen und kann deshalb von der Realität abweichen. Da die Einordnung hier einem exemplarischen Zweck erfüllt, wird bewusst auf eine weitere Untersuchung des Sachverhalts verzichtet.

Netflix

Wie in Unterabschnitt 2.1.1 dargestellt, hat Netflix die Relation zwischen Relevanz der Suchergebnisse und Geschäftszahlen herstellen können. Dazu wurden Metriken benötigt, um die Relevanz der Suchergebnisse zu beurteilen. Da Netflix außerdem die Suchanfragen der Kunden in drei Segmente unterteilt hat, ist es sehr wahrscheinlich, dass Netflix die Rolle des „Content curators“ eingeführt hat. Dieser ist dafür zuständig, solche Segmente zu erkennen und entsprechende Anforderungen zu stellen, die eine auf diese Segmente zugeschnitten Suchfunktion widerspiegeln. Da Netflix auch noch einen sehr hohen Grad der Personalisierung besitzt, ist es sehr wahrscheinlich, dass sie auch „Learning to rank“ nutzen, um die Ergebnisse personalisiert zu optimieren. Daher ist Netflix in die Stufe „Learning to rank“ einzuordnen.

Spotify

Spotify hat, wie in Unterabschnitt 2.1.2 erläutert, die bisherigen Metriken sogar erweitert. Da wie schon bei Netflix drei Kundensegmente identifiziert worden sind, ist es sehr wahrscheinlich, dass Spotify die Rolle des „Content curators“ eingeführt hat. Die Zuordnung von Musikstücken zu gewissen Stimmungen benötigt eine sehr gute Datenqualität, was diese Annahme stärkt, da die Rolle des „Content curators“ auch für die Datenqualität der Suche zuständig ist. Unter dieser Annahme ist Spotify mindestens in die Stufe „Kuratieren des Inhalts“ einzuordnen.

Zalando

Zalando hat im Vergleich zu den anderen vorgestellten Organisation mit Abstand die größte Komplexität in ihrer Suchplattform. Die in Unterabschnitt 2.1.3 vorgestellte Architektur umfasst Komponenten, wie z.B. die „Fashion Language“, die für andere Domänen nicht notwendig sind, aber bei Zalando eine wichtige Rolle bei der automatischen Optimierung der Relevanz einnehmen. Die Metriken werden automatisch aus den Search Logs erzeugt, das Architektur-Team hat die Rolle des „Content curators“ eingenommen, es gibt dedizierte Teams, die sich mit der Datenqualität beschäftigen und außerdem noch mehrere Teams, die an „Data Driven Sorting“ arbeiten, also einer Implementierung von „Learning to rank“. Zalando erfüllt nicht nur die Kriterien einer Organisation, die „Learning to rank“ umsetzt, sondern setzt außerdem Funktionen um, die weit darüber hinausgehen.

Shopify

Für die Messung der Relevanz der Suchergebnisse hat Shopify die Datenklasse „Search Fact“ eingeführt. Mithilfe dieser lassen sich verschiedene Metriken erstellen und auswerten. Die vom Kundensupport kuratierten Suchergebnisse und Anfragen müssen koordiniert werden, daher existierten wahrscheinlich eine oder mehrere Personen, die der Rolle des „Content curators“ zuzuordnen sind. Die Offlinemetriken von Shopify zeigen, dass Test-gesteuerte Relevanzoptimierung genutzt wird. Da „Learning to rank“ nicht erwähnt wurde, ist Shopify wahrscheinlich in die Stufe „Test-gesteuerte Relevanzoptimierung“ einzuordnen.

2.3 Einführung eines Relevance Quality Management Systems

2.3.1 Voraussetzungen

Damit ein RQMS in einer Organisation wertbringend eingesetzt werden kann, müssen einige organisatorische Voraussetzungen gegeben sein. Da ein RQMS auf der Basis von Metriken arbeitet, benötigt die Organisation ein Geschäfts- und Domänenbewusstsein, damit diese Metriken in das RQMS eingepflegt werden können. In automatisierten Tests können diese Metriken dann ausgewertet werden. Um diese Tests durchführen zu können, muss eine Organisation Testdaten bereitstellen. Hierfür wird die Rolle des Content Curators benötigt. Damit die im RQMS eingepflegten Tests auch sinnvoll genutzt werden können, sollte eine Organisation vor der Einführung alle Vorbereitungen treffen, die für die „Testgesteuerte Relevanzoptimierung“ notwendig sind.

Ein RQMS sollte aber nur in Organisationen zum Einsatz kommen, in welchen auch die Notwendigkeit besteht, die Relevanz stetig zu optimieren. Organisationen, für die eine Verbesserung der Relevanz keine positiven Auswirkungen hat, z.B. Gewinnsteigerung oder strategische Vorteile, ist von der Einführung eines RQMS abzuraten, da hier viel Komplexität, Aufwand und somit auch Kosten entstehen. Falls eine Organisation eine statische Ergebnismenge besitzt (z.B. die Produktsuche eines Möbelhauses mit ca. 1000 Artikeln, welche nur einmal im Jahr verändert werden), ist es wahrscheinlich besser, statische Tests z.B. in Form von „Unit-Tests“ zu nutzen, um die Relevanz zu optimieren und diese jährlich anzupassen. In solchen Fällen kann die Einführung eines RQMS mehr Aufwände erzeugen, als sie an Mehrwert bringt.

Organisationen, welche mit sehr großen, dynamischen und mehrsprachigen Ergebnismengen umgehen müssen, können am meisten von der Nutzung eines RQMS profitieren. Durch die hohe Komplexität der Suche kann das RQMS dabei helfen, Seiteneffekte rechtzeitig zu erkennen und zu beseitigen.

2.3.2 Change Management nach Kotter

Da die Einführung neuer Software-Systeme insbesondere dann, wenn diese die Ablauforganisation ändern, häufig mit Widerstand oder Ablehnung einhergeht, ist es wichtig, einen Change Management-Prozess zu nutzen[Kot12, S. 4ff]. Auch die Einführung eines RQMS kann die Ablauforganisation ändern, da die in Abschnitt 2.2 vorgestellten Rollen und Strukturen Voraussetzungen für die Nutzung eines RQMS sind. Damit ein RQMS erfolgreich eingeführt wird, sollte auch hierfür ein Change Management-Prozess genutzt werden. Dieser Aspekt kann auch für die Architektur eines RQMS wichtig sein. Zum Beispiel sollen nach Kotter in einem Change Management-Prozess „short term wins“ erzielt werden[siehe Kot12, S. 125ff]. Das RQMS muss es also ermöglichen „short term wins“ zu erzielen, ohne dabei großen Aufwand zu erzeugen. Anhand Kotters Acht Phasen Modell[Kot12, S. 37ff] wird in den nächsten Abschnitten exemplarisch dargestellt, wie die Einführung eines RQMS ablaufen könnte. Diese Phasen sind in Abbildung 2.3 dargestellt und werden im Folgenden genauer erläutert.



Abbildung 2.3: Acht Phasen Modell nach Kotter, aus *Leading change*, S. 37[Kot12]

Establishing a sense of urgency

Kotters Modell startet in den Change Management-Prozess mit dem Aufbau eines Dringlichkeitsgefühls. In der Organisation soll ein Gefühl dafür existieren, dass Veränderungen notwendig sind[Kot12, S. S37ff]. Um dieses Gefühl im Bezug auf die Relevanz der Suchergebnisse herzustellen, sind Metriken notwendig, mit welchem die Relevanz gemessen werden kann. Außerdem muss eine Relation zwischen Relevanz und Geschäftsmetriken erstellt werden, damit die Konsequenzen fehlender Relevanz-Qualität der Suchergebnisse dargestellt werden können. Die Szenarien, wie es der Organisation geht, wenn die Relevanz-Qualität der Suchergebnisse nicht optimiert wird, sollten diskutiert werden und Möglichkeiten zur Verbesserung der Relevanz identifiziert werden (z.B. Learning to rank oder Personalisierung). Daher muss die Organisation mindestens die Reifestufe „Geschäfts und Domänen Bewusstsein“ erreicht haben, bevor mit dem Aufbau eines Dringlichkeitsgefühls begonnen werden kann.

Creating the guiding coalition

Damit große Veränderungen in Organisationen umgesetzt werden können, empfiehlt Kotter, eine Gruppe zusammenzustellen, welche den Veränderungsprozess anleitet. Diese Gruppe braucht genug Befugnisse und Möglichkeiten, um Veränderungen in der gesamten Organisation anzustoßen. Nach Kotter soll diese Gruppe möglichst eng zusammenarbeiten und Personen aus verschiedenen Bereichen der Organisation beinhalten[Kot12, S. S53ff]. Für die Einführung eines RQMS benötigt die Gruppe Personen aus allen Abteilungen, die Einfluss auf oder Anforderungen an die Suchfunktion haben, also die Stakeholder der Suchfunktion. Häufig gehören hierzu Vertreter der Fachbereiche Software Development, Produktmanagement und Marketing. Da alle Personen eng zusammenarbeiten müssen, sollten schon vor dem Aufbau der „Guiding Coalition“ alle Barrieren zwischen den beteiligten Abteilungen abgebaut sein. Die Reifestufe „Geschäfts- und Domänenbewusstsein“ in einer Organisation muss also erreicht

sein, bevor ein RQMS eingeführt wird.

Developing a vision and strategy

Die Vision, wie eine Organisation nach der Veränderung aussehen soll und eine Strategie, wie diese Vision erreicht werden kann, muss in dieser Phase entwickelt werden [Kot12, S. S69ff]. Für die Einführung eines RQMS bedeutet das, dass die Erwartungen an das System definiert werden und ein Projektplan entwickelt wird, mit welchem das System implementiert werden kann. Wichtig dabei ist, dass die Vision eine realistische Erwartungshaltung aufbaut, da unrealistische Erwartungen zur Ablehnung des Systems führen können.

Communicating the change vision

Für die Kommunikation der Vision soll nach Kotter jeder verfügbare Kommunikationskanal genutzt werden, um ständig die Informationen über Vision und Strategie bei allen Beteiligten präsent zu halten. Dabei nimmt die „Guiding coalition“ die Vorbildfunktion ein [Kot12, S. S87ff]. Damit die Einführung eines RQMS gelingt, sollte die Kommunikation nicht nur Vorträge der Vision und Strategie enthalten, sondern auch Schulungen, Workshops und One-on-One Meetings mit den Mitgliedern der „Guiding coalition“. Diese Mittel sorgen nicht nur für ein Verständnis des RQMS, sondern können auch eine Möglichkeit sein, um Verbesserungen am RQMS vorzunehmen und neue Strategien zum Erreichen der Vision zu finden.

Empowering broad-based action

Diese Phase dient nach Kotter dazu, Hindernisse zu beseitigen, Strukturen zu verändern und Systeme einzuführen, um die entwickelte Vision zu stützen. In dieser Phase sollen bewusst Risiken eingegangen werden und neue Ideen, Aktivitäten und Aktionen gefördert werden [Kot12, S. S105ff]. Diese Phase ist der richtige Zeitpunkt, um das Team zu erweitern und mindestens die Rollen des Relevance Engineers und des Content Curators einzuführen. Das RQMS sollte in dieser Phase implementiert werden, wenn auch vorerst nur als Proof of Concept oder Minimal Viable Product.

Generating short-term wins

Damit in einem lang andauernden Veränderungsprozess keine Trägheit entsteht und auch Vertrauen aufgebaut werden kann, sollen von Anfang an „short-term wins“ eingeplant werden. Allen Personen, die an den „short-term wins“ beteiligt sind, muss Anerkennung zuteilwerden und sie sollten eine Belohnung für ihre Leistung bekommen [Kot12, S. S121ff]. Ein RQMS muss die Möglichkeit bieten diese „short-term wins“ umzusetzen. Eine Relevanzverbesserung sollte dabei weitestgehend ohne domänenspezifische Funktionalität möglich sein. Wenn die Rolle des Content Curators neu eingeführt wurde, kann dieser versuchen, auch ohne ein vollständig implementiertes RQMS, Verbesserungen von domänenspezifischer Funktionalität zu erzielen.

Consolidating gains and producing more change

Nachdem mehr Glaubwürdigkeit gewonnen wurde, kann diese genutzt werden, um alle anderen Systeme, Strukturen, Rollen und Richtlinien an die Veränderungen anzupassen. In dieser Phase ist es wichtig, neue Mitarbeiter anzustellen, sowie Mitarbeiter zu befördern und sie zu entwickeln, damit diese die Vision umsetzen. Der Veränderungsprozess muss in dieser Phase mit neuen Projekten, Themen und Vertretern der Veränderung wiederbelebt werden [Kot12, S. S137ff]. In dieser Phase muss ein RQMS auch schon fest in den Organisationprozessen etabliert sein, sodass neue Ideen zur Verbesserung der Relevanz getestet werden können und Daten über die aktuelle Relevanz allen Beteiligten zur Verfügung stehen.

Anchoring new approaches in the culture

In dieser Phase werden laut Kotter die Relationen zwischen der Veränderung und dem Erfolg der Organisation sichtbar. Es ist die Phase, in welcher die Veränderung zur Normalität wird und die Organisation ihre Leistung stetig durch optimierte Führungsstärke und effektives Management verbessert [Kot12, S. S153ff]. In dieser Phase werden die Ergebnisse der Relevanzoptimierung deutlich sichtbar. Die Organisation hat die Relevanz in allen Segmenten im Blick und kann agieren, falls sich diese verschlechtert. Sowohl Führungskräfte, als auch das Management verstehen die Relevanz und deren Auswirkungen. Sie nutzen die Metriken, um Entscheidungen zu treffen und die Leistung der Organisation insgesamt zu verbessern. Neue Ideen können getestet werden und die Suchfunktionalität kann stetig an neue Anforderungen, Marktsituationen und weitere Veränderungen angepasst werden.

3 Anforderungen an ein Relevance Quality Management System

Das folgende Kapitel orientiert sich für den Systementwurf am Prozess des „User centered design“[siehe Har09b, S. 36f]. Dabei werden als erstes die allgemeinen Anforderungen anhand der Domäne und des Nutzungskontextes[siehe Har09a, S. 9ff] erfasst. Dann werden die Benutzer modelliert[siehe Har09a, S. 21ff] und zuletzt die Aufgaben der jeweiligen Nutzer beschrieben[siehe Har09a, S. 38ff]. Dieses Kapitel enthält außerdem eine kurze Analyse bestehender Anwendungen und die Anforderungen an technische Schnittstellen, da diese für die Arbeit mit einem RQMS von großer Bedeutung sind.

3.1 Allgemeine Anforderungen

3.1.1 Definitionen der Domäne

Qualitätsmanagementsysteme nach ISO Normen

Qualitätsmanagementsysteme werden in vielen verschiedenen Organisationen eingesetzt. Die ISO Norm 9001 [ISO15b, vgl.] definiert Anforderungen an ein Qualitätsmanagementsystem. Organisationen, die nach ISO 9001 zertifiziert werden wollen, müssen ein Qualitätsmanagementsystem umsetzen, welches diesen Anforderungen genügt. Die in ISO 9001 genutzte Definition für Qualitätsmanagementsysteme ist in der ISO Norm 9000 [S. 10f ISO15a, vgl.] erläutert. Ein Qualitätsmanagementsystem wird dort als Zusammenfassung von Tätigkeiten definiert, mit welchen die Organisation ihre Ziele ermittelt, sowie Prozesse und Ressourcen verwaltet, um festgelegte Ergebnisse zu erreichen. Dabei kommt dem Qualitätsmanagementsystem die Aufgabe der Steuerung und Führung von Prozessen und Ressourcen zu, welche genutzt werden, um die Ergebnisse zu erreichen. Ein Qualitätsmanagementsystem ermöglicht der Unternehmensführung, den Ressourceneinsatz zu optimieren, wobei lang- und kurzfristige Folgen berücksichtigt werden. Um diese Folgen zu behandeln, stellt das Qualitätsmanagementsystem auch Mittel zur Verfügung, mit welchen Maßnahmen zur Behandlung identifiziert werden[S. 10f ISO15a, vgl.].

Qualitätsmanagement in der Softwareentwicklung

Das Qualitätsmanagement in der Softwareentwicklung hat das Ziel, die Qualität der Software zu steuern, sodass die Qualitätsanforderungen der Kunden oder des Unternehmens und die zutreffenden rechtlichen Qualitätsanforderungen erfüllt werden. Bei Erfüllung dieser Aufgabe können softwarebasierte Qualitätsmanagementsysteme unterstützen. Diese können neben automatisierten Tests auch Historien abbilden und somit auch aufzeigen, inwieweit sich die

Qualität einer Software verändert hat. Beispiele für solche Systeme sind Codacy [siehe Cod22] und Sonar [siehe Son22]. Einige Systeme erfassen auch Fehler, die während der Ausführung der Software auftreten. Damit kann sichergestellt werden, dass die richtigen Tests zur Qualitätssicherung vorhanden sind. Außerdem ermöglichen diese Systeme, Qualitätsmängel, die nicht in Tests abgedeckt sind, schnellstmöglich zu erkennen. Qualitätsmanagementsysteme in der Softwareentwicklung können nicht nur als Organisationssysteme definiert werden, sondern auch als Anwendungen, welche das Qualitätsmanagement unterstützen. Ein Beispiel für ein solches System ist Sentry [siehe Sen22]. In der Softwareentwicklung werden für die genannten Anwendungen verschiedene Bezeichnungen genutzt. Von „Quality automation platform“ [vgl. Cod22], über „Quality management tool“ [vgl. Son22] bis „Quality management system“ [vgl. Bru22]. Zur Vereinheitlichung werden im Folgenden alle diese Anwendungen als Qualitätsmanagementsystem bezeichnet.

Relevanz-Qualitätsmanagementsystem

In den Abschnitten „Qualitätsmanagementsysteme nach ISO Normen“ und „Qualitätsmanagement in der Softwareentwicklung“ wurde der Begriff Qualitätsmanagementsystem auf unterschiedliche Weisen definiert. Da je nach Anwendung andere Anforderungen gestellt sind, muss der Begriff „Relevanz von Suchergebnissen“ kontextspezifisch definiert werden. Eine solche Definition, welche für viele Kontexte zutrifft, ist, dass ein Suchergebnis dann relevant ist, wenn es den vom Nutzer erwarteten Wert liefert [siehe TB16, S. 4ff]. Daher kann ein RQMS wie folgt definiert werden:

Ein RQMS (Relevance Quality Management System) ist eine Anwendung, welche Mittel zur Verfügung stellt, um die Relevanz der Suchergebnisse in einer Suchfunktion zu verbessern.

Definition 3.1: Relevanz-Qualitätsmanagementsystem - RQMS

3.1.2 Nutzungskontext

Benutzer

Die Liste der Benutzer eines RQMS lässt sich aus den Rollendefinitionen einer „Relevance Driven Organization“ (vgl. Abschnitt 2.2) ableiten. Neben den dort aufgeführten Rollen müssen zusätzlich noch die üblichen Rollen aus der Softwareentwicklung [vgl. SS20, S. 5f] betrachtet werden, da einige davon ebenfalls Nutzer des RQMS sind. Im Folgenden sind alle relevanten Nutzerrollen eines RQMS aufgelistet:

- Relevance Engineer
- Content Curator
- Domain Expert
- Software Engineer
- Agile Coach
- Product Owner
- Manager
- Endnutzer der Suchfunktion (z. B. Kunden in einem Onlineshop)

Aufgaben

Auch die Aufgaben der Benutzer eines RQMS lassen sich aus den Definitionen einer „Relevance Driven Organization“ (vgl. Abschnitt 2.2) ableiten. Eine der wichtigsten Aufgaben ist die Kommunikation von Feedback, sowohl organisationsintern, als auch von Endnutzern der Suchfunktion. Da nur wenige Nutzer aktiv Feedback geben, muss auch das Nutzerverhalten analysiert werden, damit die Suchergebnisrelevanz optimiert werden kann. Das „Paired relevance tuning“ (vgl. Abschnitt 2.2.1) ist eine weitere wichtige Aktivität, welche zwischen Relevance Engineer und Content Curator stattfindet, um die Suchfunktionalität anzupassen. Zudem müssen für automatische Tests sowohl Judgment Lists, als auch Assertion Tests erstellt und regelmäßig gepflegt werden (vgl. Abschnitt 2.2.1).

Arbeitsmittel

Die Nutzung eines RQMS ist stark in die Softwareentwicklung integriert, daher sind im Relevance Engineering und in der Softwareentwicklung die Arbeitsmittel häufig dieselben. Neben analogen Whiteboards werden in der Softwareentwicklung üblicherweise einige der folgenden Software-Systeme als Arbeitsmittel eingesetzt:

- **Ticket Systeme** zur Verwaltung der Aufgaben in einem Team, z. B. JIRA [Atl]
- **Chat Systeme** zur einfachen und direkten Kommunikation in einem Team bzw. der Organisation, z. B. Microsoft Teams [Mic], Slack [Sla]
- **E-Mail Systeme** für die Kommunikation mit anderen Abteilungen aus einer Organisation und externen Personen.
- **Integrierte Entwicklungsumgebungen (IDE)** zur Entwicklung von Software und naheliegenden Funktionalitäten, z. B. IntelliJ [Jet], Eclipse [Gui]
- **Monitoring Systeme** zur Überwachung der Anwendungen, z. B. Grafana [Lab], Prometheus [Pro]
- **Analytics Systeme** zur Analyse des Nutzerverhaltens, z. B. Adobe Analytics [Ado], Azure App Insights [Max]
- **A/B-Test Systeme** zur Durchführung von A/B-Tests mit Nutzern, z. B. Optimizely [Opt]

Umfeld

Das Umfeld für die Nutzung eines RQMS ist dasselbe, wie in der Softwareentwicklung, da beide Aktivitäten in derselben Organisationseinheit stattfinden. In der Softwareentwicklung ist eine häufig genutzte Arbeitsmethode Scrum[siehe SS20]. Diese Arbeitsmethode beschreibt auch das Umfeld, in welchem Softwareentwicklung stattfinden soll. Viele agile Arbeitsmethoden in der Softwareentwicklung weisen große Überschneidungen mit Scrum auf[Pol18b]. Populäre Beispiele dafür sind z. B. Kanban oder Extreme Programming[Pol18b]. Daher wird in der weiteren Betrachtung nur Scrum als Methode berücksichtigt. Es wird für diese Arbeit

angenommen, dass ein Großteil der Anforderungen aus Scrum abgeleitet werden kann.

Der zentrale Bestandteil von Scrum ist das Scrum Team. Es besteht aus einer kleinen Gruppe an Personen. Pro Scrum Team gibt es einen Scrum Master, einen Product Owner und mehrere Entwickler. Die Größe des Teams sollte in der Regel zehn Personen nicht überschreiten[SS20, S. 5]. Ein Relevance Engineer und ein Content Curator können ebenfalls in ein Scrum Team eingeordnet werden. Eine Anforderung an Scrum Teams ist, dass sie „cross-functional“ arbeiten. Das bedeutet, dass ein Scrum Team alle Fähigkeiten besitzt, um Mehrwert in einem Sprint zu erzeugen[SS20, S. 5]. Der Mehrwert ist dabei vom Team zu definieren, für ein Entwicklungsteam im E-Commerce kann der Mehrwert zum Beispiel darin bestehen, den Kunden einfacher die Produkte zu zeigen, welche sie kaufen möchten und dadurch mehr Umsatz zu generieren. Gerade für die Zusammenarbeit zwischen Softwareentwicklern, Relevance Engineers und Content Curators ist es sinnvoll, dass alle Rollen in einem Team vertreten sind. Somit können in einem Sprint Relevanzverbesserungen an der Suche umgesetzt und den Endnutzern der Suchfunktion zur Verfügung gestellt werden. Der Product Owner ist im Scrum Team verantwortlich für die Erzeugung von Mehrwert im Produkt[SS20, S. 5f]. Im Kontext der Relevanzverbesserungen kann das auch bedeuten, dass der Product Owner die Verantwortung dafür trägt, dass die Relevanz der Suche den vorgegebenen Qualitätsanforderungen genügt. Damit keine Konflikte zwischen Content Curator und Product Owner entstehen, muss eine klare Trennung der Aufgaben und Verantwortlichkeiten erfolgen. In einem Scrum Team hat der Product Owner die endgültige Entscheidungsgewalt. Das heißt, er darf auch Entscheidungen gegen den Willen des Content Curators treffen, auch wenn diese die Relevanz der Suchfunktion verschlechtern[SS20, S. 6]. In den meisten Fällen sollten beide aber zu einer einvernehmlichen Lösung kommen.

3.2 Benutzermodellierung

3.2.1 Stakeholder

Im Prozess des „User centered design“[siehe Har09b, S. 36f] ist die Identifizierung von Stakeholdern[siehe Har09a, S. 22] die Grundlage für die Entwicklung von User Profiles, Personae und Szenarien. „[Stakeholder sind Personen(-gruppen), die] Interesse in dem Entwicklungsprozess bzw. dessen Ergebnissen haben bzw. dadurch in irgendeiner Weise berührt sind“[aus Har09a, S. 26]. Für ein RQMS sind im Folgendem alle identifizierten Stakeholder aufgelistet und beschrieben. Einige Stakeholder lassen sich aus den Definitionen einer „Relevance Driven Organization“ (vgl. Abschnitt 2.2) ableiten.

- Endnutzer (der Suchfunktion)
- Relevance Engineer
- Content Curator
- Domain Expert
- Software Engineer
- Agile Coach
- Product Owner
- Manager
- IT-Security
- IT-Administrator
- Controlling

Damit nur die für die Entwicklung relevanten Stakeholder betrachtet werden müssen, sind

alle Stakeholder in eine Stakeholder Map (siehe Abbildung 3.1) kategorisiert. Eine Stakeholder Map ist ein grundlegendes Mittel, um die Verhältnisse zwischen den Stakeholdern darzustellen[vgl. Gio+18, S. 3f]. Die Stakeholder Map in Abbildung 3.1 stellt dar, wie der Einfluss von Stakeholdern auf ein RQMS ausgeprägt ist und wie die Auswirkungen durch die Nutzung des RQMS auf die Stakeholder ausgeprägt sind. Als Vorlage wurde die Stakeholder Map von Lucidchart genutzt[Luc].

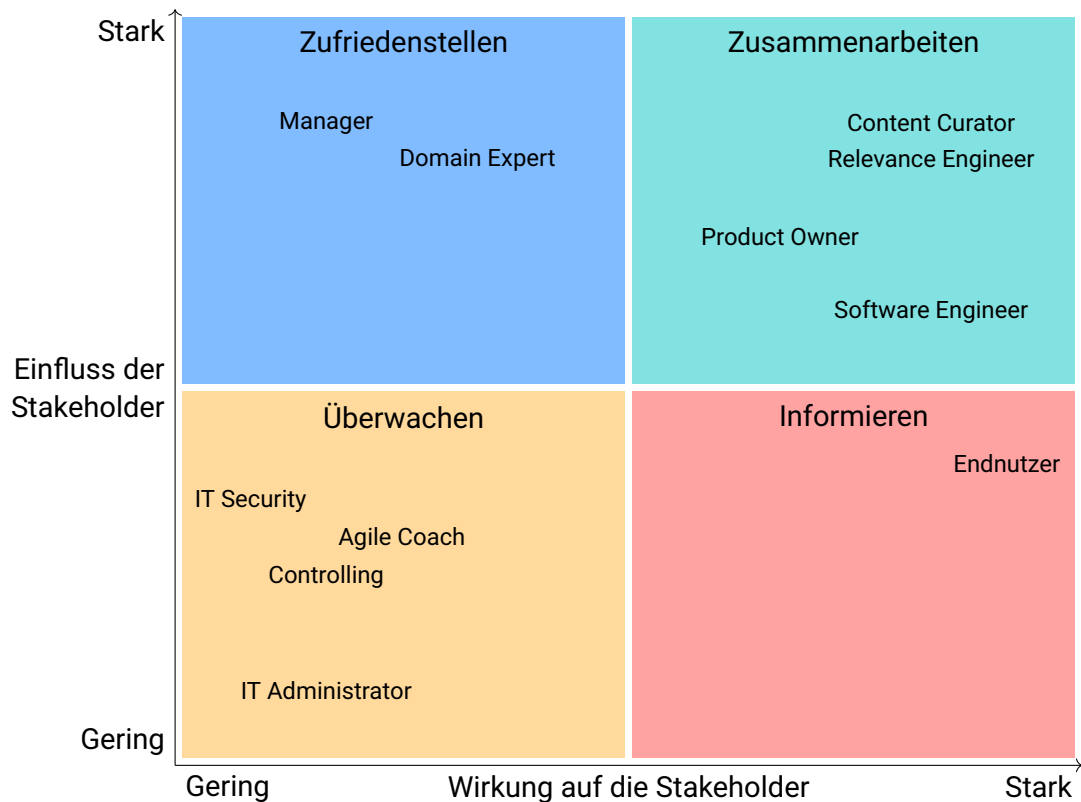


Abbildung 3.1: Stakeholder Map für ein RQMS

Die für die Entwicklung eines RQMS wichtigsten Stakeholder sind in den Quadranten „Zufriedenstellen“ und „Zusammenarbeit“ zu finden. Die Stakeholder im Quadrant „Zufriedenstellen“ nehmen durch ihre Rolle in der Organisation einen großen Einfluss auf das RQMS bzw. die Suchfunktion, sie haben jedoch wenige Berührungspunkte mit den umgesetzten Relevanzverbesserungen. Für diese Gruppe sollten im RQMS Möglichkeiten vorgesehen werden, die Verbesserungen erfahrbar zu machen. Die Gruppe der Stakeholder im Quadrant „Zusammenarbeit“ sind von Änderungen der Relevanz stark betroffen, haben aber auch großen Einfluss auf das RQMS. Diese Stakeholder sind die wichtigste Nutzergruppe für das RQMS.

3.2.2 User Profiles

„Ein User Profile enthält in Auflistungsform alle für einen bestimmten Stakeholder wesentliche Merkmale samt der Ausprägungsattribute des jeweiligen Merkmals“[siehe Har09a, S. 27]. Für die Charakterisierung der Stakeholder eines RQMS werden folgende Merkmale genutzt:

- Berufserfahrung
- Position in der Organisation
- Formale Qualifikationen
- Kernkompetenzbereich
- Aufgaben
- Motivation
- Verantwortungsbereich

Aus der Liste von Merkmalen in „Grundlagen des Systementwurfs“[siehe Har09a, S. 27] sind diese ausgewählt worden, da sie als die relevantesten Merkmale eingeschätzt werden.

Content Curator

Ein Content Curator nimmt teilweise dieselben Aufgaben wie ein Product Owner ein. Wenn ein Scrum Team auch noch für andere Themen außer der Suche verantwortlich ist, ist es häufig sinnvoll, dass sowohl ein Product Owner als auch ein Content Curator im Team vorhanden sind. Bei dedizierten Teams wird nur der Content Curator benötigt. Üblicherweise ist der Content Curator ein erfahrener Product Owner mit Spezialisierung auf Suche. Die Charakteristiken dieser Rolle sind in Tabelle 3.1 zusammengefasst.

Berufserfahrung	> 5 Jahre
Position in der Organisation	Teammitglied im Scrum Team
Formale Qualifikationen	Product Owner mit Spezialisierung auf Suche
Kernkompetenzbereich	Domänenwissen und Nutzergefühl
Aufgaben	<ul style="list-style-type: none">• Analyse der Nutzerdaten• Sammlung von Feedback• Erstellung von Aufgaben zur Verbesserung der Relevanz
Motivation	Kundennutzen im Mittelpunkt
Verantwortungsbereich	Relevanz der Suchergebnisse

Tabelle 3.1: Merkmale des Content Curator

Relevance Engineer

Die zentrale Aufgabe eines Relevance Engineers ist es, die Search Engine in ein intelligentes System zu verwandeln, welches die Anforderungen der Nutzer und der Organisation versteht[TB16, S. 2f]. Üblicherweise ist der Relevance Engineer ein erfahrener Software Engineer mit Spezialisierung auf Suche. Die Charakteristiken dieser Rolle sind in Tabelle 3.2 zusammengefasst.

Berufserfahrung	> 3 Jahre
Position in der Organisation	Teammitglied im Scrum Team
Formale Qualifikationen	Softwareentwickler mit Spezialisierung auf Relevanz
Kernkompetenzbereich	Search Engine Handling
Aufgaben	<ul style="list-style-type: none"> • Umsetzung der Relevanzverbesserungen • Performance Optimierung der Suche • Definition der Anforderungen an die Daten
Motivation	Qualitätsbedürfnis der Relevanz
Verantwortungsbereich	Search Engine

Tabelle 3.2: Merkmale des Relevance Engineer

Software Engineer

Die Eigenschaften eines Softwareentwicklers sind je nach Domäne anders. Sie müssen die Qualität der Software aufrecht erhalten, sowohl im Bereich der Fehlerfreiheit, als auch die Performance einer Anwendung [SS20, S. 5]. In der Entwicklung und Bereitstellung einer Suchfunktion fällt ihnen die Integrationsaufgabe zu. Die Softwareentwickler müssen alle für die Suche benötigten Daten aus den verschiedenen Quellen zusammenführen und dem Endnutzer der Suchfunktion eine Schnittstelle zur Interaktion mit der Suche bereitstellen. Die Charakteristiken dieser Rolle sind in Tabelle 3.3 zusammengefasst.

Berufserfahrung	> 1 Jahr
Position in der Organisation	Teammitglied im Scrum Team
Formale Qualifikationen	Informatik Studium, Ausbildung zum Fachinformatiker
Kernkompetenzbereich	Softwareentwicklung
Aufgaben	<ul style="list-style-type: none"> • Zusammenführen der Daten aus den Quellen • Bereitstellung der Suche für die Endnutzer • Sicherstellung von Qualität und Performance
Motivation	Qualitätsbedürfnis der Anwendung
Verantwortungsbereich	Softwaresysteme die mit der Search Engine interagieren

Tabelle 3.3: Merkmale des Software Engineers

Product Owner

Der Product Owner trägt die Verantwortung für die Suche bzw. das Produkt, in welches die Suche integriert ist. Ein Product Owner hat die Aufgabe, den Wert des Produkts zu steigern. Dabei muss auf einen effektiven Ressourceneinsatz geachtet werden. Viele Aufgaben kann der Product Owner delegieren, aber er ist die Schnittstelle zwischen dem Scrum Team und der restlichen Organisation [SS20, S. 5f]. Die Charakteristiken dieser Rolle sind in Tabelle 3.4 zusammengefasst.

Berufserfahrung	> 3 Jahre
Position in der Organisation	Teammitglied im Scrum Team
Formale Qualifikationen	<ul style="list-style-type: none"> • Betriebswirtschaftsstudium, oder ähnliches • Mehrjährige Erfahrung in der Domäne
Kernkompetenzbereich	Domänenwissen und Nutzergefühl
Aufgaben	<ul style="list-style-type: none"> • Verwaltet alle Aufgaben für das Team • Schnittstelle in die Organisation • Optimale Nutzung der Entwicklungskapazität
Motivation	Wertsteigerung des Produkts
Verantwortungsbereich	Produktverantwortung für die Suchfunktion und gegebenenfalls weitere Komponenten

Tabelle 3.4: Merkmale von Product Owners

Domain Expert

Viele Fachbereiche besitzen Wissen, welches für Außenstehende schwer zu erlernen ist. Gerade dann ist die Rolle des Domain Expert besonders wichtig. Dieser versteht nicht nur den Fachbereich, sondern hat auch genug Erfahrung, um Anderen fachspezifische Begriffe, Handlungsweisen, etc. zu erklären[TB16, S. 265]. Die Charakteristiken dieser Rolle sind in Tabelle 3.5 zusammengefasst.

Berufserfahrung	> 5 Jahr
Position in der Organisation	Fachbereich zugehörig
Formale Qualifikationen	Variiert stark
Kernkompetenzbereich	Experte in der Domäne
Aufgaben	Fachbereichsinterne Aufgaben z. B. Verwalten des Produktsortiments
Motivation	Variiert stark
Verantwortungsbereich	Domänenspezifischer Fachbereich (z. B. Category Management)

Tabelle 3.5: Merkmale von Domain Experts

Manager

Die Rolle des Managers ist nach Scrum nicht vorgesehen [SS20]. Viele Organisationen ordnen Scrum Teams in ihre Organisationsstrukturen ein, sodass diese einem Manager untergeordnet sind. Der Manager übernimmt dabei die üblichen Aufgaben der Personalführung und steuert auch die Prozesse innerhalb der Organisation, um möglichst effizient zu agieren. Ein Manager hat einige Jahre Berufserfahrung, meistens auch im Bereich, für welchen er zuständig ist. Die Charakteristiken dieser Rolle sind in Tabelle 3.6 zusammengefasst.

Berufserfahrung	10-35 Jahre
Position in der Organisation	Führungskraft mit Personalverantwortung
Formale Qualifikationen	Betriebswirtschaftlicher oder Informatischer Hintergrund
Kernkompetenzbereich	Personalverwaltung
Aufgaben	<ul style="list-style-type: none"> • Steuerung der Prozesse in der Organisation • Personalmanagement • Kontrolle des Scrum Teams
Motivation	Effiziente Organisation
Verantwortungsbereich	Personalverantwortung für das Scrum Team

Tabelle 3.6: Merkmale von Managern

3.2.3 Personae

„[...] Persona (pl. Personae oder Personas) stammt aus dem angelsächsischen Sprachraum und bezeichnet Aspekte eines Menschen, der anderen präsentiert oder durch diese wahrgenommen wird bzw. eine Rolle oder eine Figur, die durch einen Autor oder Schauspieler eingenommen wird“[Har09a, S. 29]. Sie sind eine Methode des „user centred design“. Personae dienen dazu Benutzer in einer Art und Weise zu charakterisieren, die sich Menschen vorstellen können[Har09a, S. 29ff]. Im Folgenden werden Personae für ein Scrum Team in einem E-Commerce Unternehmen präsentiert. Als Name für dieses Unternehmen wird im Folgenden „ACME“ genutzt. Diese Personae sind aufgrund der Informationen aus verschiedenen Quellen erstellt worden. Die wichtigsten Quellen sind *Relevant search: with applications for Solr and Elasticsearch*[TB16], „The modern architecture of search“[EB17] und *Implementing a modern E-Commerce Search*[Ree20].

Antje Bartz - Content Curator



Abbildung 3.2: Antje Bartz - Content Curator, aus Unsplash[Sol20]

Antje ist 41 Jahre alt, verheiratet und hat zwei Kinder im Alter von 8 und 11 Jahren. Familie hat einen hohen Stellenwert in ihrem Leben. Bereits während ihres Studiums für Betriebswirtschaftslehre hat sie das Projektmanagement in einer IT Agentur als Werksstudentin unterstützt. Sie ist bis heute davon begeistert, Produkte zu entwickeln, welche von Kunden gerne genutzt werden - daher hat sie auch schon während ihres Studiums viel Energie in ihre Arbeit investiert. Die Fähigkeit, für Qualität im Detail zu sorgen, ohne dabei das große Ganze aus den Augen zu verlieren, zeichnet Antje aus. Dadurch hat sie schließlich ihren Weg zum Produktmanagement gefunden. Weil sie das Gefühl hatte, nicht effektiv genug zu arbeiten, hat sie sich viel mit Personalmanagement beschäftigt und besitzt die Fähigkeit, Aufgaben zu priorisieren und dabei ihrem hohen Qualitätsbedürfnis zu genügen. Aus dem Bedürfnis heraus, sich selbst weiter zu verbessern, hat sie eine Schulung mit Zertifizierung zum Agile Product Owner absolviert. In ihrer Kommunikation ist sie kompetent und ruhig. Sie mag es nicht, wenn Leute reden, ohne auf den Punkt zu kommen oder zu wissen, was sie eigentlich sagen. Daher sind auch ihre eigenen Antworten immer sehr bedacht. Seit der Geburt ihrer Kinder achtet sie auf eine ausgewogene Work-Life-Balance und hat deshalb auch den Arbeitgeber gewechselt. Antje ist nun seit zwei Jahren Content Curator und für die Suche bei ACME zuständig. Vorher war sie als Product Owner im Bereich Discovery zuständig, als die Organisation sich jedoch aufgrund von starkem Wachstum neu strukturierte, hat sie die Chance genutzt, um sich auf die Stelle des Content Curators zu bewerben. Die Stelle bietet für sie die Möglichkeit, ihre Work-Life-Balance zu verbessern, da weniger Abstimmungen mit externen Partnern notwendig sind und die Kommunikation innerhalb der Organisation wesentlich einfacher ist. Sie ist sehr zufrieden mit ihrer aktuellen Position und möchte in dieser Rolle noch ein paar Jahre arbeiten. Bei ACME arbeitet sie eng mit Boris und Edith zusammen, um die Suchergebnisse relevanter zu gestalten und wenn nötig das Sortiment anzupassen. Sie würde gerne wissen, welche Effekte die von Boris implementierten Änderungen

auf die Nutzer haben. Aktuell überwachen sie die Änderungen im Nutzerverhalten nur sporadisch und haben keinen Überblick über die Relevanz aller Suchen. Außerdem hat sie oft das Gefühl, dass Relevanz-Probleme erst im echten Onlineshop auffallen. Sie würde gerne neue Versionen mit einem guten Gefühl veröffentlichen. Automatische Tests der Relevanz sind für sie ein wichtiger Schritt in diese Richtung.

Boris Reichert - Relevance Engineer



Abbildung 3.3: Boris Reichert - Relevance Engineer, aus Unsplash[Pol18a]

Seit der achten Klasse interessierte sich Boris für Computer und Softwareentwicklung. Nach seinem Abitur hat er ein Informatik-Studium begonnen. Schon neben dem Studium hat Boris in einem mittelständischen Unternehmen in der Softwareentwicklung gearbeitet. Dort hat er unter anderem „Test Driven Development“, „Unit Tests“ und „Integration Tests“ kennengelernt. In seiner täglichen Arbeit hat er Spaß daran, sich zu überlegen, was noch nicht getestet wurde und Fehler enthalten könnte. Nach dem Studium hat er bei einer Versicherung gearbeitet. Dort war es für ihn schwierig, mit den Kollegen zusammenzuarbeiten, da die Unternehmenskultur Fehler nicht tolleriert hat. Fehler wurden systematisch vertuscht und Qualitätskontrolle wurde nur pro forma durchgeführt. Als engagierter Softwareentwickler wollte sich Boris weiterentwickeln und hat einen Arbeitgeber gesucht, für welchen Qualität eine hohe Priorität hat. Eine Agentur für E-Commerce-Systeme hat dabei sein Interesse geweckt und es stellte sich heraus, dass beide die gleichen Ansprüche an Softwarequalität teilen. In seinem neuen Team ist er gut zurecht gekommen und gemeinsam haben sie einige Jahre lang erfolgreich Projekte umgesetzt. In einem Projekt lernte er sogar seine jetzige Frau kennen, mit welcher er zwei Kinder im Alter von 10 und 13 Jahren hat. Während dieser Zeit hat sich Boris immer mehr auf das Thema Suche spezialisiert. Er hat Schulungen und Konferenzen rund um das Thema Suche besucht und war immer die erste Wahl, wenn eine Suchfunktion für den Kunden benötigt wurde. Vor zwei Jahren haben die ersten Kollegen die Firma verlassen und Boris fing auch an neue Herausforderungen zu suchen. Bei ACME ist er

fündig geworden und arbeitet dort seit 1,5 Jahren als Relevance Engineer. Mit seiner aktuellen Position ist er zufrieden, aber er möchte langfristig als Architekt für Suchsysteme aufsteigen. Boris ist inzwischen 41 Jahre alt und achtet auch auf seine Gesundheit. Ebenso wie Antje ist ihm eine gesunde Work-Life-Balance wichtig. Nicht nur deswegen kommt er mit Antje gut zurecht. Er hat auch das Gefühl, dass sie ihm die notwendigen Freiheiten gibt, um neue Ideen auszuprobieren. Ihm missfällt es aber ab und an, dass er nicht weiß, wie die Nutzer im Allgemeinen die Suche benutzen und wo die größten Pain Points für die Nutzer liegen. Zwar bekommt er über Antje Feedback aus der Organisation und von Endnutzern Feedback über den Kundenservice, aber diese Informationen sind immer nur subjektive Eindrücke. Auch mit seinen Kollegen aus der Softwareentwicklung arbeitet er gerne zusammen. Besonders Crista ist sehr interessiert am Thema Suche und hat auch viele Ideen, wie man Kunden besser mit der Suche interagieren lassen kann. Gelegentlich arbeiten die beiden paarweise an diesen Themen.

Crista Gärtner - Software Engineer



Abbildung 3.4: Crista Gärtner - Software Engineer, aus Unsplash[Chr19]

Crista ist jung, dynamisch und ambitioniert. Sie hat ein duales Informatik Studium absolviert und ist nach ihrem Bachelor nach Köln gezogen. Dort arbeitete sie in einer Medienagentur. Ihr gefällt es, Software zu entwickeln, die es in die echte Welt schafft. Vor ein paar Jahren hat sie auf dem Geburtstag ihrer Freundin ihren aktuellen Partner kennengelernt. Im Agenturleben war der Zeitdruck immer so hoch, dass sie ihre Arbeit qualitativ nie so hochwertig entwickeln konnte, wie sie es gerne getan hätte. Deswegen hat sie sich vor etwas mehr als einem Jahr nach einem neuen Arbeitgeber umgesehen. Seit neun Monaten ist sie nun im Scrum Team bei ACME und hat sich gut eingearbeitet. Ihr Wunsch ist es, in der Produktentwicklung bei ACME einen qualitativ guten Onlineshop zu entwickeln, in welchem die Endnutzer gerne einkaufen. Sie ist 29 Jahre alt und möchte auch langfristig bei ACME arbeiten. Aktuell liegt ihr Interessensbereich im User Interface der Suchfunktion. Sie über-

legt sich immer wieder, wie man es den Kunden einfacher machen könnte, die relevanten Produkte im Onlineshop zu finden. Häufig arbeitet sie deshalb mit Boris zusammen. Bei Fragen wendet sie sich an Antje, mit der sie gut zurecht kommt. Allerdings wünscht sie sich, auch selbst mehr über das Nutzerverhalten erfahren zu können. Gerne würde sie sich in die Richtung des Relevance Engineering entwickeln, aber sie möchte die Arbeit am User Interface nicht vollständig aufgeben. Wenn sie Software entwickelt, erstellt sie gewissenhaft Unit-, Integration- und End-to-End Tests. Sie nutzt auch privat gerne den Onlineshop und ist stolz darauf, daran mitarbeiten zu können.

Daniel Sturm - Product Owner



Abbildung 3.5: Daniel Sturm - Product Owner, aus Unsplash[Dis19]

Daniel strotzt nur vor Energie. Er ist ein Macher. Wenn man mit Daniel zusammenarbeitet, merkt man schnell, worauf es ihm ankommt: „Das Produkt zum Kunden bringen“. Seine höchste Priorität ist es, ein Produkt bzw. ein Feature so schnell wie möglich den Kunden zur Verfügung zu stellen. Dabei stellt er hinten an, ob das Feature schon vollkommen fertig oder getestet ist. Hauptsache die Endnutzer können darauf zugreifen, sodass das Team Erkenntnisse darüber sammeln kann, wer das Feature wie und wann benutzt. Basierend auf diesen Daten trifft Daniel seine Entscheidungen, ob eine Weiterentwicklung sinnvoll ist oder nicht. Häufig hat er deshalb Diskussionen mit Entwicklern, wenn diese zunächst sicherstellen wollen, dass die Anwendung wie erwartet funktioniert. Seine Strategie funktioniert bisher ganz gut, denn Daniels wichtigste Fähigkeit, neben der Priorisierung von Aufgaben, ist das Organisieren von Budget. Wenn Daniel davon überzeugt ist, dass Ressourcen sinnvoll eingesetzt werden, bekommt er diese auch vom Management genehmigt. Neben der Arbeit geht Daniel gerne feiern, häufig mit seinen ehemaligen Kommilitonen. Er hat IT-Management und Consulting studiert, bevor er als Product Owner für den Onlineshop bei einem Baumarkt angefangen hat. Vor zwei Jahren ist er, als Product Owner für den Bereich Checkout, zu ACME gewechselt. Nach zwölf Monaten ergab sich für ihn die Chance, in die Rolle des Pro-

duct Owners für das Team „Search and Products“ zu wechseln, welches die Suche und die Produktdetailseite im Onlineshop betreut. Weil dieses Team im Unternehmen sehr anerkannt ist und er es als Karrieresprung gesehen hat, hat er die Chance ergriffen. Daniel ist 35 Jahre alt und Single. Wenn sich die Möglichkeit eröffnet, möchte er in die Rolle des „Delivery Managers“ wechseln. Mit seinem Team kommt er gut zurecht, wobei sich seine und Antjes Persönlichkeiten komplementär ergänzen. Er respektiert und schätzt ihre Meinung und folgt fast immer ihren Vorschlägen. Selbst wenn er die Situation manchmal anders sieht, vertraut er doch auf Antjes Einschätzung.

Edith Marquardt - Domain Expert



Abbildung 3.6: Edith Marquardt - Domain Expert, aus Unsplash[Lia20]

Edith ist in Köln geboren. Dort hat sie auch ihren Realschulabschluss gemacht und bei ACME eine Ausbildung zur Industriekauffrau abgeschlossen. Seitdem arbeitet Edith im Category Management, am Anfang noch für das stationäre Geschäft und seit einigen Jahren ausschließlich für den Onlineshop. Sie war eine der ersten, die aus dem klassischen Teil der Organisation in den Bereich E-Commerce gewechselt ist. Während ihre beiden Kinder aufgewachsen sind, hat sie in Teilzeit gearbeitet. Seitdem ihre Kinder aus dem Haus sind, ist sie wieder in Vollzeit im Büro. Dort wird sie etwas scherzhaft „die Mutti“ genannt, nicht nur weil sie immer ein offenes Ohr für ihre Kollegen hat, sondern auch weil man bei komplexen Fragen zum Thema Category Management am Ende bei ihr landet, egal wen man anfangs fragt. Die meisten Fragen beantwortet sie bei einer Tasse Kaffee. Dabei erzählt sie gerne und ausführlich über die Hintergründe der Thematiken und schafft es so, tiefgehendes Wissen zu vermitteln. Das Gespür für die Kundensegmente hat sie sich über die Jahre hinweg erarbeitet. Sie kennt alle Segmente und weiß, welche Produkte für welches Segment benötigt werden. Auch bei aktuellen Trends ist sie gut informiert und überlegt stetig, wie das Sortiment angepasst werden muss. Den Umgang mit der IT hat sie erlernen müssen. Sie nutzt zwar einige Anwendungen gerne, da diese ihr Arbeit abnehmen, aber im Zweifel hat sie immer Papier

und Stift griffbereit. Edith ist 57 Jahre alt und fährt jeden Tag mit dem E-Bike zur Arbeit, egal bei welchem Wetter. In ihrer Freizeit spielt sie Badminton und geht gerne wandern. Mit ihrer Position ist sie sehr zufrieden und möchte auch noch ein paar Jahre bei ACME arbeiten.

Florian Lemke - Manager



Abbildung 3.7: Florian Lemke - Manager, aus Unsplash[Sol19]

Als Entwicklungsleiter ist Florian für alle Entwickler des ACME Onlineshops verantwortlich. Seine berufliche Laufbahn hat Florian, nach seinem Informatikstudium, als Softwareentwickler begonnen. Nach über zehn Jahren in dieser Rolle hat er erst als Teamleiter und später als Entwicklungsleiter bei einem mittelständischen IT-Unternehmen Personalverantwortung übernommen. Durch einen ehemaligen Arbeitskollegen, mit dem er noch regelmäßig segeln geht, ist er zu ACME gekommen. Der Kollege war bei ACME für das Produktmanagement im Onlineshop zuständig und hat Florian als Entwicklungsleiter vorgeschlagen. Florian hat vor sieben Jahren diese Chance genutzt und ist nun für über 200 Entwickler verantwortlich. Er ist aktuell 62 Jahre alt, plant seine Rente und möchte die Organisation bis dahin weiter stabilisieren, sodass der Onlineshop auch dann noch effektiv entwickelt werden kann, wenn er die Entwicklungsleitung abgegeben hat. Er wünscht sich von seinem Nachfolger dieselben Ansprüche an Qualität und Kundennutzen, die er selbst verfolgt. In seiner Freizeit segelt Florian gerne, einmal im Jahr trifft er sich dafür mit aktuellen und ehemaligen Kollegen am Ijsselmeer. Als Entwicklungsleiter gehört es zu seinen Aufgaben, Schulungen und Konferenzen für Entwickler zu genehmigen. Dabei ist es ihm wichtig, dass die Entwickler auf möglichst viele verschiedene Konferenzen gehen und auch intern über das dort erlangte Wissen berichten. Eine diversifizierte Organisation sieht er als relevant an, wobei er auch versucht, einige Aspekte zu vereinheitlichen, um die Organisation effizient zu halten. Eine weitere Aufgabe ist es, die Anschaffung von externen Systemen zu genehmigen. Dabei berät er sich mit den Architekten, Entwicklern und Product Ownern, um die Notwendigkeit der Investitionen abzuschätzen.

3.3 Aufgabenmodellierung

Zur Anforderungsermittlung können Use Cases und Szenarien beitragen. Beide beschreiben funktionale Anforderungen der Benutzer, zeigen die Interessen der Stakeholder auf und beschreiben, wie sich das System verhalten soll. Weder Use Cases noch Szenarien stellen eine vollständige Anforderungsdokumentation dar. Deshalb müssen diese später noch um die technischen Anforderungen ergänzt werden[vgl. Har09a, S. 38]. Im Folgenden werden als erstes Szenarien dargestellt, aus welchen nachfolgend Use Cases abgeleitet werden.

3.3.1 Szenarien

„Ein Szenario [...] stellt eine konkrete Instanz eines Use Cases dar. Hier geht es um einen bestimmten Bezug in einem konkreten Nutzungskontext“[Har09a, S. 39]. Aus Szenarien können Use Cases abgeleitet werden oder ein Use Case kann durch ein ausformuliertes Szenario verständlicher beschrieben werden. „Charakteristisch für ein Szenario ist es, dass die Modellrepräsentation in narrativer Form (nicht notwendigerweise, aber häufig: textuell basiert) vorliegt und die Aufgabenerledigung reichhaltig, detailliert und aussagekräftig beschreibt“[Har09a, S. 39]. Wenn die Szenarien aus Use Cases erstellt werden, sollte eine „n zu 1“-Relation vermieden werden. Stattdessen sollte es für jeden Use Case eines oder mehrere Szenarien geben. Da im Folgenden erst Szenarien beschrieben werden, aus welchen dann Use Cases abgeleitet werden, können mehrere Use Cases durch ein Szenario repräsentiert werden. Es werden die Personae aus Unterabschnitt 3.2.3 wiederverwendet.

Bereitstellung und Betrieb

Boris ist auf das RQMS aufmerksam geworden und möchte es ausprobieren. Am einfachsten ist es für Boris, das RQMS auf seinem lokalem Rechner zu starten und es dort auszuprobieren. Dazu nutzt Boris Docker und startet mit Docker Compose das RQMS und seine Abhängigkeiten. Nach wenigen Sekunden kann Boris sich auf der Oberfläche des RQMS anmelden. Er versucht einen Dump der Produktionsdaten in die Search Engine des RQMS zu laden und erstellt eine Judgment List. Für Boris ist es wichtig, dass das RQMS mit sinnvollen Default-Werten gestartet wird, damit er das System lokal schnell ausprobieren kann. Für den Produktionsbetrieb muss es möglich sein, alle notwendigen Einstellungen einfach vorzunehmen. An einem der folgenden Tage zeigt Boris einem Kollegen aus der IT Security Abteilung das RQMS. Der Kollege ist begeistert von der Funktionalität des RQMS und der einfachen Konfiguration. Er merkt an, dass für einen Betrieb die Authentifizierung über ein zentrales System mithilfe von z. B. LDAP, SAML oder OIDC erfolgen muss, damit die organisationsinternen Richtlinien zur Nutzerverwaltung erfüllt sind. Außerdem muss das System hochverfügbar und sicherungsfähig sein, damit es den unternehmensinternen Richtlinien genügt. Da das RQMS diese Anforderungen erfüllt, will der Kollege aus der IT Security-Abteilung Boris dabei unterstützen, das System einzuführen.

Manuelle Tests

In einer Pairing Session mit Antje hat Boris eine Idee, wie man für einen bestimmten Fall die Relevanz der Suchergebnisse verbessern könnte. Boris erstellt die neue Konfiguration und speichert diese in Git. Im User Interface des RQMS wählt er die Konfiguration aus und das RQMS erstellt eine neue Instanz einer Search Engine, welche nicht nur die Konfiguration, sondern auch die Produktionsdaten enthält. Innerhalb von wenigen Minuten können die beiden ihre Hypothese überprüfen und sehen, dass Boris Idee zumindest für den betrachteten Fall relevantere Ergebnisse liefert. Auch wenn das RQMS noch nicht vollständig integriert ist, bieten manuelle Tests die Möglichkeit, erste Erfolge zu erzielen. Damit Florian auch überzeugt ist, dass die Nutzung eines RQMS der richtige Weg ist, laden Antje und Boris ihn ein und präsentieren ihm die Ergebnisse der manuellen Tests. Gerade bei der Einführung eines RQMS ist es wichtig, Erfolge zu erzielen, damit der Rückhalt in der Organisation gegeben ist (siehe dazu Unterabschnitt 2.3.2).

Offline-Tests

Antje hat schon länger Judgment Lists erstellt, um die Relevanz zu testen. Bisher hat Boris ein Skript dazu genutzt, um die Tests mit den Judgment Lists durchzuführen. Mit dem Skript war die Auswertung der Tests nur binär möglich. Nachdem Antje die Judgment Lists in das RQMS übertragen hat, startet Sie einen Testlauf. Dafür wählt sie eine für den Fall passende Testmetrik aus[siehe Sch20, S. 17ff]. Wenige Minuten später präsentiert ihr das RQMS das Ergebnis. Sie erhält Werte für die ausgewählten Metriken. Für ein agiles und iteratives Arbeiten möchten Antje und Boris diesen Ablauf automatisieren, sodass bei jeder Anpassung die Tests ausgeführt werden. Das RQMS erlaubt es ihr außerdem Assertion Tests zu definieren, mit welchen sie sicherstellen kann, dass grundlegende Bedingungen erfüllt sind (z. B. bei einer Suche nach „Milch“ enthalten mindestens die Hälfte der zurückgegebenen Ergebnisse „Milch“ im Titel). Diese Assertion Tests definiert Antje zusammen mit Boris. Die Tests halten sie eher allgemein, da sie nur dazu dienen sollen, grobe Konfigurationsfehler zu erkennen.

Einige Tage nach der vollständigen Einrichtung des RQMS erhalten Antje und Boris die ersten Vorschläge zur Erstellung von Judgment Lists. Diese Vorschläge hat das RQMS aufgrund des Nutzerverhaltens erstellt. Antje und Boris prüfen die Vorschläge und fügen sie der Menge der genutzten Judgment Lists hinzu. Das RQMS vergleicht weiterhin die bestehende Menge der Judgment Lists und macht auch Vorschläge, welche Judgment Lists nicht mehr benötigt werden (z. B. weil die Kunden nach anderen Begriffen suchen).

A/B-Tests

Nachdem Antje und Boris sich einige Verbesserungen überlegt haben, möchten sie diese mit Endnutzern der Suchfunktion testen. Zur Verwaltung von A/B-Tests nutzen sie `Optimize.ly[Opt]`. Dort erstellen sie die Nutzergruppen und das Experiment für diesen Test. Im RQMS müssen sie nun die Variationen des Experiments den entsprechenden Konfigurationen zuweisen. Für alle Nutzer, die an keinem A/B-Test teilnehmen, gibt es eine Default-Konfiguration.

Das RQMS dupliziert nun alle Produktionsdaten für die verschiedenen Konfigurationen. Nach 14 Tagen können Antje und Boris den A/B-Test auswerten. Das RQMS stellt dazu verschiedene Metriken bereit. Die wichtigsten Metriken für Antje und Boris sind: Click-through rate (CTR), Average Rank (AR), Abandoment (AB) und Deflection[siehe Sch20, S. 22ff]. Im RQMS kann der Verlauf innerhalb einer A/B-Test Variation und über alle Variationen abgefragt werden.

Auswertung der Suchen

Damit Edith das Sortiment anpassen kann, möchte sie wissen, nach welchen Begriffen die Kunden suchen, aber keine bzw. wenige Ergebnisse für diesen Suchbegriff erhalten. Diese Suchen werden auch Nullsuchen genannt. Häufig lässt sich daraus erkennen, welche Produkte die Kunden kaufen möchten. Bisher musste Boris ihr diese Informationen aus den Logs der Suchanwendungen herausfiltern. Mit der Einführung des RQMS kann Edith sich diese Begriffe einfach anzeigen lassen und somit schnell Produkte finden, welche in das Sortiment aufgenommen werden sollten. Für Florian ist vor allem die Metrik Average Rank (AR) interessant. Er leitet daraus die Relevanz der Suchergebnisse ab, denn je geringer der Wert dieser Metrik, desto weiter oben in der Ergebnisliste ist das gesuchte Produkt positioniert. Auch Antje und Boris interessieren sich für Metriken, allerdings nicht nur für den Average Rank, sondern auch für die Click-through rate (CTR), Abandoment (AB), Deflection und Nullsuchen, da diese Metriken hinweise auf die Qualität der Relevanz der Suchergebnisse geben[siehe Sch20, S. 22ff].

3.3.2 Use Cases

„Ein Use Case (Anwendungsfall) stellt ein Modell dar, das beschreibt, wie Benutzer eine Funktion eines interaktiven Systems zur Durchführung ihre Aufgabe(n) einsetzen“[Har09a, S. 38]. Ein Use Case beinhaltet neben einer Zielsetzung die allgemeine Aufgabendurchführung, welche die geschlossenen Handlungsabfolgen des Benutzers repräsentiert[Har09a, S. 38]. Nach Cockburn enthält ein Use Case eine eindeutige Nummer, die charakterischen Informationen (z. B. Hauptakteure, Ziel, Umfang), ein Hauptszenario, die Erweiterungen dieses Szenarios, alle Untervariationen und weitere optionale Angaben[vgl. Coc98]. Diese Informationen werden in den nächsten Abschnitten in tabellarischer Form dargestellt:

- **Szenario:** Das Szenario, aus welchem der Use Case abgeleitet wurde
- **Ebene:** Die Detailstufe des Use Cases, welche als „Zusammenfassung“, „Hauptaufgabe“ oder „Unteraufgabe“ angegeben wird
- **Ziel:** Ausformulierte Beschreibung des Ziels, welches durch die Handlung erreicht werden soll
- **Vorbedingungen:** Die Ausgangssituation, in welcher die Handlung startet
- **Erfolgreich:** Der Zustand der Situation, in welcher die Handlung im Erfolgsfall endet

- **Fehlgeschlagen:** Der Zustand der Situation, in welcher die Handlung im Fehlerfall endet
- **Hauptakteure:** Rollenbezeichnungen für die primär beteiligten Akteure
- **Auslösung:** Optionale Angabe, wenn die Handlung aufgrund eines vorangegangenen Ereignisses startet
- **Beschreibung:** Eine vollständige Liste an Schritten, vom Start bis zur Zielerreichung und, wenn nötig, auch noch danach (z. B. Aufräumarbeiten)
- **Erweiterungen:** Untergeordnete Handlungsabfolgen
- **Varianten:** Verzweigungen im beschriebenen Handlungsablauf

UC 1 - Einfache Bereitstellung

UC 1	Einfache Bereitstellung
<i>Szenario</i>	Bereitstellung und Betrieb
<i>Ebene</i>	Zusammenfassung
<i>Ziel</i>	Ein Relevance Engineer sammelt mit geringem Aufwand Erfahrungen mit dem RQMS
<i>Vorbedingungen</i>	Der Relevance Engineer sitzt an einem Computer, auf welchem er das RQMS testen kann
<i>Erfolgreich</i>	Der Relevance Engineer hat das RQMS nutzen können, hat dabei Erfahrungen gesammelt und hatte wenig Aufwand
<i>Fehlgeschlagen</i>	Der Relevance Engineer konnte das RQMS nicht nutzen
<i>Hauptakteure</i>	Relevance Engineer
<i>Auslösung</i>	Ein Relevance Engineer möchte Erfahrungen mit dem RQMS sammeln

Beschreibung

1. Der Relevance Engineer sitzt an seinem Computer
2. Er öffnet die Dokumentation des RQMS
3. Er befolgt die Schritte zur Installation des RQMS
4. Er startet das RQMS
5. Das RQMS startet und stellt eine Nutzeroberfläche bereit
6. Der Relevance Engineer meldet sich mit den Zugangsdaten aus der Dokumentation an der Nutzeroberfläche an
7. Er erstellt eine Judgment List
8. Er testet die Judgment List gegen einen Beispieldatensatz

9. Das RQMS präsentiert die Testergebnisse
10. Der Relevance Engineer beendet das RQMS

UC 2 - Authentifizierung über externes System

UC 2	Authentifizierung über externes System
<i>Szenario</i>	Bereitstellung und Betrieb
<i>Ebene</i>	Hauptaufgabe
<i>Ziel</i>	Das RQMS nutzt die zentrale Nutzerverwaltung der Organisation, weil es für die Organisation einfacher und sicherer ist, Nutzer zentral zu verwalten[siehe RH06]
<i>Vorbedingungen</i>	Der Nutzer besitzt einen gültigen Account am externen System und die Authentifizierung über ein externes System ist am RQMS konfiguriert
<i>Erfolgreich</i>	Der Nutzer ist am RQMS authentifiziert und kann dieses nutzen
<i>Fehlgeschlagen</i>	Der Nutzer kann das RQMS nicht nutzen
<i>Hauptakteure</i>	IT-Security, Content Curator, Relevance Engineer, Software Engineer, Product Owner, Domain Expert und Manager
<i>Beschreibung</i>	<ol style="list-style-type: none"> 1. Der Nutzer öffnet das RQMS 2. Das RQMS ermöglicht dem Nutzer die Authentifizierung am externen System 3. Der Nutzer meldet sich mit den Zugangsdaten des externen Systems an 4. Der Nutzer ist am RQMS authentifiziert und kann alle seiner Rolle entsprechenden Funktionen nutzen

Varianten

- 2a Der Nutzer muss die Authentifizierung am externen System explizit auswählen
 - 2b Für den Nutzer wird automatisch die Authentifizierung am externen System ausgewählt
 - 3a Das RQMS bietet eine Anmeldemaske an und authentifiziert den Nutzer selbst am externen System
 - 3b Der Nutzer wird an das externe System weitergeleitet und authentifiziert sich dort
-

UC 3 - Hochverfügbarkeit

UC 3	Hochverfügbarkeit
<i>Szenario</i>	Bereitstellung und Betrieb
<i>Ebene</i>	Zusammenfassung
<i>Ziel</i>	Das RQMS soll immer verfügbar sein, auch wenn Hardware ausfällt
<i>Vorbedingungen</i>	Das RQMS läuft stabil
<i>Erfolgreich</i>	Das RQMS ist zu jedem Zeitpunkt erreichbar
<i>Fehlgeschlagen</i>	Das RQMS ist nicht erreichbar
<i>Hauptakteure</i>	IT-Administrator, Content Curator, Relevance Engineer, Software Engineer, Product Owner, Domain Expert und Manager
<i>Auslösung</i>	Hardwareausfall

Beschreibung

1. Das RQMS läuft stabil und alle Benutzer können darauf zugreifen
2. Einer der Rechner, auf welchem das RQMS läuft, wird vom Strom getrennt

3. Das Orchestrierungssystem erkennt den Ausfall
 4. Alle Anfragen werden auf die verbleibenden Instanzen des RQMS geleitet
 5. Eine neue Instanz des RQMS wird vom Orchestrierungssystem gestartet
 6. Die neue Instanz läuft stabil und wird vom Orchestrierungssystem als stabil erkannt
 7. Das RQMS läuft stabil und alle Benutzer können darauf zugreifen
-

UC 4 - Sicherungsfähigkeit

UC 4	Sicherungsfähigkeit
<i>Szenario</i>	Bereitstellung und Betrieb
<i>Ebene</i>	Hauptaufgabe
<i>Ziel</i>	Die gelöschte Judgment List kann wiederhergestellt werden
<i>Vorbedingungen</i>	Der Content Curator hat eine Judgment List gelöscht
<i>Erfolgreich</i>	Die gelöschte Judgment List ist wiederhergestellt
<i>Fehlgeschlagen</i>	Die gelöschte Judgment List kann nicht wiederhergestellt werden
<i>Hauptakteure</i>	IT-Administrator, Content Curator
<i>Auslösung</i>	Ein Nutzer möchte gelöschte Daten wiederherstellen
<i>Beschreibung</i>	
<ol style="list-style-type: none"> 1. Der Content Curator kontaktiert den IT-Administrator 2. Der IT-Administrator stellt das Backup mit den entsprechenden Daten wieder her 3. Der Content Curator überprüft die wiederhergestellten Daten 	

UC 5 - Manuelle Relevanztests

UC 5	Manuelle Relevanztests
<i>Szenario</i>	Manuelle Tests
<i>Ebene</i>	Hauptaufgabe
<i>Ziel</i>	Für eine Konfiguration soll bestimmt werden, ob sie die Relevanz der Suchergebnisse verbessert
<i>Vorbedingungen</i>	Judgment Lists existieren und die Konfiguration ist für das RQMS zugänglich
<i>Erfolgreich</i>	Der Wert der Relevanzmetrik für die Konfiguration ist bekannt
<i>Fehlgeschlagen</i>	Der Wert der Relevanzmetrik konnte nicht ermittelt werden
<i>Hauptakteure</i>	Relevance Engineer, Content Curator
<i>Auslösung</i>	Eine neue Konfiguration soll getestet werden
<i>Beschreibung</i>	
<ol style="list-style-type: none">1. Der Relevance Engineer erstellt einen Testfall und weist die Konfiguration dem Testfall zu2. Das RQMS erstellt eine Instanz der Search Engine mit der neuen Konfiguration3. Der Relevance Engineer wählt die passende Relevanzmetrik aus und startet einen Testlauf4. Das RQMS führt den Testlauf auf allen Judgment Lists aus und berechnet den Wert der ausgewählten Relevanzmetrik5. Das RQMS speichert den Testlauf und stellt den Wert der Relevanzmetrik im zeitlichen Verlauf dar	

UC 6 - Jugment List erstellen

UC 6	Jugment List erstellen
<i>Szenario</i>	Offline-Tests
<i>Ebene</i>	Hauptaufgabe
<i>Ziel</i>	Eine neue Suchanfrage soll für die automatischen Relevanztests verwendet werden
<i>Vorbedingungen</i>	Der Content Curator hat die erwarteten Ergebnisse für die Suchanfrage definiert
<i>Erfolgreich</i>	Eine Jugment List für automatische Relevanztests wurde erstellt
<i>Fehlgeschlagen</i>	Die Judgment List konnte nicht erstellt werden
<i>Hauptakteure</i>	Content Curator
<i>Auslösung</i>	Eine Suchanfrage wird aktuell nicht von den automatischen Relevanztests abgedeckt
<i>Beschreibung</i>	<ol style="list-style-type: none">1. Der Content Curator erstellt eine neue Judgment List im RQMS2. Das RQMS zeigt ein User Interface, mit welchem zu einem Suchbegriff die relevanten Produkte zugeordnet werden können3. Der Content Curator speichert die erstellte Judgment List

UC 7 - Jugment List bearbeiten

UC 7	Jugment List bearbeiten
<i>Szenario</i>	Offline-Tests
<i>Ebene</i>	Hauptaufgabe
<i>Ziel</i>	Die Judgment List muss andere Ergebnisse beinhalten, da sich die Ergebnismenge geändert hat
<i>Vorbedingungen</i>	Eine Judgment List existiert und der Content Curator hat die erwarteten Ergebnisse für die Suchanfrage definiert
<i>Erfolgreich</i>	Die Judgment List beinhaltet andere Ergebnisse
<i>Fehlgeschlagen</i>	Die Judgment List konnte nicht angepasst werden
<i>Hauptakteure</i>	Content Curator
<i>Auslösung</i>	Die Ergebnismenge hat sich geändert
<i>Beschreibung</i>	<ol style="list-style-type: none">1. Der Content Curator wählt die zu bearbeitende Judgment List im RQMS aus2. Die für den Suchbegriff relevanten Produkte werden in der Judgment List angepasst3. Die Judgment List wird gespeichert

UC 8 - Judgment List löschen

UC 8	Jugment List löschen
<i>Szenario</i>	Offline-Tests
<i>Ebene</i>	Hauptaufgabe
<i>Ziel</i>	Eine Judgment List soll gelöscht werden, weil die Suchanfrage nicht mehr getestet werden soll
<i>Vorbedingungen</i>	Eine Judgment List existiert
<i>Erfolgreich</i>	Die Judgment List existiert nicht mehr
<i>Fehlgeschlagen</i>	Die Judgment List konnte nicht gelöscht werden
<i>Hauptakteure</i>	Content Curator
<i>Auslösung</i>	Eine Judgment List wird nicht mehr benötigt
<i>Beschreibung</i>	<ol style="list-style-type: none">1. Der Content Curator wählt die zu löschende Judgment List im RQMS aus2. Der Content Curator löscht die ausgewählte Judgment List

UC 9 - Automatische Relevanztests

UC 9	Automatische Relevanztests
<i>Szenario</i>	Offline-Tests
<i>Ebene</i>	Hauptaufgabe
<i>Ziel</i>	Eine Konfiguration soll in ihrer Relevanz verbessert werden
<i>Vorbedingungen</i>	Judgment Lists existieren und eine erste Version der Konfiguration existiert

<i>Erfolgreich</i>	Eine Konfiguration wurde iterativ in ihrer Relevanz verbessert
<i>Fehlgeschlagen</i>	Die Judgment List-Tests haben keinen Beitrag zur Verbesserung der Relevanz der Konfiguration beigetragen
<i>Hauptakteure</i>	Content Curator und Relevance Engineer
<i>Auslösung</i>	Qualität der Suchfunktion soll verbessert werden
<i>Beschreibung</i>	
<ol style="list-style-type: none"> 1. Der Relevance Engineer erstellt einen Testfall und weist die Konfiguration dem Testfall zu 2. Das RQMS erstellt eine Instanz der Search Engine mit der zugewiesenen Konfiguration 3. Der Relevance Engineer wählt die passende Relevanzmetrik aus und aktiviert die automatische Testausführung 4. Das RQMS führt initial einen Testlauf auf allen Judgment Lists aus und berechnet den Wert der ausgewählten Relevanzmetrik 5. Bei jeder Änderung der Konfiguration führt das RQMS einen Testlauf aus und berechnet den Wert der ausgewählten Relevanzmetrik 6. Das RQMS stellt den Wert der Relevanzmetrik im zeitlichen Verlauf dar 	
<i>Erweiterung</i>	
<ol style="list-style-type: none"> 5a Das RQMS erstellt mit der Konfiguration eine Instanz der Search Engine mit der neuen Konfiguration 5b Das RQMS führt den Testlauf auf allen Judgment Lists aus und berechnet den Wert der ausgewählten Relevanzmetrik 5c Das RQMS speichert den Testlauf und den Wert der Relevanzmetrik für spätere Analysen 	

UC 10 - Jugment List-Vorschläge

UC 10	Jugment List-Vorschläge
<i>Szenario</i>	Offline-Tests
<i>Ebene</i>	Hauptaufgabe
<i>Ziel</i>	Eine Judgment List wurde automatisch erstellt
<i>Vorbedingungen</i>	Das RQMS erhält alle benötigten Informationen zu den von Endnutzern getätigten Suchen
<i>Erfolgreich</i>	Eine neue Judgment List wird auf Grundlage einer bisher nicht abgedeckten, häufig genutzen Suchanfrage erstellt
<i>Fehlgeschlagen</i>	Für häufig genutzte Suchanfragen existieren keine Judgment Lists
<i>Hauptakteure</i>	Content Curator
<i>Auslösung</i>	Suche eines Endnutzers
<i>Beschreibung</i>	
<ol style="list-style-type: none">1. Ein Endnutzer sucht nach einem Begriff und interagiert mit einem Ergebnis2. Das RQMS erhält Informationen über die erfolgreiche Suche3. Der Begriff wurde bereits häufig von anderen Endnutzern gesucht4. Das RQMS erstellt eine Judgment List für den Suchbegriff und die von den Kunden genutzten Ergebnisse5. Die neue Judgment List ist in einem Zustand, in welchem sie nicht für Testläufe genutzt wird6. Der Content Curator bearbeitet die Judgment List wenn nötig7. Der Content Curator aktiviert die Judgment List, sodass diese für Tests genutzt werden kann	

UC 11 - Assertion Test Management

UC 11	Assertion Test Management
<i>Szenario</i>	Offline-Tests
<i>Ebene</i>	Zusammenfassung
<i>Ziel</i>	Der Relevance Engineer möchte mit binären Tests sicherstellen, dass eine Konfiguration definierte Bedingungen erfüllt
<i>Vorbedingungen</i>	Es sind Bedingungen bekannt, welche eine Konfiguration erfüllen muss
<i>Erfolgreich</i>	Eine Konfiguration wurde auf die definierten Bedingungen hin getestet
<i>Fehlgeschlagen</i>	Es ist kein Test möglich
<i>Hauptakteure</i>	Relevance Engineer, Content Curator
<i>Beschreibung</i>	
<ol style="list-style-type: none">1. Der Relevance Engineer erstellt für jede Bedingungen eine Assertion2. Der Relevance Engineer wählt einen Testfall aus, dessen Konfiguration getestet werden soll3. Das RQMS erstellt einen Testlauf und führt die Assertion Tests für die Konfiguration aus4. Alle Ergebnisse werden im RQMS gespeichert und dargestellt5. Der Relevance Engineer kann die Konfiguration anpassen, sodass sie alle Assertions erfüllt	

UC 12 - Erstellen eines A/B-Tests

UC 12	Erstellen eines A/B-Tests
<i>Szenario</i>	A/B-Tests
<i>Ebene</i>	Hauptaufgabe
<i>Ziel</i>	Die Qualität einer neuen Konfiguration soll anhand der Nutzung von Endnutzern beurteilt werden
<i>Vorbedingungen</i>	Ein Experiment ist im A/B-Testing Tool angelegt
<i>Erfolgreich</i>	Der A/B-Test ist gestartet und die verschiedenen Nutzergruppen erhalten unterschiedliche Ergebnisse
<i>Fehlgeschlagen</i>	Die Nutzergruppen erhalten alle dieselben Ergebnisse
<i>Hauptakteure</i>	Relevance Engineer, Content Curator
<i>Auslösung</i>	Offline-Tests einer Konfiguration zeigen eine Steigerung der Relevanz

Beschreibung

1. Ein neuer A/B-Test wird angelegt
2. Jeder Variante des A/B-Tests wird eine Search Engine Konfiguration zugewiesen
3. Der A/B-Tests wird gespeichert
4. Der A/B-Test wird aktiviert
5. Für jede Suchanfrage eines Endnutzers wird im Search Service entschieden, welche Konfiguration genutzt werden soll
6. Jede Suchanfrage wird im RQMS erfasst und entsprechend in den Statistiken gespeichert

UC 13 - Auswerten eines A/B-Tests

UC 13	Auswerten eines A/B-Tests
<i>Szenario</i>	A/B-Tests
<i>Ebene</i>	Hauptaufgabe
<i>Ziel</i>	Ein durchgeführter A/B-Test soll analysiert werden, um herauszufinden, ob die Relevanz der Ergebnisse verbessert wurde
<i>Vorbedingungen</i>	Ein A/B-Test ist abgeschlossen
<i>Erfolgreich</i>	Es ist bekannt, ob eine Konfiguration relevantere Ergebnisse erzeugt
<i>Fehlgeschlagen</i>	Keine verwendbaren Informationen vorhanden
<i>Hauptakteure</i>	Relevance Engineer, Content Curator
<i>Auslösung</i>	Abschluss eines A/B-Tests
<i>Beschreibung</i>	<ol style="list-style-type: none">1. Alle Akteure können die Metriken für die Click through rate (CTR), Average Rank (AR), Abandonment (AB) und Deflection im Zeitverlauf analysieren2. Die Analyse der Metriken ermöglicht, zu beurteilen, welche Konfiguration im A/B-Test Suchergebnisse mit einer höheren Relevanz für die Nutzer erzeugt

UC 14 - Fehlende Ergebnisse erkennen

UC 14	Fehlende Ergebnisse erkennen
<i>Szenario</i>	Auswertung der Suchen
<i>Ebene</i>	Hauptaufgabe
<i>Ziel</i>	Damit die Ergebnismenge effektiv erweitert werden kann, sollen Suchanfragen identifiziert werden, für welche keine Ergebnisse vorhanden sind
<i>Vorbedingungen</i>	Suchen der Nutzer werden im RQMS gespeichert
<i>Erfolgreich</i>	Es wurden Suchanfragen identifiziert, für welche neue Ergebnisse erstellt werden
<i>Fehlgeschlagen</i>	Keine verwendbaren Informationen vorhanden
<i>Hauptakteure</i>	Content Curator, Domain Expert
<i>Beschreibung</i>	<ol style="list-style-type: none">1. Das RQMS bietet eine Liste der Suchbegriffe, für welche keine oder sehr wenig Ergebnisse gefunden werden2. Der Content Curator beurteilt, warum keine Ergebnisse gefunden wurden3. Der Content Curator und Domain Expert beurteilen, für welche Suchbegriffe neue Ergebnisse hinzugefügt werden sollen und können (z. B. neues Produkt in Onlineshop aufnehmen)4. Die neuen Ergebnisse werden der Ergebnismenge hinzugefügt5. Die Endnutzer finden die gewünschten Ergebnisse

UC 15 - Beurteilung Qualitätsentwicklung

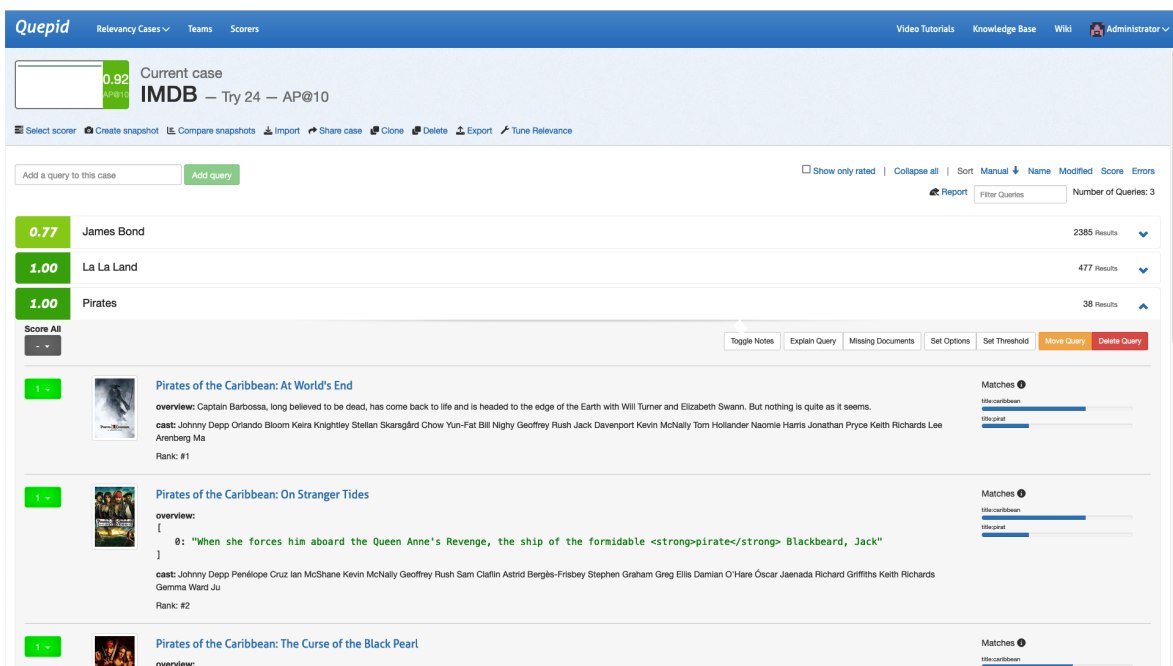
UC 15	Beurteilung Qualitätsentwicklung
<i>Szenario</i>	Auswertung der Suchen
<i>Ebene</i>	Hauptaufgabe
<i>Ziel</i>	Das Management möchte die Qualitätsentwicklung der Suche sehen, um die Leistung der Mitarbeiter zu beurteilen
<i>Vorbedingungen</i>	Qualitätsmetriken werden im RQMS erfasst
<i>Erfolgreich</i>	Das RQMS stellt Metriken zur Ermittlung der Qualität im Zeitverlauf dar
<i>Fehlgeschlagen</i>	Die Qualität der Suchfunktion kann nicht ermittelt werden
<i>Hauptakteure</i>	Manager, Content Curator, Relevance Engineer
<i>Auslösung</i>	Leistungsbeurteilung der Mitarbeiter
<i>Beschreibung</i>	
<ol style="list-style-type: none">1. Das Nutzerverhalten der Suchfunktion wird erfasst2. Das RQMS bereitet die Rohdaten auf3. Das RQMS stellt die folgenden Metriken im Zeitverlauf zur Verfügung:<ol style="list-style-type: none">a) Click-through rate (CTR)b) Average Rank (AR)c) Abandoment (AB)d) Deflectione) Nullsuchen4. Der Manager kann anhand der Metriken die Qualität der Suchfunktion beurteilen	

3.4 Bestehende Anwendungen

Die bisherige Analyse der Anforderungen basiert auf den Ergebnissen des „User centered design“-Prozesses. Eine weitere Möglichkeit, Anforderungen für den Systementwurf zu erarbeiten, ist die Analyse von bestehenden Systemen. Seit mehreren Jahren gibt es schon allgemeine Systeme zur Verbesserung der Relevanz von Suchergebnissen. Die Umsetzung dieser Systeme kann für die Entwicklung einer Architektur genutzt werden, dabei sollten die positiven Eigenschaften der Systeme in die Architektur übernommen werden. Im Folgenden werden drei Vertreter dieser Systeme dargestellt und, soweit möglich, deren Stärken und Schwächen erläutert. Die erste Anwendung ist „Quepid“, sie wurde ausgewählt, weil sie als einziges der drei Systeme vollständig Open Source ist und sowohl Solr als auch Elasticsearch unterstützt. Die zweite Anwendung ist „App Search“, ein Produkt von Elastic, welches auch Open Source ist, aber nur Elasticsearch unterstützt. Das Produkt „Connected Search“ ist die dritte Lösung, es ist eine SaaS Lösung, welche nur von Lucidworks angeboten wird und auch nur Solr unterstützt.

3.4.1 Quepid

Quepid wird als „The Test-Driven Relevancy Dashboard“ beschrieben[Con20b]. Die Anwendung ermöglicht das Verwalten von Judgment Lists, führt automatische Relevanztests mit diesen aus und bietet einen interaktiven Editor für die Struktur der Suchanfragen.



The screenshot shows the Quepid web interface. At the top, there's a navigation bar with 'Quepid' and links for 'Relevancy Cases', 'Teams', 'Scorers', 'Video Tutorials', 'Knowledge Base', 'Wiki', and 'Administrator'. The main content area shows a 'Current case' for 'IMDB' with a score of 0.92. Below this, there's a list of results with scores: 'James Bond' (0.77), 'La La Land' (1.00), and 'Pirates' (1.00). The 'Pirates' result is expanded, showing a detailed view for 'Pirates of the Caribbean: At World's End'. This view includes an overview, a cast list, and a 'Matches' section with a progress bar for 'Microbeeman'.

Abbildung 3.8: Screenshot eines Relevance Cases (bzw. Judgment List) in Quepid

Das System wurde von OpenSource Connections[Wri+22] entwickelt. Doud Turnbull, der Autor von *Relevant search: with applications for Solr and Elasticsearch*[TB16], hat dort bis 2020 gearbeitet und auch selbst an Quepid mitentwickelt[Tur22]. Quepid ist deshalb stark darauf ausgerichtet, die Prozesse in einer Relevance Driven Organization zu unterstützen.

Stärken

Da Quepid nur ein Verwaltungssystem für Judgment Lists ist, liegt genau hier die Stärke des Systems. Es lassen sich sogenannte „Relevancy Cases“ erstellen. Für jeden „Relevancy Case“ muss eine Search Engine definiert werden, welche für die Tests genutzt werden soll. Ein „Relevancy Case“ beinhaltet dann eine Menge an Judgment Lists, für welche der jeweilige Relevance Score ermittelt wird. Außerdem werden alle Relevance Scores auch noch zu einem globalen Relevance Score des „Relevancy Case“ zusammengefasst. Zur Verbesserung der Relevanz ermöglicht Quepid das Anpassen der Suchanfrage in einem speziellen Editor, der sogenannten „Query Sandbox“. Für jedes Dokument in der Judgment List wird auch eine detaillierte Erläuterung geliefert, wie sich der Relevance Score für das jeweilige Dokument zusammensetzt. Erfahrene Benutzer haben außerdem die Möglichkeit, eigene Bewertungsalgorithmen, sogenannte „Scorer“ in das System einzubinden. Diese müssen in JavaScript geschrieben werden und können über das User Interface in der Anwendung gespeichert werden. Das System stellt selbst schon die Bewertungsalgorithmen „Precision P@5“, „Average Precision AP@5“, „Cumulative Gain CG@5“, „Discount Cumulative Gain DCG@5“ und „normalized Discount Cumulative Gain nDCG@5“ bereit[vgl. Con20a].

Schwächen

Quepid ist eine Anwendung, die manuell bedient werden muss. Die Judgment Lists müssen manuell verwaltet werden, die Tests manuell ausgeführt werden und auch die Konfigurationen der Search Engines müssen manuell verwaltet werden. Außerdem fließen keine Informationen über die Nutzung der Suchfunktion in Quepid ein. Die Anwendung ist nur an die Search Engine angebunden und kann somit auch nur Tests ausführen.

Zusammenfassung

Für Organisationen, die im Aufbau einer Suchfunktion sind, kann die Anwendung sehr hilfreich sein, um Judgment Lists zu erstellen, zu verwalten und damit Tests auszuführen. Bei einer existierenden Suchfunktion bietet die Anwendung einen geringeren Mehrwert, da die Nutzungsdaten der Suchfunktion in keiner Weise mit eingebunden werden. Weil die Anwendung Open Source ist, können Entwickler diese erweitern und nach eigenen Vorstellungen anpassen.

3.4.2 Elastic App Search

Elastic App Search ist eine Anwendung zur Erstellung, Verwaltung und Optimierung von einfachen Search Services mit Elasticsearch. Mit App Search lassen sich über eine grafische Nutzeroberfläche einfach Schnittstellen anbinden, grafische Bedienelemente für eine Suchfunktion im Web erstellen, grundlegende Relevanzoptimierungen vornehmen, die Nutzungsdaten der Suchfunktion auf vordefinierte Metriken hin analysieren und die Konfigurationen von Elasticsearch verwalten [vgl. Ela22]. Wenn alle von App Search bereitgestellten Funktionen genutzt werden, lässt sich sehr einfach eine gute Suchfunktionalität erstellen. Falls die Suchfunktionalität komplexere Anwendungsfälle abgedeckt werden soll, wird es teilweise sehr aufwendig

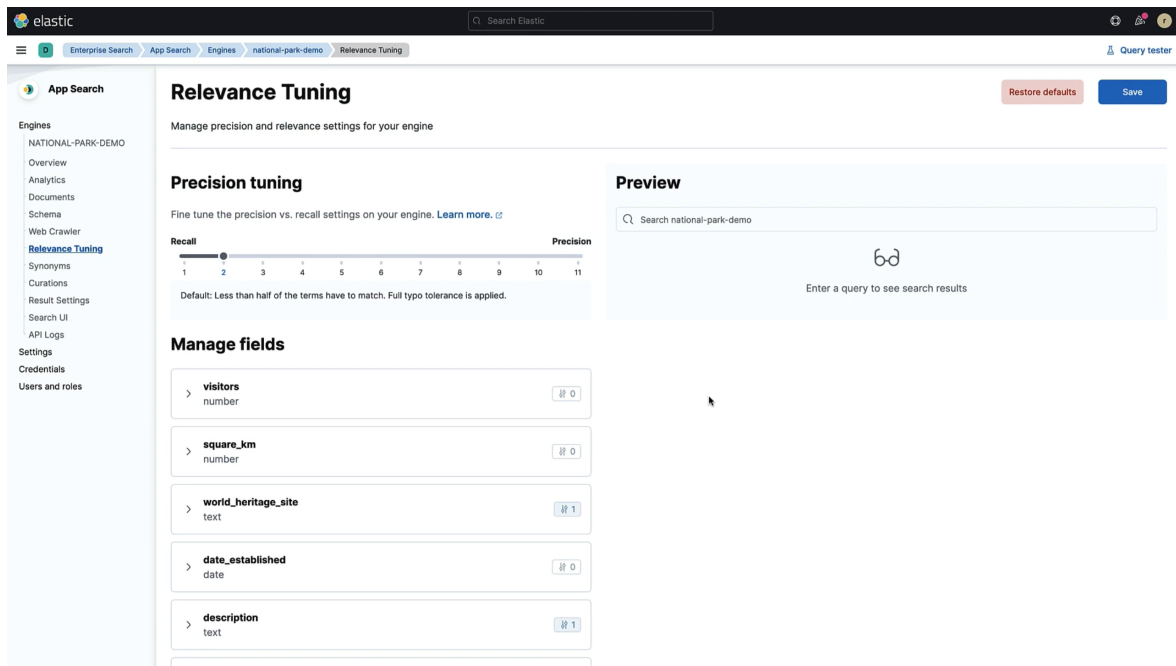


Abbildung 3.9: Screenshot des Relevance Tuning Dialogs in App Search, aus *Elastic App search documentation*[Ela22]

oder sogar unmöglich, diese mit App Search abzubilden. Ein gutes Beispiel ist der Schieberegler im Relevance Tuning Dialog, welcher eine Abwägung zwischen Precision und Recall ermöglicht. Für die komplexeren Anwendungsfälle ist der Schieberegler aber nicht nutzbar, da zum einen viele Parameter sehr spezifisch eingestellt werden müssen und zum anderen diese Abwägung pro Suchbegriff bzw. Art der Suche gemacht werden muss. Ein Beispiel hierfür sind die unterschiedlichen Typen an Suchen bei Spotify (siehe Unterabschnitt 2.1.2), in welchen, je nach Typ, entweder eine sehr hohe Precision (Fokussierte Suche) oder ein sehr hoher Recall (Explorative Suche) benötigt wird.

Stärken

Die Stärke von App Search liegt in der engen Kopplung an den Elastic Stack (die Software Suite von Elastic). So können viele Konfigurationsparameter über die Oberfläche angepasst werden und sind damit für die Content Curators und Domain Experten leicht zugänglich. App Search versucht, die über die verschiedenen Schnittstellen angelieferten Daten automatisch zu verbessern. Gerade für einfache Anwendungsfälle kann diese Funktion sehr hilfreich sein.

Schwächen

Durch die starke Kopplung an den Elastic Stack sind einige komplexere Anwendungsfälle nicht möglich. Sobald mehrere Elasticsearch Cluster genutzt werden sollen, müssen Workarounds implementiert werden oder einige Funktionalitäten sind nicht verfügbar (z. B. Analysen über mehrere Cluster). Auch A/B-Tests sind schwierig umzusetzen, da verschiedene „Engines“ nicht miteinander verglichen werden können. Die Benutzeroberfläche ist auf einfache Anwendungsfälle ausgelegt und kann komplexere Anforderungen nicht abbilden. So kann zum

Beispiel eine segmentierte Engine nicht abgebildet werden. Eine segmentierte Engine tritt unter anderem im E-Commerce auf, wenn ein Produktsortiment aus mehreren Verkaufskanälen besteht. So kann ein Produkt nur für Kunden in einem bestimmten Gebiet auffindbar sein, weil es direkt angeliefert wird. Ein anderes Produkt, welches per Paket versendet wird, kann von allen Kunden gefunden werden.

Außerdem sind die Konfigurationen der jeweiligen Engine nur in Elasticsearch gespeichert und können somit schwer in externen Systemen verwaltet werden. Für Umgebungen, in welchen Systeme nach dem GitOps Konzept administriert werden, stellt das eine große Herausforderung dar. Auch wenn die Konfiguration zusätzlich gesichert werden soll, ist es schwierig, diese wiederherzustellen, da viele Abhängigkeiten in den Elastic Stack bestehen.

Zusammenfassung

Elastic App Search ist gerade für kleinere, einfache Anwendungsfälle gut geeignet, da sich hier direkt eine Suchfunktionalität erstellen lässt. Wenn die Daten auch noch über eine kompatible Schnittstelle angeliefert werden oder statisch sind und damit manuell gepflegt werden, kann App Search alle Anforderungen erfüllen. Je komplexer der Anwendungsfall, desto schwieriger wird es App Search zu nutzen. Gerade in Organisationen mit vielen, sich schnell ändernden Ergebnissen, verschiedenen Typen an Suchen oder auch vielen A/B-Tests ist App Search ungeeignet.

3.4.3 Lucidworks Connected Search

Das System Lucidworks Connected Search ist auf einfache Anwendungsfälle für Suchen zugeschnitten, ähnlich wie Elastic App Search. Im Gegensatz zu App Search setzt Connected Search auf Solr als Search Engine und ist nur als SaaS verfügbar. Das bedeutet, es kann nicht lokal deployed werden. In großen Organisationen müssen häufig solche zentralen Bestandteile der Infrastruktur in der eigenen Umgebung deployed sein. Daher können diese Organisationen Connected Search nicht nutzen. Lucidworks stellt aktuell auch keine Möglichkeit bereit, Connected Search einfach zu testen. Als Unternehmen kann man eine Anfrage für ein Meeting mit Lucidworks stellen, in welchen Connected Search gezeigt werden soll. Alle Angaben über Connected Search basieren deshalb auf dem Marketingmaterial von Lucidworks[vgl. 22a].

Stärken

Durch die No Code Strategie ist es einfach, für Domain Experten und Content Curatoren Connected Search zu nutzen. Auch die Features „Keyword Extraction“ und „Automatic Boosting“ sind für diese Zielgruppe interessant.

Schwächen

Die größte Schwäche des Produkts ist die fehlende bzw. unzugängliche Dokumentation. Die No Code Strategie ist für einfache Anwendungsfälle gut geeignet, skaliert aber selten für Organisationen mit einer komplexen Suchfunktion.

Zusammenfassung

Connected Search unterscheidet sich in seinem Funktionsumfang nur unwesentlich von App Search und Quepid. Damit kann sich Connected Search als eine erfolgreiche Konkurrenz etablieren. Derzeit mangelt es dafür jedoch an einer zugänglichen Dokumentation, sowie einem Community Support. Außerdem unterscheidet sich Connected Search in seinen bekannten Features nur unwesentlich von den beiden anderen Produkten.

3.5 Technische Schnittstellen

3.5.1 Search Engines

Das RQMS muss direkt mit den Search Engines interagieren, daher wird eine Schnittstelle benötigt, über welche das RQMS mit der Search Engine kommuniziert. Für die Schnittstellen der Search Engines gibt es aktuell, keinen Standard für Anfragen. Im Gegensatz dazu nutzen relationale Datenbanken SQL, welches es ermöglicht Anfragen meist unabhängig von der Implementierung der relationalen Datenbank zu erstellen. Die auf Lucene basierenden Search Engines Elasticsearch und Solr haben die größte Funktionalitätsvielfalt und sind beide Open Source[vgl. Sch20, S. 9ff]. Aufgrund dieser Eigenschaften muss ein RQMS mindestens diese beiden Search Engines unterstützen. Eine Anbindung muss so abstrahiert sein, sodass später auch weitere Search Engines mit dem RQMS genutzt werden können.

3.5.2 Nutzerdatenanalyse

Damit das RQMS in seinem vollen Funktionsumfang arbeiten kann, werden die Nutzungsdaten der Suchfunktion benötigt. Diese Nutzungsdaten orientieren sich am Shopify Modell für „Search Facts“[vgl. Slo21]. Die Nutzungsdaten müssen mindestens die folgenden Informationen enthalten:

- Suchanfrage
- Ergebnisse
- Session ID
- Interaktionen mit Ergebnissen

Wenn möglich, sollten auch Nutzer anhand einer User-ID identifiziert werden können. Das RQMS benötigt eine Schnittstelle zum Empfangen dieser Daten. Da gerade bei großen Organisationen häufig Kafka schon genutzt wird (z. B. Shopify[Slo21], Netflix[HVA18], Zalando[Kel17]), sollte das RQMS mindestens eine Schnittstelle zu Kafka bieten. Auch diese Anbindung muss so abstrahiert sein, dass später weitere Datenquellen, wie z. B. SAP Hana, Microsoft Insights oder Adobe Analytics angebunden werden können.

3.5.3 Erweiterbarkeit durch Plugins

Die in Unterabschnitt 3.5.1 und Unterabschnitt 3.5.2 beschriebenen Schnittstellen müssen beide abstrahiert werden, damit später noch andere Systeme angebunden werden können. Die Anbindung von anderen Systemen soll aber nicht durch die Änderung am ursprünglichen Quellcode erfolgen, sondern durch das Verwenden von Plugins. Diese Strategie ermöglicht es Organisationen, mit wenig Aufwand ihre eigenen Systeme anzubinden. Da der originale Quellcode nicht verändert werden muss, kann eine Organisation ein Plugin erstellen, welches im RQMS mitläuft und die Schnittstellenfunktionalität übernimmt. Eine Implementierung im originalen Quellcode ist nicht immer möglich, da dieser gegebenenfalls Open Source gestellt werden muss (z. B. Lizenzbestimmungen), eine Organisation ihre Implementierung selbst pflegen muss, was erheblichen Aufwand verursachen kann, oder die Änderungen so verallgemeinert werden müssen, dass sie für alle Nutzer passend sind. Wenn Plugins verwendet werden, kann eine Organisation diese intern entwickeln, ohne sie mit anderen teilen zu müssen. Darüberhinaus wird der Aufwand erspart, die eigene Implementierung des RQMS pflegen zu müssen. Dabei können die Plugins spezifische Funktionen erfüllen und müssen nicht allgemeingültig entwickelt werden. Daher ist ein RQMS ohne Plugin System für viele Organisation nicht nutzbar.

4 Architekturvorschlag für ein Relevance Quality Management System

In diesem Kapitel wird eine Architektur für eine RQMS dargestellt. Als Rahmenwerk dieser Architektur wird Domain-Driven Design[Eva04] genutzt, daher beginnt dieses Kapitel mit einer kurzen Einführung in Domain-Driven Design, bevor die Architektur des RQMS erläutert wird. Die Architekturdokumentation benutzt englische und deutsche Begriffe, da das Architekturmodell auf Englisch implementiert werden soll. Soweit möglich, wird auf eine konsistente Verwendung geachtet, allerdings ist das aufgrund der deutschen Beschreibung nicht immer praktikabel.

4.1 Domain-Driven Design

Eric Evans definiert Domain-Driven Design wie folgt: „Domain-driven design is both a way of thinking and a set of priorities, aimed at accelerating software projects that have to deal with complicated domains“[Eva04, S. XXI]. Demnach soll Domain-Driven Design Software-Projekte dabei unterstützen, komplizierte Aufgabenumfelder abzubilden.

4.1.1 Ubiquitous Language

Für Eric Evans stellt dabei die im Projekt genutzte Sprache eine wichtige Grundlage dar. Diese Sprache muss drei Anforderungen erfüllen[Eva04, S. 23]:

1. Sie muss in ihrem Vokabular alle Namen der Klassen und wichtigen Operationen beinhalten.
2. Sie soll alle Begriffe, welche zum Diskutieren der Regeln des Domain Models benötigt werden, bereitstellen.
3. Sie enthält die Namen der im Team verwendeten Entwurfs- und Entwicklungsmuster.

Diese Sprache bezeichnet Eric Evans als „Ubiquitous language“. Er definiert diesen Begriff folgendermaßen: „A language structured around the domain model and used by all team members to connect all the activities of the team with the software“[Eva04, S. 517]. Das Domain Model stellt für ihn das Rückgrat dieser Sprache dar. Es ist auch wichtig, dass das Team, welches die Sprache verwendet, diese in allen Kommunikationen über die Software nutzt. Wenn es Situationen gibt, in welchen die Sprache nicht passt, müssen diese diskutiert werden und die Sprache muss entsprechend angepasst werden. Bei Änderung an der Ubiquitous Language muss gegebenenfalls das Domain Model angepasst werden, sodass die Sprache und das Domain Model immer konsistent sind. Alle Schwächen des Domain Models werden

somit offengelegt und können behoben werden. Bei Kommunikationen, die die Aktivitäten der Software nicht betreffen und somit außerhalb des Domain Models liegen, kann eine andere Sprache genutzt werden. Somit bleibt die „Ubiquitous language“ immer eng an das Domain Model gekoppelt. Dabei soll die Sprache nicht als eigenes Artefakt angesehen werden, sondern ein integraler Bestandteil aller Entwicklungsprozesse sein. Die Aufgabe der „Ubiquitous language“ ist es, die Kommunikation über die Design-Aspekte zu übernehmen, welche nicht im Source Code auftauchen[Eva04, S. 26ff].

4.1.2 Model-Driven Design

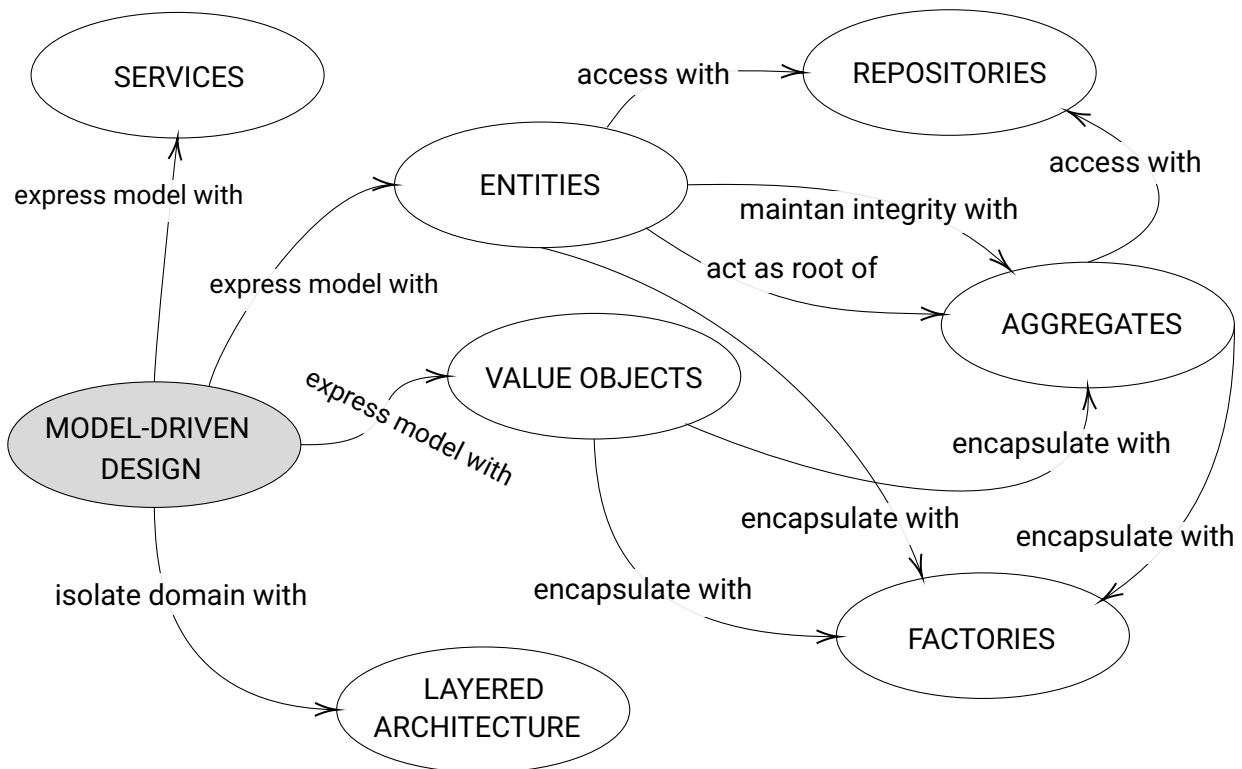


Abbildung 4.1: Eine Übersicht der im Model-Driven Design verwendeten Begriffe, aus *Domain-driven Design - Tackling Complexity in the Heart of Software*, S. 65[Eva04]

Im Prozess des Model-Driven Designs soll das Domain Model mit der Implementierung eng gekoppelt werden, sodass jede Änderung an der Implementierung auch eine Änderung des Domain Models bewirkt. Dabei werden alle notwendigen Rollen in einem Projekt einbezogen, sodass die Änderung ein neues Domain Model erzeugt, welches sowohl die reale Welt, als auch die technischen Anforderungen abbildet[Eva04, S. 64]. Nach Eric Evans lässt sich die Methodik des Model-Driven Designs wie folgt zusammenfassen: „A design in which some subset of software elements corresponds closely to elements of a model. Also, a process of co-developing a model and an implementation that stay aligned with each other“[Eva04, S. 513]. Das Design eines Domain Models ist nach Evans sehr individuell, die Überführung desselben in eine Implementierung kann jedoch systematisch erfolgen. Die wichtigsten dabei genutzten Muster und Konzepte sind in Abbildung 4.1 in ihrem Zusammenhang aufgelistet und werden

im Nachfolgenden einzeln erklärt. Das Domain Model wird dabei durch Entities, Value Objects und Services abgebildet. Die Abtrennung zwischen Domain Model und dem restlichen Anwendungscode erfolgt durch die Layered Architecture. Zur Erstellung von Entities und Value Objects sollen Factories genutzt werden. Die Grundlage für Aggregates bilden Entities, welche als Einheit im Programmkontext verwendet werden. Aggregates können durch Value Objects repräsentiert werden. Sowohl auf Entities als auch auf Aggregates kann durch ein Repository zugegriffen werden[Eva04, S. 64f].

Entity

Entities im Model-Driven Design bilden reale Objekte ab, welche eine eigene Identität haben (z. B. Personen, Gegenstände). Eric Evans definiert diese so: „An object fundamentally defined not by its attributes, but by a thread of continuity and identity“[Eva04, S. 512]. In der Implementierung sollten Objekte dann als Entity behandelt werden, wenn zwei Objekte, deren Attribute alle gleich sind, unterschieden werden müssen. Falls zwei Objekte, deren Attribute alle gleich sind, nicht mehr zu unterscheiden sind (z. B. Farbe, Betrag, Kategorie eines Konzerttickets), handelt es sich nicht um Entities. Im Allgemeinen sollten Entity Objekte eine einfache Klassendefinition besitzen und sich auf den Lebenszyklus und die Identität des Objekts konzentrieren. Software-Systeme können hierbei externe oder selbst erzeugte Kennungen nutzen, damit Objekte auseinandergehalten werden können. Ob es sich bei einem Objekt um eine Entity handelt oder nicht, hängt stark vom Kontext ab, in welchem dieses genutzt wird. Dabei müssen die Bedingungen aus der realen Welt betrachtet und übernommen werden[Eva04, S. 89ff]. Ein Beispiel ist die Zuordnung der Sitzplätze in der Deutschen Bahn. Ein Zugticket hat im Regionalverkehr keinen zugewiesenen Sitzplatz. Daher ist es unnötig Sitzplätze als Entity zu modellieren. Es ist natürlich sinnvoll zu wissen, wie viele Sitzplätze ein Zug hat, aber diese müssen nicht einzeln identifiziert werden. Im Fernverkehr hingegen ist es wichtig, einzelne Sitzplätze zu identifizieren, da hier zum einen Sitzplätze reserviert werden können und zum anderen einige Sitzplätze auch nur unter bestimmten Voraussetzungen genutzt werden können (z. B. Kinderabteil, Konferenzabteil, 1. Klasse). Daher ist es notwendig, jeden Sitzplatz einzeln zu identifizieren, auch wenn nicht jedes Ticket einen zugewiesenen Sitzplatz hat. In diesem Fall sollten die Sitzplätze als eigene Entities modelliert werden.

Value Object

Die Definition eines Value Objects nach Eric Evans ist: „An object that describes some characteristic or attribute but carries no concept of identity“[Eva04, S. 514]. Diese Objekte beschreiben dementsprechend Konzepte, welche keine eigene Identität besitzen. Als Beispiele werden Farben, Buchstaben und Zahlen aufgelistet. Im Code einer Anwendung ist es irrelevant, um welche „3“ bzw. um welches „Q“ es sich handelt[Eva04, S. 97f]. Einige Software-Systeme können durch die Nutzung von Value Objects enorme Performanceoptimierungen erzielen (z. B. `std::string` in `libc++`[Lai20]). In einer Implementierung sollten Value Objects immer dann eingesetzt werden, wenn nur die Attribute eines Objekts genutzt werden. Im Allgemeinen sind Value Objects nicht veränderlich. Sollte ein Attribut einen neuen Wert annehmen, dann muss eine neue Instanz für das Value Object erzeugt werden. Ein Value Object

soll einzelne Attribute sinnvoll zusammenfassen (z. B. Straße, Hausnummer, Postleitzahl und Stadt werden zu einer Adresse zusammengefasst)[Eva04, S. 99].

Service

Ein Service im Domain Layer einer nach Model-Driven Design implementierten Anwendung muss stark von der Definition eines Service in der Anwendungsschicht unterschieden werden. Eric Evans definiert den Begriff so: „An operation offered as an interface that stands alone in the model, with no encapsulated state“[Eva04, S. 513]. Ein Service bietet also eine Aktion bzw. Operation an, welche zustandslos ausgeführt und nicht direkt einer Entity oder einem Value Object zugeordnet werden kann. Mit zustandslos ist hier gemeint, dass jede Instanz dieselbe Aktion ausführt wie alle anderen Instanzen, unabhängig davon, welche Operationen davor auf dieser Instanz ausgeführt wurden. Die Aufgabe eines Service im Domain Layer ist also vor allem das Kapseln von Aktionen und Operationen, welche mehr als ein Objekt betreffen und weder einer Entity noch einem Value Object zugeordnet werden können[Eva04, S. 104ff]. Ein Beispiel für eine solche Aktion ist eine Umbuchung bei einer Bank. Der Service ruft hierbei zwar nur die Aktionen an den Bankkonto-Entities auf, kapselt aber so den Prozess und bietet die Möglichkeit, Entity-übergreifende Regeln zu etablieren (z. B. manuelle Überprüfung aufgrund von Betrag und Historie).

Aggregate

Die Definition eines Aggregates nach Eric Evans ist: „A cluster of associated objects that are treated as a unit for the purpose of data changes. External references are restricted to one member of the aggregate, designed as the root. A set of consistency rules applies within the aggregates boundaries“[Eva04, S. 511]. Als spezielle Art von Entity, bzw. als Kapsel um eine Entity, ermöglichen Aggregates die Verwaltung von Abhängigkeiten zwischen Entities und Value Objects. Alle Zugriffe auf die beinhalteten Entities und Value Objects erfolgen durch den Root des Aggregates, also der Entity selbst. Ein Aggregate kann dabei ein eigenes Objekt sein, welches die Root Entity und alle abhängigen Objekte beinhaltet. Durch die Nutzung von Aggregates kann sichergestellt werden, dass alle Invarianten erfüllt sind, weil jede Änderung an der Root Entity oder den Value Objects nur durch das Aggregate erfolgen kann[Eva04, S. 125ff]. Ein Beispiel für ein Aggregate ist ein E-Bike. Ein E-Bike hat neben der Batterie auch einen Motor und eine Bedieneinheit. Die Batterie ist eine eigenständige Entity, da sie ersetzt oder zwischen Fahrrädern getauscht werden kann. Der Motor und die Bedieneinheit sind mit dem Fahrrad verknüpft und können nur durch einen Fachhändler getauscht werden. Diese beiden Komponenten können als Aggregate mit dem E-Bike als Root Entity modelliert werden. Wenn der Fahrer des E-Bikes die Supportstufe ändern will, kann die E-Bike Entity sicherstellen, dass sowohl im Motor als auch in der Bedieneinheit die gleiche Supportstufe gesetzt ist.

Factory

Eine Factory ist definiert als: „A mechanism for encapsulating complex creation logic and abstracting the type of a created object for the sake of a client“[Eva04, S. 512]. Wenn komplexe

Entities, Value Objects oder Aggregates erstellt werden müssen, ist es in der Implementierung übersichtlicher, die Logik zur Erstellung dieser Objekte aus der jeweiligen Klasse auszulagern. Dabei ist diese Logik immer noch Bestandteil des Domain Models, da auch bei der Erstellung der Objekte alle Invarianten und Geschäftsregeln eingehalten werden müssen. Außerdem hilft die Verwendung von Factories dabei, dass der Client nicht mehr die exakten Klassen kennen muss, um diese zu instanziiieren, sondern abstrakte Datentypen (z. B. Java Interfaces) nutzen kann. Für einfache Objekte sollte aber der Klassenkonstruktor direkt verwendet werden, um unnötige Komplexität zu vermeiden[Eva04, S. 136ff].

Repository

Das Durchschreiten von Objektstrukturen ist eine Möglichkeit, um Objekte zu finden. Allerdings wird auch hierfür das erste initiale Objekt benötigt, von welchem aus alle anderen Objekte erreicht werden können. Dieses erste Objekt kann zum Beispiel von einem Repository bereitgestellt werden. Damit das Domain Model aber nur die fachlich notwendigen Relationen beinhaltet, sollte ein Repository statt technisch notwendiger Relationen genutzt werden, wenn auf eine Entity, ein Value Object oder ein Aggregate zugegriffen werden muss. Somit werden unnötige Relationen im Domain Model vermieden und die Persistenzschicht kann einfach ausgelagert werden[Eva04, S. 147ff]. Eric Evans definiert das Repository wie folgt: „A mechanism for encapsulating storage, retrieval, and search behaviour which emulates a collection of objects“[Eva04, S. 513]. Damit Regeln und die Logik nicht in Queries ausgelagert werden, sondern in Entities, Value Objects und Aggregates verbleiben, ist es wichtig, dass das Repository keinen direkten Zugriff auf die Persistenzschicht erlaubt. Das Repository kann aber Value Objects und Aggregates bereitstellen, welche durch eine Anfrage an die Persistenzschicht erzeugt wurden. Diese sind damit Bestandteil des Domain Models und Logik wird nicht in die Anwendungsschicht bzw. Infrastrukturschicht ausgelagert. Invarianten und Regeln eines Aggregates können somit aufrechterhalten werden. Repositories sollten nur für Aggregates und Entities ohne sonstige Relationen erstellt werden. Auf alle anderen Objekte muss dann durch Relationen aus Aggregates oder Entities zugegriffen werden, welche über Repositories bereitgestellt werden[Eva04, S. 147ff].

4.1.3 Layered Architecture

Generell ist es üblich, ein Software-System in unterschiedliche Schichten aufzuteilen. Häufig wird die three-tier architecture[Fer+08] genutzt, welche ein System in Präsentation-, Anwendungs- und Datenhaltungsschicht aufteilt. Im Domain Driven Design wird die Layered Architecture mit anderen Schichten genutzt. Die Methode der Layered Architecture im Domain-Driven Design ist wie folgt definiert: „A technique for seperating the concerns of a software system, isolating a domain layer, among other things“[Eva04, S. 513]. Dabei werden vier Schichten definiert, welche aber teilweise auch noch erweitert oder weggelassen werden können, solange eine dedizierte Schicht bestehen bleibt, die nur das Domain Model beinhaltet[Eva04, S. 68ff]. Die Notwendigkeit, ein Software-System in verschiedene Schichten zu unterteilen, kommt durch deren Komplexität zustande. Die Unterteilung in mehrere Schichten ermöglicht das Aufteilen von Verantwortlichkeiten in der Anwendung. Im Domain Driven

Design hat jede Schicht die Verantwortlichkeit für einen technischen Aspekt. Außerdem darf jede Schicht nur mit den darunterliegenden Schichten direkt kommunizieren. Eine Kommunikation mit den übergeordneten Schichten ist nur indirekt möglich (z. B. Observer Pattern). Jede Schicht soll in sich abgeschlossen und zusammenhängend sein, damit die dem Code zugrundeliegende Logik möglichst einfach zu erkennen ist[Eva04, S. 68ff]. Ein Beispiel ist die Trennung der Schichten in einem Zugticketbuchungssystem. Die Präsentationsschicht (Presentation Layer) enthält die Website und alle technischen Schnittstellen, welche von anderen Systemen angebunden werden. Die Anwendungsschicht (Application Layer) implementiert die Authentifizierung der Nutzer (das Nutzermanagement wird von einem anderen System durchgeführt und ist daher nicht Teil des Domain Models) und kann Anfragen des Benutzers an die Domain Logik weiterreichen. Die Domain Logik (Domain Layer) führt die Buchung durch und stellt die Informationen über verfügbare Sitzplätze etc. bereit. Die Infrastrukturschicht (Infrastructure Layer) ist für die Verbindung zur Datenbank, Message Queues und weiteren technischen Systemen verantwortlich. Im Gegensatz zur Anwendungsschicht besitzt die Infrastrukturschicht keine Informationen über den Inhalt und die Struktur der Daten, welche sie persistiert bzw. weitergibt. Inhalt und Struktur sind Verantwortlichkeiten der Domainschicht bzw. der Anwendungsschicht.

Präsentationsschicht

Die Präsentationsschicht ist dafür verantwortlich, den Benutzern Informationen anzuzeigen bzw. die Informationen an externen Systeme weiterzugeben. Außerdem muss diese Schicht die Eingaben des Nutzers bzw. des externen Systems annehmen, die Verarbeitung starten und eine Antwort zurückgeben[Eva04, S. 70].

Anwendungsschicht

Die Verantwortlichkeit der Anwendungsschicht ist es, die Aufgaben des Software-Systems zu koordinieren, mit anderen Systemen zu kommunizieren und die unterliegenden Schichten zu verwalten. Dabei darf diese Schicht keinen Zustand über das Domain Model besitzen und sollte möglichst klein gehalten werden. Allerdings darf sie technische Zustände besitzen und verwalten (z. B. den Fortschritt einer Aufgabe, welcher in der Domainschicht ausgeführt wird)[Eva04, S. 70].

Domainschicht

Die Domainschicht soll die Anforderungen, Konzepte und Regeln der modellierten Prozesse implementieren. Dabei muss der Zustand in dieser Schicht dem Zustand der modellierten Situation entsprechen. Die technische Grundlage wie dieser Zustand persistiert wird, muss aber in die Infrastrukturschicht ausgelagert werden[Eva04, S. 70].

Infrastrukturschicht

Diese Schicht beinhaltet die technischen Fähigkeiten, um die überliegenden Schichten zu unterstützen. Am wichtigsten ist hier die Fähigkeit zur Persistierung von Zuständen. Außerdem

kann diese Schicht auch die technische Grundlage enthalten, um die Kommunikation zwischen den Schichten zu ermöglichen (z. B. Observer Pattern)[Eva04, S. 70].

4.1.4 Bounded Context

In einem großen Projekt existieren häufig mehrere Domain Models parallel. Damit die Software einfach zu verstehen, zuverlässig und fehlerfrei bleibt, sollten diese Domain Models explizit voneinander getrennt werden[Eva04, S. 335ff]. Diese Trennung wird im Domain Driven Design als Bounded Context bezeichnet und ist wie folgt definiert: „The delimited applicability of a particular model. Bounded contexts give team members a clear and shared understanding of what has to be consistent and what can develop independently“[Eva04, S. 511]. Gerade wenn unterschiedliche Funktionalitäten durch das Domain Model abgebildet werden müssen, ist es hilfreich, das Domain Model in mehrere Bounded Contexts aufzuteilen und entsprechend für die jeweilige Funktionalität anzupassen[Eva04, S. 335ff]. Ein Beispiel für Bounded Contexts ist die Sitzplatzverwaltung der Deutschen Bahn. Da die einzelnen Sitzplätze im Nahverkehr keine Rolle spielen, aber im Fernverkehr ein wichtiger Bestandteil sind, kann es Sinn machen, im Buchungssystem diese beiden Funktionen in jeweils eigene Bounded Contexts auszulagern. Zwar muss es die Möglichkeit geben, eine Verbindung zu buchen, welche sowohl Nah- als auch Fernverkehr beinhaltet, aber die Sitzplatzreservierung spielt nur im Fernverkehr eine Rolle. Dadurch lässt sich das Domain Model für den Nahverkehr vereinfachen und es bietet sich die Möglichkeit, dort den Fokus auf andere Funktionen zu legen (z. B. Vorschläge für Alternativrouten aufgrund von Auslastung oder wahrscheinlichen Verspätungen). Beim Fernverkehr hingegen ist es wichtig, dass alle Reisenden auch mit dem gebuchten Zug fahren und sich idealerweise auf ihrem reservierten Sitzplatz befinden. Da auch ICEs Kapazitätsgrenzen haben, sollte eine Überbuchung im Vorhinein ausgeschlossen werden. Wenn die Kapazitätsgrenzen überschritten werden, darf der Zug nicht fahren. Dieser Unterschied in beiden Anwendungsfällen sollte auch im Domain Model wiederzufinden sein. Der Einsatz durch Bounded Contexts erlaubt es, diese explizit und einfach abzubilden.

4.2 Ubiquitous Language

4.2.1 Domain Model Begriffe

Offline-Metriken

Offline-Metriken sind eine Gruppe von Metriken, welche erfasst werden, indem Tests gegen eine Instanz einer Suchmaschine ausgeführt werden. Die Tests benötigen keine Interaktion von Endnutzern der Suchfunktion. Diese Gruppe wird auch als „Offline Metrics“ bezeichnet.

Online-Metriken

Online-Metriken sind eine Gruppe von Metriken, welche erfasst werden, indem das Verhalten von Endnutzern der Suchfunktion überwacht wird. Die Endnutzer können hierbei als Gesamtheit betrachtet oder in mehrere Teilgruppen unterteilt werden. Diese Gruppe wird auch als „Online Metrics“ bezeichnet.

Precision

Precision ist eine Metrik, welche ausdrückt, wie viele relevante Dokumente in der Menge aller gefundenen Dokumente existieren.

$$precision = \frac{\text{Anzahl gefundener relevanter Dokumente}}{\text{Anzahl aller gefundenen Dokumente}} \quad (4.1)$$

Rank

Der Rank ist die Position eines Dokuments in der Ergebnisliste. Dabei ist irrelevant auf welcher Seite das Dokument angezeigt wird, die Zählung des Ranks beginnt immer auf der ersten Seite mit dem ersten Dokument.

Recall

Recall ist eine Metrik, welche ausdrückt, wie viele Dokumente aus der Menge der relevanten Dokumente gefunden wurden.

$$recall = \frac{\text{Anzahl gefundener relevanter Dokumente}}{\text{Anzahl aller relevanten Dokumente}} \quad (4.2)$$

Relevance Case

Ein Relevance Case ist ein Testfall für die Relevanz einer Suchanfrage, welcher auch die Liste der erwarteten Ergebnisse enthält. Dieser kann auch als Judgment List bezeichnet werden. Der Begriff Relevance Case wurde aus Unterabschnitt 3.4.1 übernommen.

Relevance Score

Der Relevance Score ist eine Zahl zwischen 0 und 1, welche darstellt, wie gut ein Relevance Case erfüllt wird. Dieser Wert wird mithilfe von Metriken berechnet (z. B. DCG, ERR[Sch20, S. 14ff]).

Search Fact

Der Search Fact ist eine von Shopify (siehe Unterabschnitt 2.1.4) adaptierte Datenstruktur, welche eine Suche eines Nutzers abbildet. Dabei ist die Suche erst dann abgeschlossen, wenn der Nutzer mit einem Ergebnis interagiert, ohne in die Ergebnisliste dieser Suche zurückzukehren. Wenn der Nutzer erneut sucht und gar nicht mit einem Ergebnis interagiert, ist die Suche auch abgeschlossen und ein Search Fact wird erstellt. Ein Search Fact beinhaltet den Suchbegriff des Nutzers, die erzeugten Ergebnisse der Suche, alle Interaktionen mit den Ergebnissen und die jeweilige Position des Ergebnisses in der Ergebnisliste.

Suchbegriff

Ein Suchbegriff ist die Eingabe eines Nutzers in die Suchfunktion. Dabei muss der Suchbegriff je nach Art der Suchfunktion mindestens einen Buchstaben beinhalten. Ein Suchbegriff darf auch aus mehreren Wörtern bestehen.

4.2.2 Bounded Contexts

Alle im RQMS verwendeten Bounded Contexts haben einen zentralen Aspekt, nach dem diese benannt wurden. Wenn mehrere zentrale Aspekte in einem Bounded Context vorhanden wären, sollte dieser entsprechend aufgeteilt werden. Jede Kommunikation zwischen den Bounded Contexts muss über die Anwendungsschicht erfolgen. Die Benennung der Module der Anwendungsschicht orientiert sich dementsprechend auch an den Namen der Bounded Contexts.

Offline

Der Offline Context bezieht seinen Namen aus den Offline-Tests bzw. Offline-Metriken. Der Begriff Offline-Metrik wurde von Shopify genutzt (siehe Unterabschnitt 2.1.4) und beschreibt Metriken welche ohne Nutzerinteraktion erstellt werden können. Im RQMS werden alle Offline-Metriken durch Offline-Tests erzeugt. Da diese Metriken und Tests der zentrale Aspekt dieses Bounded Contexts sind, leitet sich hiervon der Name ab.

Online

Der Online Context bezieht seinen Namen aus den Online-Metriken und beschreibt Metriken, welche durch Nutzerinteraktion erstellt werden. Der Begriff Online-Metrik wurde ebenfalls von Shopify genutzt (siehe Unterabschnitt 2.1.4) und wird für den weiteren Verlauf übernommen. Im RQMS werden alle Online-Metriken aus den Daten des Search Logs und der Click Streams erzeugt, indem diese zu Search Facts zusammengefasst werden. Da die passive Erfassung der Search Facts und die automatische Erstellung der Online-Metriken der zentrale Aspekt dieses Bounded Contexts ist, leitet sich hiervon ebenfalls der Name ab.

Integration

Im Integration Bounded Context geht es darum, die Datenquellen und Suchfunktionen zu abstrahieren. Das RQMS soll mit verschiedensten Datenquellen und Suchfunktionen arbeiten können. Diese Funktionalität soll dynamisch von Plugins bereitgestellt werden. Im Domain Model soll für diese Anbindung Business-Logik implementiert werden, sodass Plugins automatisch neue Suchfunktionen und Datenquellen registrieren können, welche dann in anderen Bereichen des RQMS wiederverwendet werden. Besonders die generische Anbindung von Datenquellen ist wichtig, da es hier keinen Standard gibt und sich diese je nach Organisation unterscheiden. Die Integration dieser Datenquellen und Suchfunktionen ist also der Hauptaspekt dieses Bounded Contexts. Daher leitet sich von dort der Name ab.

4.2.3 Organisatorische Begriffe

Ein RQMS ist stark in eine Organisation eingebunden und hat daher auch einen großen organisatorischen Kontext. Im Folgenden werden einige Begriffe explizit in diesem Kontext definiert. Diese Begriffe wurden bereits in Kapitel 2 und Kapitel 3 verwendet, jedoch ohne sie explizit im Kontext eines RQMS zu definieren.

Content Curator

Der Content Curator hat die inhaltliche Verantwortung die Relevanz der Suchfunktion zu verbessern. Dabei benötigt der Content Curator ein tiefes Domänenwissen und eine gute Kommunikation mit den Domain Experts. Der Content Curator hat die Aufgabe, die Relevanz der Suchergebnisse anhand des Nutzerverhaltens zu bewerten und inhaltliche Verbesserungspotenziale zu erkennen. Diese setzt er gemeinsam mit einem Relevance Engineer um.

Domain Expert

Ein Domain Expert ist eine Person in der Organisation, welche ein tiefes Wissen über die Domäne der Suchfunktion besitzt. In einem E-Commerce-Umfeld sind es z. B. Category Manager, im Umfeld von Streaming-Diensten sind es z. B. die Editoren, welche die Inhalte für den Streaming-Dienst auswählen. Ein Domain Expert hat neben dem fachbereichsspezifischen Know-How auch noch Erfahrung, um etablierte Begriffe und Handlungsweisen zu erklären[TB16, S. 265].

Relevance Engineer

Der Relevance Engineer hat die technische Verantwortung, die Relevanz der Suchfunktion zu verbessern. Dabei erfasst ein Relevance Engineer mit Unterstützung des Content Curators die Anforderungen der Endnutzer, die der Organisation und implementiert diese mit Hilfe einer Search Engine. Bei Verbesserungsvorschlägen stimmen sich Relevance Engineer und Content Curator ab und implementieren diese gegebenenfalls auch gemeinsam.

4.2.4 Technische Begriffe

Im RQMS werden einige technische Begriffe verwendet, die nicht nur für die Entwicklung benötigt werden. Zwar kann die Domainschicht getrennt von den anderen Schichten der Architektur entwickelt werden, sie verwendet jedoch auch grundlegende Strukturen der Informatik, der Datenhaltung und der Datenklassifizierung. Daher müssen diese Begriffe auch in die Ubiquitous Language aufgenommen werden.

Labels

Labels sind eine Möglichkeit, verschiedene Objekte zu kennzeichnen. Sie besitzen immer einen Schlüssel und optional einen Wert. Ein zu kennzeichnendes Objekt kann beliebig viele Labels besitzen, allerdings immer nur ein Label pro Schlüssel. Somit kann eine Menge von Objekten einfach gefiltert werden.

Search Engine

Eine Search Engine repräsentiert eine bestimmte Konfiguration einer Suchfunktion. Dabei kann es sich zum Beispiel um einen Index in einem Elasticsearch Cluster handeln oder auch um eine eigene Anwendung, welche die Suchfunktion bereitstellt.

Plugin

Ein Plugin ist eine Erweiterung der Funktionalität des RQMS, welches dynamisch geladen werden kann und nicht zwingend mit dem RQMS ausgeliefert werden muss. Plugins können die Anbindungen zu Search Engines oder Datenquellen für Search Facts beinhalten.

Map

Eine Map ist ein Objekt, welches Schlüssel-Werte zuordnet. Häufig sind die Schlüssel Texte, aber theoretisch kann jedes Objekt ein Schlüssel in einer Map sein. Ein Beispiel ist der Barcode Scanner in einem Supermarkt, welcher jedem Barcode ein Produkt zuordnet. Dabei kann der Barcode Scanner eine Map nutzen, deren Schlüssel Barcodes sind und deren Werte Objekte sind, welche die Produkte repräsentieren.

4.3 Architekturschichten

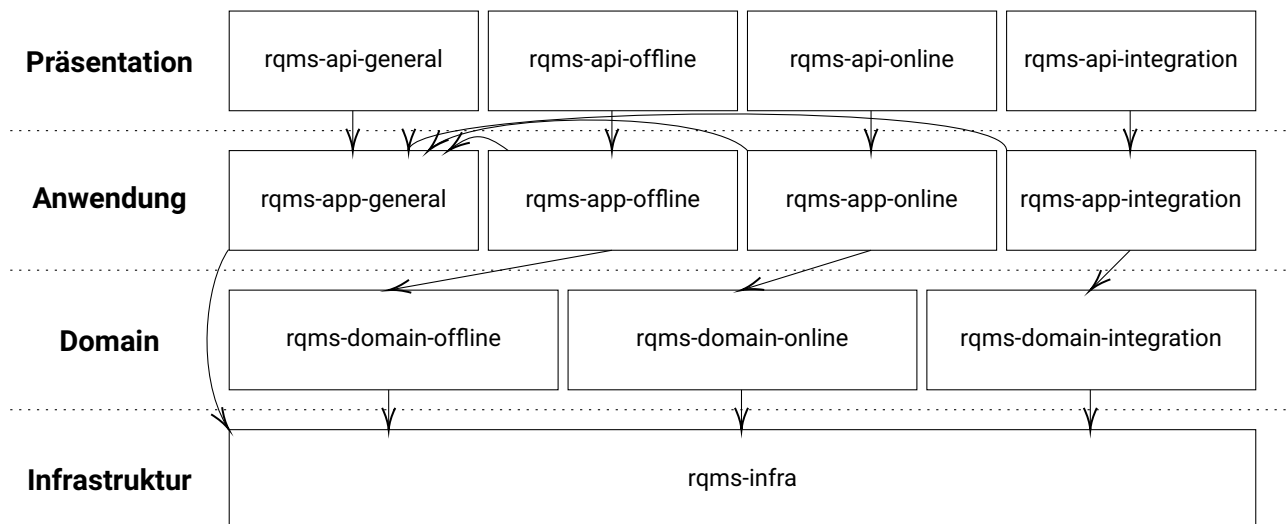


Abbildung 4.2: Übersicht der Architekturschichten und Module des RQMS

Das RQMS implementiert die Schichtenarchitektur des Domain Driven Design. Die dabei genutzten Schichten sind die Präsentationsschicht, die Anwendungsschicht, die Domainschicht und die Infrastrukturschicht. Außerdem werden drei Bounded Contexts genutzt: der Offline-Kontext, welcher das Domain Model für die Offline-Tests und Metriken enthält, der Online-Kontext, welcher das Domain Model für die Online-Tests und Metriken enthält und der Integration-Kontext, welcher das Domain Model für die Integration von Datenquellen und Suchmaschinen enthält. Für jeden Kontext werden in der Präsentationsschicht, der Anwendungsschicht und der Domainschicht eigene Module genutzt, welche sicherstellen, dass die Datenmodelle nicht geteilt werden. In der Präsentationsschicht und in der Anwendungsschicht gibt es außerdem noch ein Modul für allgemeine Aufgaben. Dort werden alle Bestandteile ausgelagert, welche der jeweiligen Schicht zugeordnet sind, aber nicht in einen der Bounded Contexts eingruppiert werden können. Das betrifft vor allem technische Bestandteil des Sys-

tems (z. B. zeitbasierte Aufrufe oder technische Fehlerbehandlung). Die Module des RQMS und die Zuordnung in die jeweilige Schicht ist in Abbildung 4.2 dargestellt. Außerdem enthält die Abbildung die Abhängigkeiten zwischen den Modulen, welche durch Pfeile dargestellt sind. Diese Abhängigkeiten werden in den folgenden Abschnitten genauer erläutert.

4.3.1 Präsentationsschicht

Die Präsentationsschicht enthält nicht nur das Nutzerinterface des RQMS, sondern auch die Anwendungsschnittstelle (API), welche von anderen Systemen angesprochen werden kann. Es gibt hier auch Funktionen und Informationen, die keinem Bounded Context zugeordnet werden können, diese meist technischen Funktionen betreffen nur die Anwendung direkt (z. B. Zustand der Anwendung, Verbindung zur Datenbank, Ressourcenverbrauch). Diese Funktionen werden im Modul „rqms-api-general“ bereitgestellt, welches direkt vom Modul „rqms-app-general“ der Anwendungsschicht abhängt. Die anderen drei Module dieser Schicht sind jeweils einem Bounded Context zugeordnet und von den entsprechenden Modulen der Anwendungsschicht direkt abhängig.

4.3.2 Anwendungsschicht

In der Anwendungsschicht ist die technische Logik des RQMS enthalten. Dazu zählen unter anderem alle zeitbasierten Aktionen, die technische Fehlerbehandlung, Funktionen zur Fehlertoleranz und Widerstandsfähigkeit [Jam+18, S. 29], welche im Modul „rqms-app-general“ implementiert sind. Alle anderen Module dieser Schicht stellen den technischen Rahmen für die Interaktion mit dem Domain Model bereit. Jedes der drei Module dieser Schicht ist jeweils einem Bounded Context zugeordnet und von den entsprechenden Modulen der Domainschicht direkt abhängig. Außerdem sind diese drei Module vom „rqms-app-general“ abhängig, welches die geteilte Funktionalität bereitstellt und indirekte Kommunikation zwischen den anderen Modulen ermöglicht. Dieses Modul hängt direkt vom „rqms-infra“-Modul der Infrastrukturschicht ab.

Eine indirekte Kommunikation ist zwischen den Modulen „rqms-app-offline“ und „rqms-app-integration“ notwendig. Die im „rqms-app-integration“-Modul erfassten Daten werden benötigt, um automatisch neue Relevance Cases im „rqms-app-offline“ generieren zu können. Statt die Domain Models voneinander abhängig zu machen und somit die klare Trennung der Bounded Contexts aufzuheben, ist diese Abhängigkeit in der Anwendungsschicht implementiert. Alle Daten, die ausgetauscht werden müssen, werden durch diese Schicht weitergegeben und dabei entsprechend transformiert. Jeder Bounded Context in der Domainschicht bleibt dabei frei von Abhängigkeiten.

Eine weitere indirekte Kommunikation ist zwischen den Modulen „rqms-app-online“ und „rqms-app-integration“ notwendig. Aus den im „rqms-app-integration“-Modul erfassten Daten müssen Search Facts erstellt werden, die im „rqms-app-online“ verwendet werden, um Metriken und Testauswertungen zu erstellen. Statt die Domain Models voneinander abhängig zu machen und somit die klare Trennung der Bounded Contexts aufzuheben, ist diese Abhängigkeit in der Anwendungsschicht implementiert. Alle Daten, die ausgetauscht werden

müssen, werden durch diese Schicht weitergegeben und dabei entsprechend transformiert. Jeder Bounded Context in der Domainschicht bleibt dabei frei von Abhängigkeiten.

4.3.3 Domainschicht

Die Domainschicht beinhaltet das Domain Model, welches in Abschnitt 4.4 im Detail dargestellt wird. Für jeden Bounded Context existiert jeweils ein Modul. Jedes Modul in dieser Schicht hängt vom „rqms-infra“ Modul der Infrastrukturschicht ab.

4.3.4 Infrastrukturschicht

Die Infrastrukturschicht enthält alle technischen Grundlagen, welche von der Anwendung benötigt werden. In dieser Schicht werden die Datenbankverbindung, Anbindung an Message Queues und vieles mehr konfiguriert und den anderen Schichten zur Verwendung bereitgestellt. Da die Funktionalität des RQMS durch Plugins erweiterbar ist, muss die Infrastrukturschicht den Mechanismus zum Laden von Plugins beinhalten. Die von den Plugins benötigte Infrastruktur (z. B. Message Queues, Search Engines) müssen vom Plugin selbst konfiguriert und bereitgestellt werden. Daher enthält die Infrastrukturschicht auch nur ein allgemeines Modul. Alle Funktionalitäten, welche aufteilbar sind, müssen in die Plugins ausgelagert werden.

Search Engine Interface

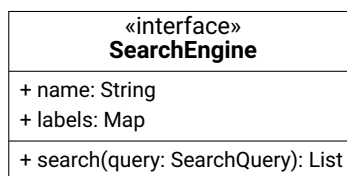


Abbildung 4.3: Search Engine in der Infrastrukturschicht

Das Search Engine Interface ist Teil der Infrastrukturschicht und repräsentiert die Konfiguration einer Suchfunktion. Dabei kann dies einem Index in Elasticsearch entsprechen oder eine eigene Anwendung sein, welche wiederum eine Search Engine benutzt. Eine Implementierung muss daher vom jeweiligen Plugin bereitgestellt werden. Diese muss dann die „search“-Methode implementieren, welche eine Liste von Suchergebnissen zurückgibt, damit zum Beispiel Relevance Cases (siehe Entity - Relevance Case) ausgeführt werden können. Dabei nimmt die Methode ein „SearchQuery“-Objekt entgegen und gibt eine Liste von „SearchResult“-Objekten zurück. Eine Search Engine ist durch eine eindeutige Bezeichnung identifizierbar. Durch das Zuweisen von Labels kann eine Instanz einfach aus der Menge aller existierenden Instanzen gefiltert werden. Das Klassendiagramm Abbildung 4.3 zeigt das Interface.

Search Query Interface

Das Search Query Interface definiert einen minimalen Search Query. Dieser muss einen Suchbegriff und optional Parameter enthalten. Außerdem muss eine Implementierung dieses In-

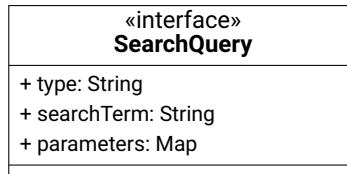


Abbildung 4.4: Search Query in der Infrastrukturschicht

Interfaces einen eindeutigen Bezeichner im „type“-Attribut hinterlegen, sodass die Implementierung eindeutig identifiziert werden kann. Plugins können dieses Interface implementieren und bereitstellen, somit können auch weitere spezifische Funktionen umgesetzt werden (z. B. Vorlagen für Suchanfragen mit Parametrisierung). Das Interface ist mit seinen Attributen in Abbildung 4.4 dargestellt.

Search Result Interface

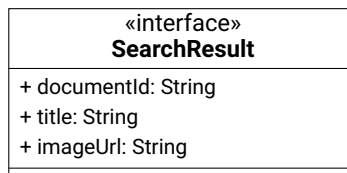


Abbildung 4.5: Search Result in der Infrastrukturschicht

Das Search Result Interface definiert die Attribute eines Ergebnisses, welches vom Search Engine Interface bei einer Suche zurückgegeben wird. Daher ist die Implementierung dieses Interfaces abhängig von der genutzten Suchfunktion und muss vom Plugin bereitgestellt werden. Das Attribut „documentId“ identifiziert ein Dokument in einer Search Engine eindeutig. Die Attribute „title“ und „imageUrl“ werden im User Interface genutzt, damit ein Nutzer Dokumente leichter identifizieren kann. Das „imageUrl“-Attribut ist optional, das „title“-Attribut muss immer gesetzt sein. Das Interface ist mit seinen Attributen in Abbildung 4.5 dargestellt.

Search Query Factory

Die Search Query Factory ermöglicht das Erstellen der Implementierungen des „SearchQuery“-Interfaces. Dabei müssen bei der Erstellung die Typ-Bezeichnung der Implementierung, der Suchbegriff und die Parameter übergeben werden. Die Parameter sind dabei optional. Das heißt, dass sie zwar übergeben werden müssen, jedoch leer sein dürfen. Daraus kann die Search Query Factory die Implementierung erzeugen, welche dann zum Beispiel für eine Suche an einer „SearchEngine“ genutzt werden kann. Alle Plugins können an der Search Query Factory ihre Implementierungen registrieren, sodass diese an allen Stellen im RQMS verwendet werden können.

4.4 Bounded Contexts - Domainschicht

Die Architektur für das RQMS besteht aus drei Bounded Contexts. Der Offline-Kontext ist der Kontext, welcher Informationen und Abläufe, die unabhängig von Nutzerinformationen der Suchfunktion sind, enthält. Der Online-Kontext ist der Kontext, welcher die Nutzerinformationen und ihre Auswertungen enthält. Der Integration-Kontext ist der Kontext, welcher die Schnittstellen für das Anbinden von externen Datenquellen und Suchfunktionen enthält. Diese Kontexte sind in der Realität nicht vollständig unabhängig, aber in dieser Schicht vollkommen getrennt. Die Integration der Information aus den Kontexten geschieht in der Anwendungsschicht, weshalb eine Trennung in der Domainschicht technisch möglich ist. Durch die starke Trennung ist eine parallele Entwicklung dieser Kontexte möglich, was die Implementierungsdauer verringern sollte und die Implementierung in jedem Kontext vereinfacht.

4.4.1 Domain Model - Offline

Das Domain Model für Offline-Metriken und Tests verwaltet Testfälle zur Überprüfung der Relevanz einer Suchfunktion. Diese Testfälle werden Relevance Case genannt. Mithilfe der Relevance Cases können Metriken erfasst werden, welche die Qualität der Suchfunktion widerspiegeln. Das Domain Model enthält außerdem Logik, um automatisiert neue Relevance Cases zu erstellen oder diese anzupassen, wenn sie vom Nutzerverhalten abweichen. Die Daten für das Nutzerverhalten werden von der Anwendungsschicht hereingereicht. Das zentrale Datenmodell dieses Bounded Context ist der Relevance Case, dieser wird im Abschnitt Aggregation - Relevance Case detailliert beschrieben.

Entity - Assertion Test

Ein Assertion Test ist eine spezielle Art von Test, bei welchem nur ein binäres Ergebnis entsteht. Diese Art von Tests werden verwendet, um sicherzustellen, dass eine Suchfunktion richtig konfiguriert wurde und notwendige Daten enthält. Ein Assertion Test besteht aus einer Suchanfrage und mindestens einer Bedingung, welche durch ein kurzes Stück Code ausgedrückt wird. Dieser Code kann auf den Inhalt der Ergebnisliste zugreifen und muss ein binäres Ergebnis liefern. Ein Beispiel für einen Assertion Test ist eine Suchanfrage nach dem Begriff „Milch“, deren Bedingung es ist, dass mindestens ein Ergebnis in der Ergebnisliste Titel das Wort „Milch“ enthält.

Value Object - Assertion Condition

Die Assertion Condition beinhaltet die Bedingungen eines Assertion Tests. Sie wird durch ein Stück Code ausgedrückt und besitzt keinen Zustand. Die Sprache, in welcher dieser Code geschrieben wird, ist von der Implementierung festzulegen.

Aggregation - Assertion Test

Die Aggregation für den Assertion Test kapselt die Assertion Test Entity mit ihren Conditions. Die Klassen dieser Aggregation sind in Abbildung 4.6 dargestellt. Eine wichtige Invariante

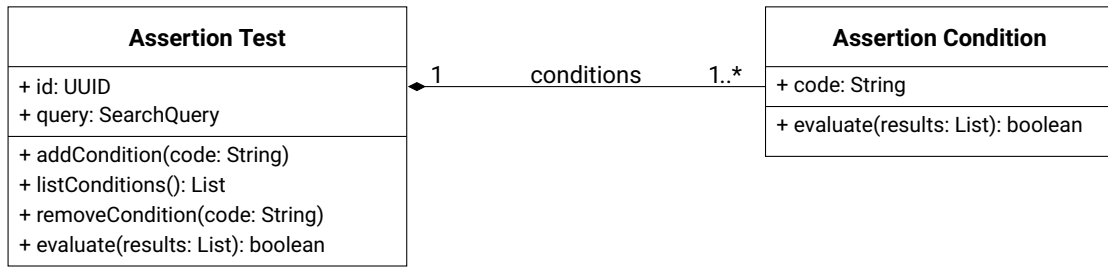


Abbildung 4.6: Aggregation - Assertion Test in der Offline Domain

dieser Aggregation ist die Beziehung zwischen Assertion Test und Assertion Condition. Hierbei muss sichergestellt werden, dass jeder Assertion Test immer mindestens eine Assertion Condition besitzt.

Entity - Relevance Case

Ein Relevance Case ist ein Testfall, bei welchem eine Suchanfrage und die dazu erwarteten Ergebnisse hinterlegt werden. Zur Auswertung eines Relevance Cases wird eine Metrik benötigt, welche die erwarteten mit den realen Ergebnissen vergleicht. Je nach Metrik kann es notwendig sein, dass auch die Relevanz eines erwarteten Suchergebnisses definiert werden muss. Ohne eine Angabe haben alle erwarteten Ergebnisse dieselbe Relevanz. Zum Kennzeichnen der Relevance Cases besitzen diese Labels, welche aus einem Schlüssel und optional einem Wert bestehen. Ein Relevance Case kann beliebig viele Labels besitzen, allerdings immer nur ein Label pro Schlüssel. Somit kann die Menge der Relevance Cases einfach gefiltert werden.

Value Object - Expected Result

Ein Expected Result ist ein Value Object, welches ein erwartetes Ergebnis für einen Relevance Case referenziert. Dabei enthält jedes Objekt eine ID zum Dokument aus der Suchfunktion, den Titel des Dokuments und optional eine URL zu einem Bild, welches dem Dokument zugeordnet ist (z. B. Produktbild im E-Commerce). Die Klasse ist mit ihren Attributen und Methoden in Abbildung 4.7 dargestellt.

Aggregation - Relevance Case

Eine Relevance Case Entity soll mehrere Value Objects kapseln. Daher existiert ein Relevance Case Aggregate, welches den Search Query, die erwarteten Ergebnisse und die Metrik kapseln. Das Aggregate ist in Abbildung 4.7 dargestellt. Um das Aggregate zu erstellen, wird eine Factory Methode bereitgestellt (siehe Factory - Relevance Case Factory in Abschnitt 4.4.1), welche das entsprechende Metric Objekt erstellt und die erwarteten Ergebnisse hinzufügt. Zur Verwaltung der Labels existieren die Methoden „setLabel“, „removeLabel“ und „listLabels“. Die erwarteten Ergebnisse können mit den Methoden „addResult“, „removeResult“, „listResults“ und „setRelevance“ verwaltet werden.

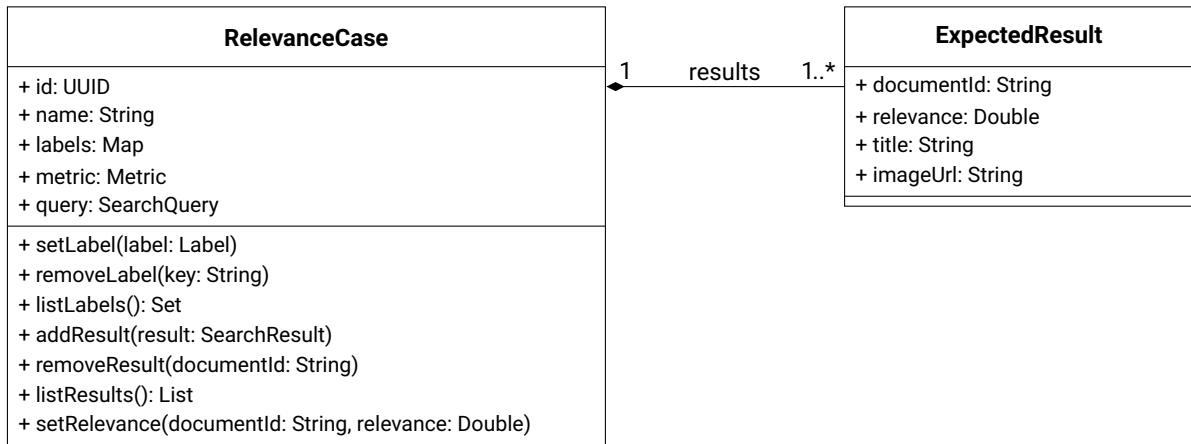


Abbildung 4.7: Aggregation - Relevance Case in der Offline Domain

Factory - Relevance Case Factory

Die Relevance Case Factory wird benötigt, wenn ein neuer Relevance Case erstellt werden soll. Da ein Relevance Case auch schon bei seiner Erstellung mehreren Bedingungen gerecht werden muss, wird hierfür eine eigene Factory benötigt. Die Factory stellt sicher, dass der Search Query valide ist, die Metric valide ist und der Relevance Case mindestens ein Expected Result enthält. Außerdem werden automatisch Labels erstellt (z. B. Erstellungsdatum, Typ der Metrik), welche später zur Filterung genutzt werden können.

Service - Relevance Case Updater

Dieser Service ermöglicht es, bestehende Relevance Cases mit dem Nutzerverhalten abzugleichen und gegebenenfalls anzupassen. Wenn durch Nutzerinteraktion eine Suchbegriff häufig genutzt wird und dazu kein Relevance Case existiert, erstellt dieser Service automatisch einen neuen Relevance Case. Die erwarteten Ergebnisse für diesen Relevance Case setzen sich aus den Ergebnissen zusammen, mit welchen die Nutzer am häufigsten interagiert haben. Optional kann dieser Service auch vorhandene Relevance Cases überprüfen und wenn nötig die erwarteten Ergebnisse anpassen, sodass sie zu den Ergebnissen passen mit welchen die Nutzer am häufigsten interagierten. Diese Funktion muss aber explizit aktiviert werden und kann nur für ausgewählte Relevance Cases ausgeführt, damit keine Änderungen der Nutzer des RQMS überschrieben werden.

Value Object - Label

Ein Label ist ein einfaches Value Object, welches einen Schlüssel (auch als „key“ bezeichnet) und einen Wert (auch als „value“ bezeichnet) besitzt. Objekte, die Labels referenzieren, dürfen pro Wert des Schlüssels immer nur genau ein Label zugewiesen haben. In der Implementierung müssen Labels nicht zwingend durch eine eigene Klasse beschrieben werden, es reicht auch, wenn sie als Attribut vom Typ „Map“ in einer Klasse verwendet werden, solange sichergestellt ist, dass jedes Objekt nur ein Label pro Schlüssel besitzt.

Aggregation - Label Filter

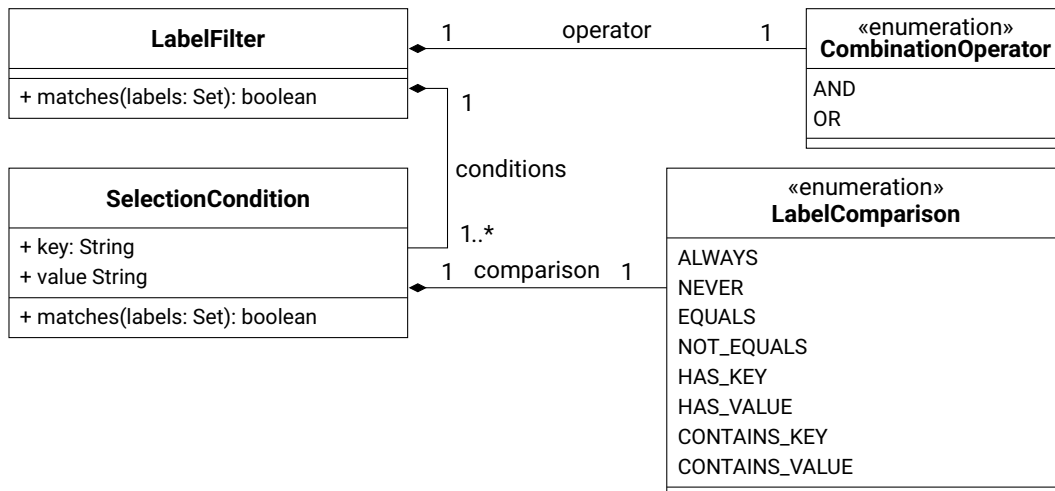


Abbildung 4.8: Aggregation - Label Filter in der Offline Domain

Der Label Filter ermöglicht die Definition von Bedingungen, um eine Menge von Objekten anhand ihrer Labels zu filtern. Dabei kann der Label Filter, durch die Wahl des entsprechenden „Combination Operators“, entweder sicherstellen, dass die gegebene Menge an Labels allen Bedingungen genügt („AND“ Operator) oder mindestens einer Bedingung genügt („OR“ Operator). Der Label Filter benötigt mindestens eine Bedingung, welche immer einen Wert für „Label Comparison“ haben muss. Je nach Wert des „Label Comparison“ muss ein Wert für „key“, „value“ oder beide gegeben sein. Die „Label Comparison“-Aufzählung bestimmt das Verhalten der „Selection Condition“. Dabei existieren zwei Spezialwerte. Der „ALWAYS“ Wert sorgt dafür, dass die Selection Condition immer „Wahr“ für alle Eingabewerte zurückgibt und der „NEVER“ Wert sorgt dafür, dass die Selection Condition immer „Falsch“ für alle Eingabewerte zurückgibt. Wenn der Wert des „comparison“-Attributes auf „EQUALS“ gesetzt ist, müssen bei einem der gegebenen Labels sowohl der Schlüssel als auch der Wert exakt übereinstimmen. Für „NOT_EQUALS“ darf bei dem Label mit dem übereinstimmenden Schlüssel der Wert nicht übereinstimmen. Für „HAS_KEY“ muss ein Label existieren, dessen Schlüssel mit dem Wert aus „key“ übereinstimmt. Für „HAS_VALUE“ muss ein Label existieren, dessen Wert mit dem von „value“ übereinstimmt. Für „CONTAINS_KEY“ muss ein Label existieren, welches in seinem Schlüssel den Wert von „key“ enthält. Für „CONTAINS_VALUE“ muss ein Label existieren, welches in seinem Wert den Wert von „value“ enthält. Die Aggregation ist mit ihren Attributen, Methoden und Referenzen in Abbildung 4.8 dargestellt.

Entity - Relevance Test Run

Ein Relevance Test Run spiegelt das Ergebnis einer Ausführung eines Relevance Cases auf einer definierten Konfiguration einer Suchfunktion wider. Die Konfiguration der Suchfunktion wird dabei nur optional referenziert, da sie nur für den Nutzer relevant ist, wenn sie noch existiert. Der Relevance Test Run soll aber länger aufbewahrt werden können als die Konfiguration. Außerdem enthält der Relevance Test Run eine Beschreibung, in welcher der

Grund für den Testlauf ausgeführt werden kann (z. B. automatischer Testlauf auf der Default Konfiguration der Suchfunktion). In einem Relevance Test Run werden die erwarteten und tatsächlichen Ergebnisse und der errechnete Score für jeden Relevance Case gespeichert. Die Relevance Cases werden dabei nicht referenziert, sondern deren Inhalt dupliziert, sodass im Nachhinein, auch wenn die Relevance Cases geändert werden, der Zustand zum Zeitpunkt der Ausführung des Tests nachvollzogen werden kann.

Value Object - Test Run Case

Ein Test Run Case dupliziert die für einen Relevance Test Run genutzten Relevance Cases, damit trotz einer Veränderung an den Relevance Cases der Relevance Test Run nachvollzogen werden kann. Ein Test Run Case referenziert außerdem nicht nur die erwarteten Ergebnisse, sondern auch die von der Suchfunktion erhaltenen Ergebnisse. In Abbildung 4.9 ist die Klasse mit ihren Attributen dargestellt.

Value Object - Test Run Search Result

Ein Test Run Search Result Objekt enthält entweder ein erwartetes oder ein erhaltenes Suchergebnis für einen Test Run Case. Dabei wird die „documentId“ des Dokuments aus der Suchfunktion, dessen Titel und wenn vorhanden auch eine URL zu einem Bild dieses Dokuments gespeichert. Wenn es sich um ein erwartetes Ergebnis handelt, kann auch die Relevanz des Ergebnisses gespeichert werden, diese ist aber optional. In Abbildung 4.9 ist die Klasse mit ihren Attributen dargestellt.

Aggregation - Relevance Test Run

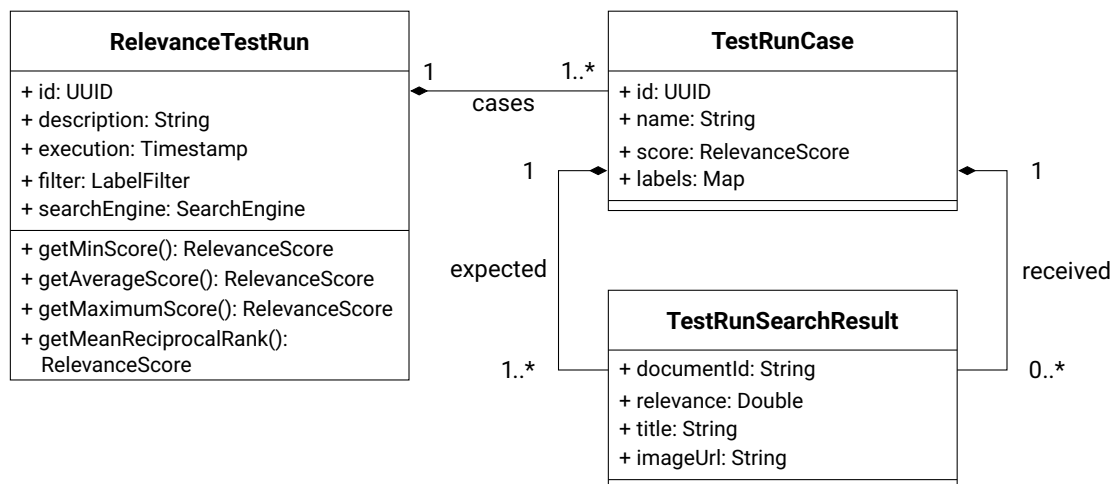


Abbildung 4.9: Aggregation - Relevance Test Run in der Offline Domain

Das Relevance Test Run Aggregate kapselt die Entity, alle Test Run Cases und deren Ergebnisse. Die Entity referenziert dabei verschiedene Value Objects, welche aber genauso wie die Entity selbst unveränderlich sind. Die referenzierten „Test Run Case“-Objekte duplizieren die genutzten Relevance Cases. Dabei werden neben den erwarteten Ergebnissen auch noch die bei der Suche erhaltenen Ergebnisse in „Test Run Search Result“-Objekten gespeichert.

Das Aggregate erlaubt außerdem den Zugriff auf Zusammenfassungen der Relevance Scores. Es werden der minimale, der maximale und der durchschnittliche Relevance Score für die genutzten Test Run Cases berechnet. Außerdem wird auch der Wert für den Mean Reciprocal Rank über die Methode „getMeanReciprocalRank“ bereitgestellt, diese sollte in der Implementierung wiederum die Reciprocal Rank Metric nutzen (siehe Value Object - Reciprocal rank). In Abbildung 4.9 ist das Klassendiagramm der Aggregation dargestellt.

Value Object - Test Run Data

Die „TestRunData“-Klasse stellt alle Assertion Tests und Relevance Cases bereit, welche für einen Testlauf benötigt widerspiegeln. Dabei werden die Relevance Cases nur über die Methode „filterRelevanceCases“ zugänglich gemacht, damit Optimierungen bei der Filterung möglich sind. Die Klasse ist in Abbildung 4.10 dargestellt.

Service - Relevance Test Run Service

Um einen Relevance Test Run durchzuführen, werden mindestens ein, aber meist mehrere Relevance Cases benötigt, mit welchen die Ergebnisse berechnet werden können. Diese werden mit einem definierten Label Filter ausgewählt. Für jeden Test Run muss dann eine definierte Suchfunktion aufgerufen werden, welche für den Search Query eine Ergebnisliste erzeugt. Hierbei können auch Suchfunktionen genutzt werden, welche nur mit Relevance Tests getestet werden sollen und nicht für Endnutzer zugänglich sind. Aus diesen Cases, den Ergebnislisten und den berechneten „RelevanceScore“-Objekten muss dann ein „RelevanceTestRun“-Objekt erzeugt werden. Vor der Durchführung der Relevance Test Runs werden alle Assertion Tests auf der gegebenen Suchfunktion ausgeführt, um sicherzustellen, dass diese korrekt konfiguriert ist. Falls nicht wird kein „RelevanceTestRun“-Objekt erzeugt, sondern ein Fehler zurückgegeben. Dieser Prozess wird im „RelevanceTestRun“-Service zusammengeführt. Dieser Service führt außerdem automatisch alle gespeicherten Test Run Schedules durch.

Entity - Test Run Schedule

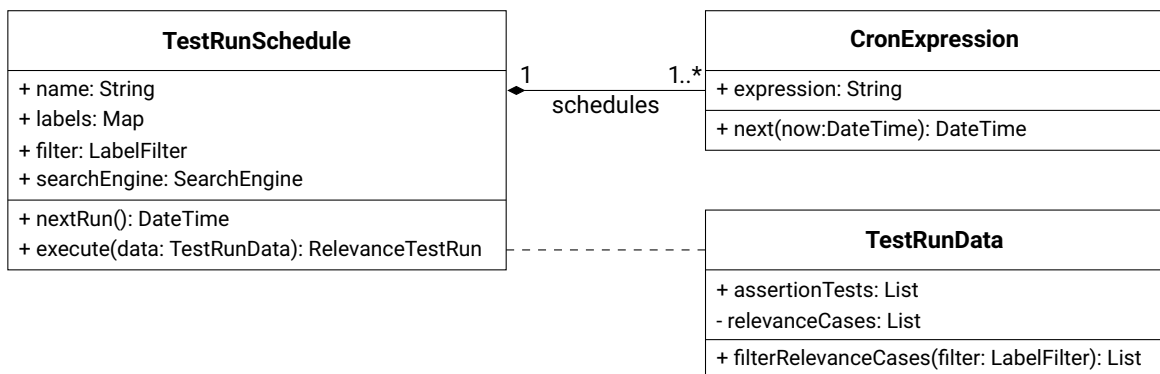


Abbildung 4.10: Entity - Test Run Schedule in der Offline Domain

Für eine Konfiguration der Suchfunktion sollen die Relevance Cases meistens regelmäßig

ausgeführt werden, um Relevanztrends frühzeitig zu erkennen. Gerade die Standardkonfiguration sollte dabei regelmäßig überprüft werden. Ein Test Run Schedule ermöglicht die Ausführung eines bestimmten Sets von Relevance Cases zu einem definierten Zeitpunkt auf einer Konfiguration der Suchfunktion. Die Auswahl der Relevance Cases erfolgt mithilfe von Labels. Die Menge der Relevance Cases kann durch die Definition eines Label Filters (z. B. ein Relevance Case muss ein Label mit dem Schlüssel „active“ enthalten) reduziert werden. Jeder Schedule besitzt selbst auch Labels, welche aber keine Auswirkung auf die Ausführung haben. Der Ausführungszeitpunkt eines Test Run Schedules wird anhand von Cron Expressions bestimmt. Dabei kann ein Schedule mehrere Cron Expressions besitzen. Jeder Test Run Schedule muss dabei eine Search Engine referenzieren, falls diese gelöscht wird, müssen auch die „TestRunSchedule“-Objekte, welche diese Search Engine referenzieren, automatisch gelöscht werden. Die „nextRun“-Methode gibt den Zeitpunkt der nächsten geplanten Ausführung zurück, an welchem die „execute“-Methode aufgerufen werden soll. Beim Aufruf der „execute“-Methode wird ein vollständiger Testlauf durchgeführt (siehe Service - Relevance Test Run Service). Für den Testlauf werden alle Assertion Tests und Relevance Tests benötigt, diese werden im „TestRunData“-Objekt zusammengefasst. In der Implementierung dürfen Spezialisierungen für das „TestRunData“-Objekt genutzt werden (z. B. zur Optimierung der Filterung von Relevance Cases). Eine naive Implementierung kann die Filterlogik des „LabelFilter“-Objekts nutzen (siehe Aggregation - Label Filter). In Abbildung 4.10 ist die Klassenstruktur im Kontext dargestellt.

Value Object - Relevance Score

Der Relevance Score ist ein Datenobjekt mit genau einem Attribut, dessen Wert eine rationale Zahl zwischen 0 und 1 sein muss. Er ist das Resultat eines Relevance Test Run und spiegelt die Qualität der Suchergebnisse wieder. Zur Berechnung des Relevance Scores kann eine Metrik genutzt werden.

Value Object - Metric

Das Metric Interface modelliert eine abstrakte Metrik, deren Implementierung dazu genutzt werden kann, um in einem Relevance Test Run den Relevance Score zu berechnen. Dazu erhält die Metrik die Liste der tatsächlichen Ergebnisse und berechnet so den Relevance Score. Ein „Metric“-Objekt enthält immer die Liste der gewünschten Suchergebnisse. In Abbildung 4.11 ist das Klassendiagramm für alle vordefinierten Metriken dargestellt. Plugins dürfen das Metric Interface implementieren und können so eigene Metriken bereitstellen. In den folgenden Abschnitten werden die fünf vordefinierten Metriken erläutert. Jede dieser Value Objects implementiert das Metric Interface.

Value Object - Precision at K

Der implementierte Algorithmus berechnet die Precision und vernachlässigt den Recall. Als zusätzlichen Parameter erhält diese Metrik „k“, eine Zahl, die angibt, wie viele Ergebnisse für die Berechnung des Relevance Scores betrachtet werden sollen [Sch20, S. 18].

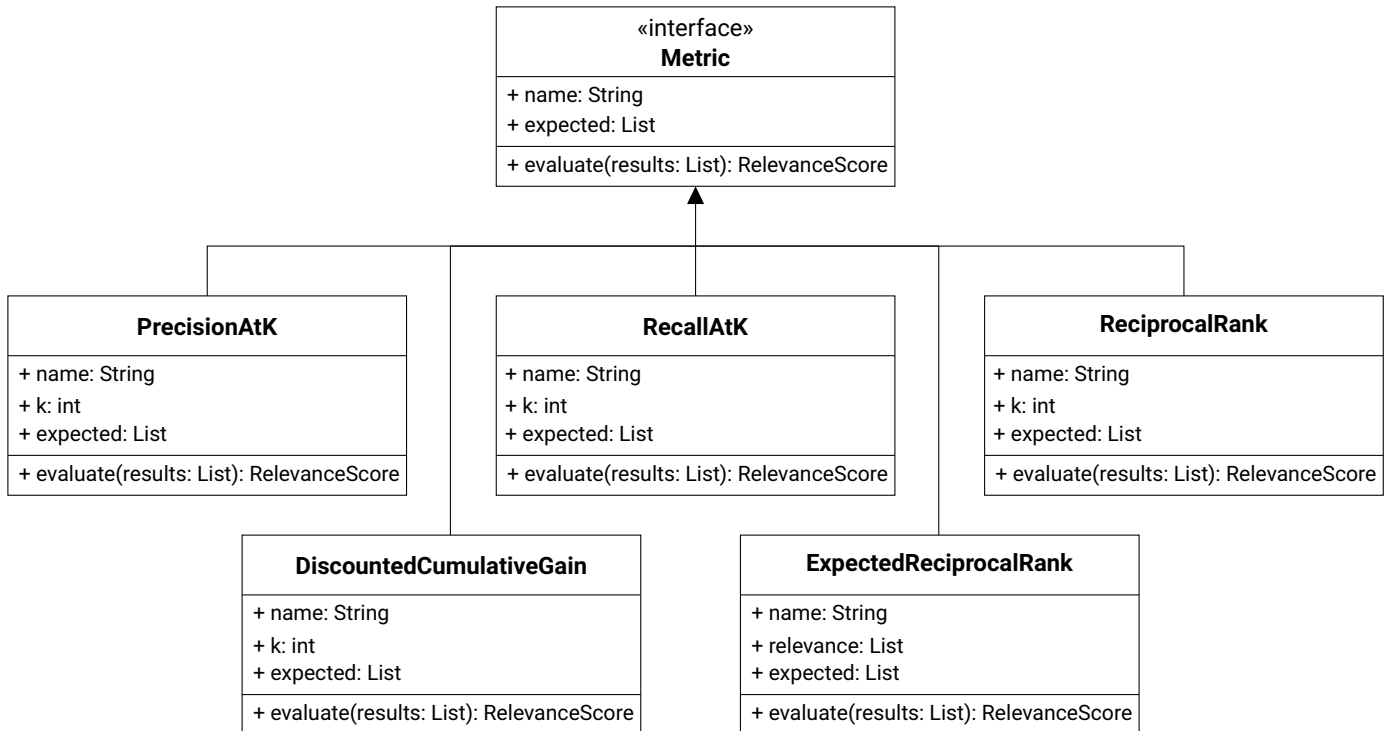


Abbildung 4.11: Value Object - Metric und dazugehörige Implementierungen in der Offline Domain

Value Object - Recall at K

Der implementierte Algorithmus berechnet den Recall und vernachlässigt die Precision. Als zusätzlichen Parameter erhält diese Metrik „k“, eine Zahl, die angibt, wie viele Ergebnisse für die Berechnung des Relevance Scores betrachtet werden sollen [Sch20, S. 18].

Value Object - Reciprocal rank

Der implementierte Algorithmus ist gut geeignet, wenn der Score für Relevance Cases berechnet werden soll, bei denen es nur ein relevantes Ergebnis gibt. Als zusätzlichen Parameter erhält diese Metrik „k“, eine Zahl, die angibt, wie viele Ergebnisse für die Berechnung des Relevance Scores betrachtet werden sollen [Sch20, S. 19]. Ein möglicher Einsatz ist der „fetch“ Use Case bei Netflix (siehe Unterabschnitt 2.1.1) oder die fokussierte Suche bei Spotify (siehe Unterabschnitt 2.1.2).

Value Object - Discounted cumulative gain

Der implementierte Algorithmus ist gut geeignet, wenn der Score für Relevance Cases berechnet werden soll, bei denen auch ein nicht relevantes Ergebnis an der ersten Position stehen darf, aber das relevante bzw. die relevanten Ergebnisse möglichst weit am Anfang stehen sollen. Als zusätzlichen Parameter erhält diese Metrik „k“, eine Zahl, die angibt, wie viele Ergebnisse für die Berechnung des Relevance Scores betrachtet werden sollen [Sch20, S. 19]. Ein häufiger Einsatz dieser Metrik ist im E-Commerce zu finden, wenn mehrere Ergebnisse

ähnlich relevant sind und nur sichergestellt werden muss, dass diese Ergebnisse möglichst weit oben in der Ergebnisliste zu finden sind. Gerade in sehr dynamischen Anwendungsfällen (z. B. Zalando, siehe Unterabschnitt 2.1.3) ist es mit dieser Metrik möglich, sinnvolle Werte für den Relevance Score zu bestimmen.

Value Object - Expected reciprocal rank

Der implementierte Algorithmus verbessert das Verhalten des Discounted Cumulative Gain indem auch die mögliche Nutzerinteraktion der weniger relevanten Dokumente, welche sich aber weiter am Anfang der Ergebnisliste befinden, betrachtet wird. Wenn ein Nutzer mit einem ausreichend relevanten Ergebnis am Anfang der Suchergebnisliste interagiert, verhindert diese Interaktion gegebenenfalls das Auffinden des relevantesten Ergebnisses, wenn dieses weiter unten in der Ergebnisliste folgt. Der Algorithmus benötigt für jedes Dokument, neben der erwarteten Position, auch die Wahrscheinlichkeit einer Interaktion des Nutzers mit dem jeweiligen Dokument. Der Algorithmus ist ein sehr gutes Model für das reale Nutzerverhalten in einer Suchfunktion, in welcher Ergebnisse mit unterschiedlichen Relevanzgraden existieren [Sch20, S. 20]. Beispiele hierfür sind die Bereiche E-Commerce (z. B. Zalando, siehe Unterabschnitt 2.1.3), Web-Suchen und Suchen in Dokumentationsportalen (z. B. Shopify, siehe Unterabschnitt 2.1.4).

Factory - Metrics

Diese Factory bietet die Möglichkeit, Metriken zu erstellen, dabei werden die Metriken anhand ihres Namens erzeugt. Plugins haben die Möglichkeit, eigene Funktionen zur Erstellung der Metriken in dieser Factory zu registrieren, so können diese dynamisch erstellt werden. Außerdem besitzt die Factory eine Methode zum Auflisten der Metriken, welche erzeugt werden können. Der Implementierung steht es frei, die Metriken nur zur Laufzeit vorzuhalten oder sie dauerhaft zu persistieren.

4.4.2 Domain Model - Online

Das Online Domain Model verwaltet die Daten, die durch Nutzer in der Suchfunktion erzeugt werden und ermöglicht eine Auswertung dieser Daten. Diese Daten werden nach der genutzten Suchfunktion und der Testvarianten der aktiven A/B-Tests gruppiert. Somit ist es möglich, dass A/B-Test detailliert ausgewertet werden können und Trends in der Nutzung der Suchfunktion frühzeitig erkannt werden. Die in diesem Domain Model verwalteten Daten bilden die Grundlage für eine automatisierte Erzeugung von Relevance Cases in der Offline Domain. Die zentrale Struktur, welche in dieser Domain verwaltet wird, ist der Search Fact, welcher alle Informationen zu einem Suchvorgang eines Nutzers bündelt. Der Search Fact wird in Abschnitt Entity - Search Fact genauer erläutert.

Entity - A/B-Test

Ein A/B-Test besteht aus mindestens zwei Varianten, einer Testvariante und einer Kontrollvariante. Wenn ein Event (z. B. Suchanfrage, Interaktion mit einem Suchergebnis) keiner

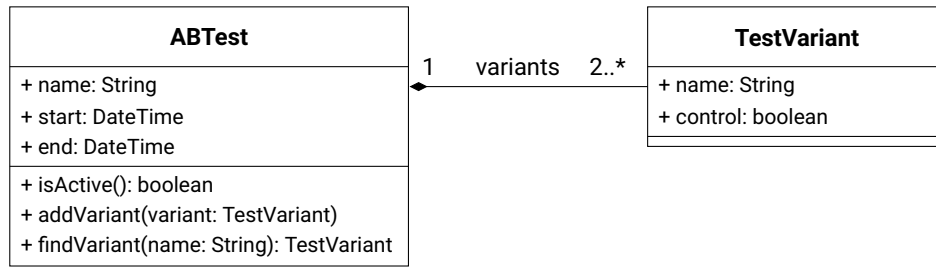


Abbildung 4.12: Entity - A/B-Test in der Online Domain

Testvariante zugeordnet ist, fällt es automatisch in die Kontrollvariante. Ein A/B-Test muss einen Startzeitpunkt und den Endzeitpunkt besitzen. Nur zwischen diesen Zeitpunkten gilt ein A/B-Test als aktiv und es werden Metriken erfasst. Da es mehrere A/B-Tests geben kann, muss ein A/B-Test immer alle Varianten definieren, für welche er getestet werden soll. Wenn ein Event dann keine Variante besitzt die dem jeweiligen A/B-Test zugewiesen ist, muss es automatisch der Kontrollvariante des aktuellen A/B-Tests zugeordnet werden. Von allen einem A/B-Test zugeordneten „TestVariant“-Objekten darf nur eines den Wert „true“ für das „control“-Attribut besitzen. Wenn eine Variante in einem A/B-Test gesucht wird und diese nicht gefunden wird, wird immer die Kontrollvariante zurückgegeben. Die Klasse, ihre Attribute, Methoden und Referenzen werden in Abbildung 4.12 dargestellt.

Entity - Test Variant

Ein „TestVariant“ Objekt enthält einen eindeutigen Bezeichner und referenziert die verwendete Search Engine. Ein A/B-Test besitzt mindestens zwei Testvarianten, davon immer genau eine Kontrollvariante, welche auch als solche über ein Attribut gekennzeichnet ist. Eine „TestVariant“ Entity darf immer nur einer „ABTest“-Entity zugeordnet werden und ihr Name muss global eindeutig sein. In Abbildung 4.12 wird diese Klasse in ihrem Nutzungskontext dargestellt.

Repository - A/B Test Repository

Das A/B-Test-Repository ist der Zugangspunkt zu allen „ABTest“-Entities. Es ermöglicht die Auflistung aller A/B-Tests, das Finden eines A/B-Tests anhand des Namens und das Finden eines A/B-Tests anhand des Namens einer seiner Varianten. Außerdem ist das Abspeichern von A/B-Tests möglich, sodass diese auch über einen Neustart der Anwendung hinaus verfügbar sind. Zum Löschen von A/B-Tests stellt das Repository auch eine Funktion bereit. Eine Implementierung muss sicherstellen, dass alle A/B-Tests persistiert sind und immer abgerufen werden können (z. B. relationale Datenbank).

Entity - Search Fact

Ein Search Fact bündelt alle Informationen für eine Suche eines Nutzers. Diese bestehen aus mindestens einem Search Query und einer Liste von Interaktionen mit den Ergebnissen der Suchanfrage. Aus diesen Daten werden dann unter anderem die Anzahl der Interaktionen mit

Suchergebnissen und der durchschnittliche Platz eines Suchergebnisses, mit welchem interagiert wurde berechnet. Damit diese Werte in einem A/B-Test als zusammengefasste Metriken ausgewertet werden, wird das „variants“-Attribut benötigt, welches die Testvarianten und somit auch die A/B-Tests selbst referenziert. Falls dem Search Fact kein A/B-Test zugeordnet werden kann, ist die Liste der Varianten leer. Ein Search Fact referenziert mindestens einen Search Query und gegebenenfalls eine oder mehrere Interaktionen mit den Ergebnissen. Weitere Search Queries für einen Search Fact treten unter anderem dann auf, wenn ein Nutzer mehr als eine Seite der Suchergebnisliste sehen möchte und daher auf eine andere Seite wechselt. Weitere Interaktionen für einen Search Fact treten dann auf, wenn ein Nutzer aus der Interaktion mit einem Ergebnis zurück in die Ergebnisliste kommt oder in einem E-Commerce Szenario, wenn der Nutzer seinem Warenkorb mehrere Produkte hinzufügt. Auch wenn ein Nutzer ein Produkt wieder aus seinem Warenkorb entfernt, muss diese Interaktion gespeichert werden, damit ermittelt werden kann, ob der Kunde dieses Produkt durch ein anderes ersetzt hat. Ein Search Fact referenziert außerdem die genutzte Search Engine. Falls diese gelöscht wird, muss dieser Wert auf „null“ gesetzt werden, da dann keine Search Engine mehr referenziert werden kann. Die Dauer, welche ein Search Fact angeben kann, bezieht sich auf den Zeitraum zwischen der ersten Suche und der letzten Interaktion bzw. Suchanfrage. Falls keine Interaktion oder weitere Suchanfrage vorhanden ist, kann keine Dauer berechnet werden. Die Klasse, ihre Attribute, Methoden und Referenzen werden in Abbildung 4.13 dargestellt.

Value Object - Search Query

Der Search Query speichert alle Informationen der Suchanfrage eines Nutzers. Dazu gehören der eingegebene Suchbegriff, der Zeitpunkt der Suchanfrage, die zusätzlichen Parameter der Suchanfrage, welche gegebenenfalls nicht explizit vom Nutzer angegeben werden, das Offset zum Beginn der Ergebnisliste, falls der Nutzer nicht nur die erste Seite besucht und optional die Anzahl der Suchergebnisse pro Seite. Ein Search Fact kann mehrere Suchanfragen enthalten, diese müssen aber derselben Suche zugeordnet werden können. Eine Korrektur des Suchbegriffs ist erlaubt, wenn es sich weiterhin um dieselbe Suche handelt. Dieser Fall tritt unter anderem dann auf, wenn die Autokorrektur der Suche einen verbesserten Suchbegriff vorschlägt. In Abbildung 4.13 wird diese Klasse in ihrem Nutzungskontext dargestellt.

Value Object - Search Result Interaction

Eine Interaktion mit einem Suchergebnis wird in einem „SearchResultInteraction“-Objekt erfasst. Dabei werden der Zeitpunkt der Interaktion, die ID des Dokuments aus der Suchfunktion und die Position des Ergebnisses in der Ergebnisliste (Rank) erfasst. Jede Interaktion wird außerdem einem Search Fact zugeordnet. Damit für eine Menge an Interaktionen die Reihenfolge bestimmt werden kann, wird der Zeitpunkt der Interaktion genutzt. Dieser muss daher präzise genug erfasst werden, damit eine zuverlässige Sortierung der Interaktionen möglich ist. Die eigentliche Implementierung für ein „SearchResultInteraction“-Objekt muss von der Datenquelle bereitgestellt werden, da es auch noch Eigenschaften enthalten kann, welche nur für den jeweiligen Nutzungskontext benötigt werden (z. B. im E-Commerce ist der Typ einer Interaktion das Entfernen aus dem Einkaufswagen). Diese Eigenschaften werden über

die Methoden „listPropertyNames“ und „getProperty“ bereitgestellt. In Abbildung 4.13 wird das Interface in seinem Nutzungskontext dargestellt.

Aggregation - Search Fact

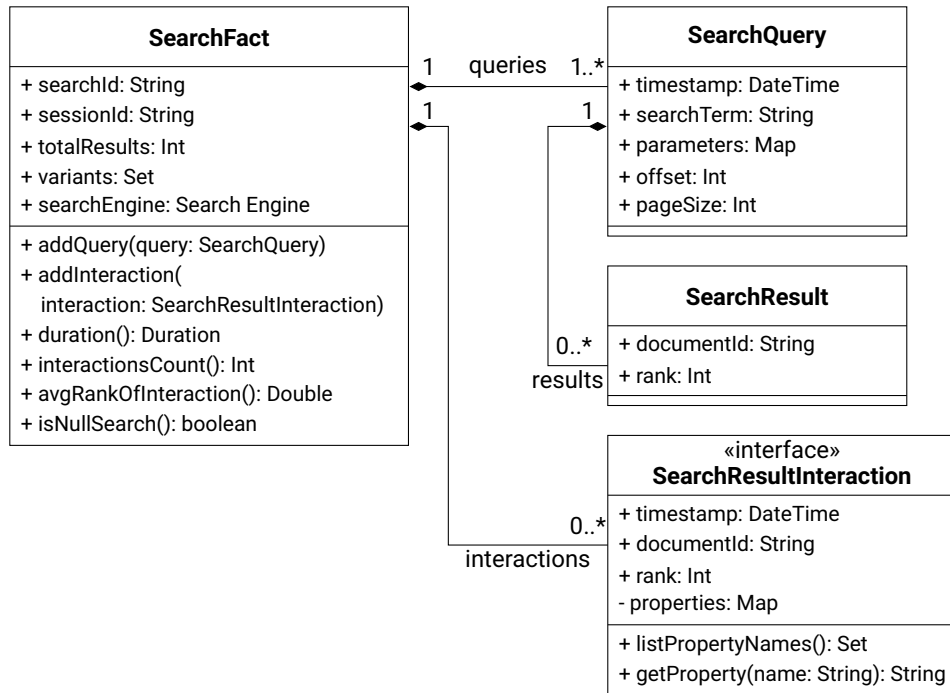


Abbildung 4.13: Aggregation - Search Fact in der Online Domain

Die Search Fact Aggregation kapselt den Zugriff auf die Elemente eines Search Facts. Somit ist sichergestellt, dass hinzugefügte Search Queries und Interaktionen nicht mehr verändert werden können. Außerdem können Werte über die Menge an Search Queries und Interaktionen berechnet werden. Durch die Kapselung ist es möglich, die Werte für die Dauer der Suche („duration“-Methode), die Anzahl der Interaktionen („interactionsCount“-Methode), die durchschnittliche Position für einen Interaktion mit einem Suchergebnis („avgRankOfInteraction“-Methode) und das Flag, ob es sich um eine Nullsuche handelt („isNullSearch“-Methode) zu bestimmen. Die Klassenstruktur mit Attributen, Methoden und Referenzen ist in Abbildung 4.13 dargestellt.

Service - Search Fact Consolidator

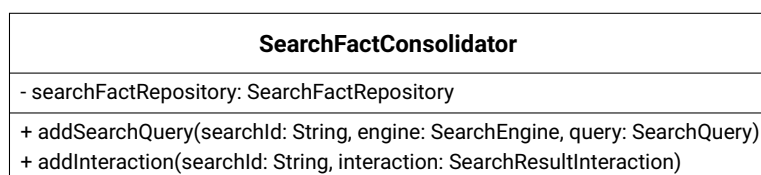


Abbildung 4.14: Service - Search Fact Consolidator in der Online Domain

Der Search Fact Consolidator nimmt einzelne Informationen zu den Suchen eines Nutzers

entgegen und kombiniert diese in einem Search Fact. So lässt sich in Echtzeit das Nutzerverhalten erfassen und auswerten. Da nur indirekt bekannt ist, ob eine Suche abgeschlossen ist oder nicht, ermöglicht der Search Fact Consolidator, alle bestehenden Search Facts zu erweitern. Die Klasse, ihre Attribute und Methoden werden in Abbildung 4.14 dargestellt.

Repository - Search Fact Repository

Das Search Fact Repository ist der Zugangspunkt zu allen Search Facts. Es ermöglicht die Auflistung aller Search Facts, die Auflistung anhand eines A/B-Tests bzw. einer Testvariante eines A/B-Tests und das Finden eines Search Facts anhand seines „searchId“-Attributes. Außerdem ist das Abspeichern von Search Facts möglich, sodass diese auch über einen Neustart der Anwendung hinaus verfügbar sind. Das Repository stellt auch eine Funktion zum Löschen von Search Facts bereit. Eine Implementierung muss sicherstellen, dass alle Search Facts persistiert sind und immer abgerufen werden können (z. B. relationale Datenbank).

Value Object - Time Interval

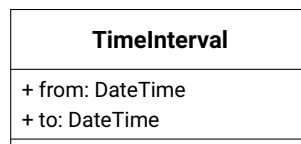


Abbildung 4.15: Value Object - Time Interval in der Online Domain

Ein Time Interval enthält einen Start- und den Endzeitpunkt eines zu repräsentierenden Zeitraums. Dabei muss der Startzeitpunkt immer vor dem Endzeitpunkt liegen, die beiden Zeitpunkte dürfen also auch nicht gleich sein. Die Klasse mit ihren Attributen wird in Abbildung 4.15 dargestellt.

Value Object - Metric Bucket

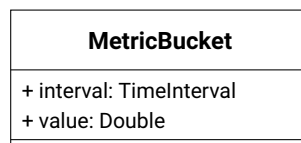


Abbildung 4.16: Value Object - Metric Bucket in der Online Domain

Ein Metric Bucket repräsentiert einen Wert für ein Zeitintervall. Der Wert ist eine rationale Zahl und wird im Attribut „value“ gespeichert, das Zeitintervall wird durch ein „TimeInterval“-Objekt repräsentiert, welches durch das „interval“-Attribut referenziert wird. Die Klasse mit ihren Attributen wird in Abbildung 4.16 dargestellt.

Value Object - Metric

Das Metric Interface modelliert eine abstrakte Metrik, deren Implementierung dazu genutzt werden kann, um für eine Menge an Search Facts einen Wert zu berechnen. Dazu erhält eine

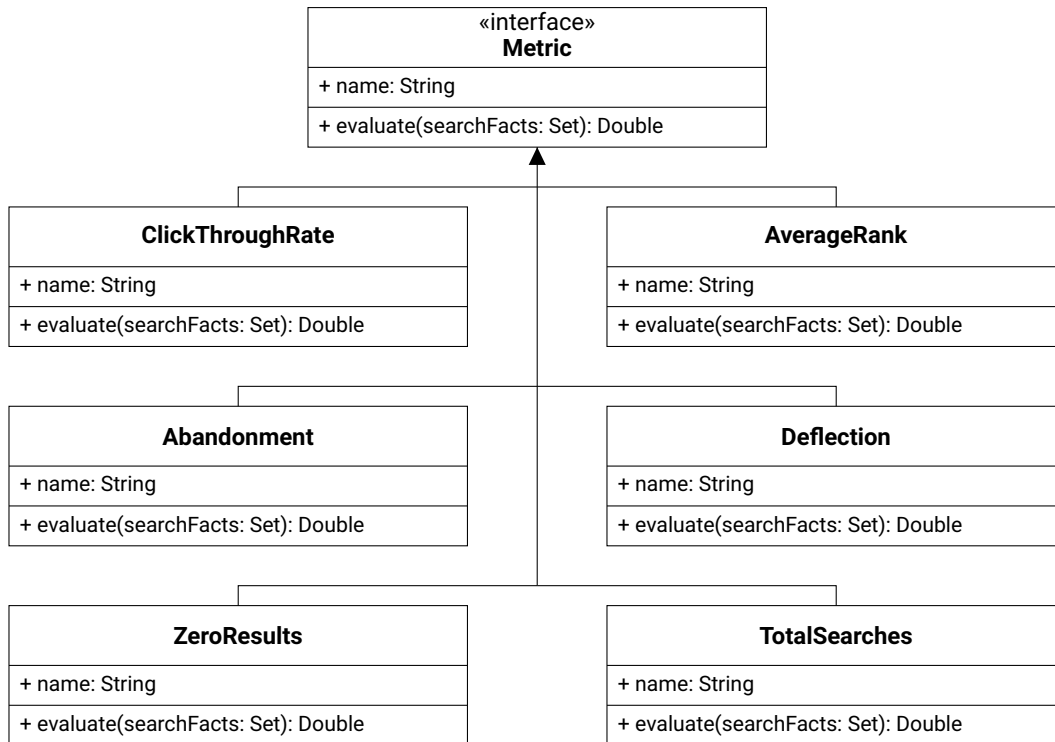


Abbildung 4.17: Value Object - Metric mit Implementierungen in der Online Domain

„Metric“-Implementierung eine Liste von Search Facts und berechnet eine rationale Zahl. Jede Metrik ist über ihren Namen eindeutig identifizierbar. In Abbildung 4.17 ist das Klassendiagramm für alle vordefinierten „Metric“-Implementierungen dargestellt. Plugins dürfen das „Metric Interface“ implementieren und können so eigene Metric Implementierungen bereitstellen.

Value Object - Click-through Rate

Die Click-through Rate-Metrik implementiert das Metric Interface und berechnet den Anteil der Suchanfragen mit Interaktion aus der Menge aller Suchanfragen. Diese Metrik ist sinnvoll, um die Qualität der Suchergebnisse im Allgemeinen zu beurteilen, also ob ein Nutzer das gesuchte Dokument in der Ergebnismenge findet oder nicht. Falls der Nutzer das gewünschte Dokument nicht findet, kann eine Ursache die fehlende Relevanz der Suchergebnisse sein oder das gewünschte Ergebnis existiert in der Ergebnismenge nicht. Dementsprechend ist diese Metrik sowohl für Domain Experts, Content Curators als auch Relevance Engineers ein wichtiger Indikator der Qualität der Suchfunktion. Üblicherweise wird diese Metrik für explorative Suchen immer niedriger ausfallen als für spezifische Suchen, da ein Nutzer, der explorativ sucht, nicht zwingend mit relevanten Ergebnissen interagiert, bei spezifischen Suchen ist eine Interaktion mit relevanten Ergebnissen wesentlich wahrscheinlicher [Sch20, S. 22].

Value Object - Average Rank

Die Average Rank-Metrik implementiert das Metric Interface und berechnet die durchschnittliche Position der Suchergebnisse, mit welchen interagiert wurde. Die Metrik gibt ein Indiz, ob die relevanten Ergebnisse am Anfang der Ergebnisliste zu finden sind, es werden jedoch nur Ergebnisse in die Berechnung einbezogen, welche eine Interaktion besitzen. Daher ist es sinnvoll, diese Metrik mit der Click-through Rate zu kombinieren (siehe Abschnitt 4.4.2). Wenn der Wert dieser Metrik gegen Eins strebt, steht das relevanteste Ergebnis am Anfang der Ergebnisliste. Der Wert dieser Metrik ist immer größer gleich Eins[Sch20, S. 22f].

Value Object - Abandonment

Die Abandonment-Metrik implementiert das Metric Interface und berechnet den Anteil der Suchanfragen ohne Interaktion mit den Ergebnissen bzw. ohne Folgesuchanfragen mit Interaktionen. Dabei ist der Wert der Metrik stets im Intervall Null bis einschließlich Eins. Diese Metrik gibt an, wie groß der Anteil der Suchen ist, welche keine Interaktion haben bzw. keine weitere inhaltsgleiche Suchanfrage mit Interaktion. Im Allgemeinen gilt, je kleiner der Wert dieser Metrik, desto hilfreicher ist die Suchfunktion für ihre Nutzer[Sch20, S. 23]. In der Praxis wird diese Metrik aber durch Spam, Bots und Crawler verfälscht, da diese nie mit den Ergebnissen interagieren[Slo21].

Value Object - Deflection

Die Deflection-Metrik implementiert das Metric Interface und berechnet den Anteil der Nutzer, welche in der Suchfunktion gehalten werden konnten. Wenn ein Nutzer die Suchfunktion ohne Interaktionen verlässt, ist davon auszugehen, dass dieser unzufrieden mit der Suchfunktion bzw. den Ergebnissen ist. Da diese Metrik stark vom Nutzungskontext des RQMS abhängig ist, kann eine generische Implementierung nur einen Annäherungswert berechnen. Eine kontextabhängige Implementierung dieser Art von Metrik sollte durch ein Plugin implementiert werden, welches Zugriff auf weitere Datenquellen besitzt[Sch20, S. 23].

Value Object - Zero Results

Die Zero Results-Metrik implementiert das Metric Interface und berechnet den Anteil der Suchanfragen ohne Ergebnisse. Der Wert dieser Metrik ist relevant für den Content Curator bzw. Domain Expert. Denn wenn dieser Wert zu groß wird, deutet die Metrik darauf hin, dass die vom Kunden gesuchten Ergebnisse nicht existieren (bzw. zumindest nicht gefunden werden können). Gerade bei dieser Metrik ist es wichtig, alle darin vorkommenden Search Facts individuell zu prüfen. Der Wert dieser Metrik ohne zeitlichen Bezug bringt nur einen geringen Mehrwert für die Nutzer des RQMS, aber wenn sich der Wert dieser Metrik im zeitlichen Verlauf verändert, kann er großen Aufschluss über die Wirkung von Relevanzverbesserungsmaßnahmen geben.

Value Object - Total Searches

Diese Metrik gibt immer die Anzahl der erhaltenen „SearchFact“-Objekte zurück. Sie wird vor allem dann benötigt, wenn andere Metriken in relativen Werten dargestellt werden sollen. Ein Beispiel ist die zusammengesetzte Metrik aus Anzahl der Nullsuchen (siehe Value Object - Zero Results) in Relation zu allen getätigten Suchen. Damit lässt sich dann errechnen, wie groß der Anteil der Nullsuchen aktuell ist.

Repository - Metrics Repository

Dieses Repository bietet die Möglichkeit, Metriken aufzulisten. Dabei kann eine Metrik anhand ihres Namens gefunden werden oder eine Liste aller Metriken angefragt werden. Plugins haben die Möglichkeit, eigene Metriken in diesem Repository zu speichern. Der Implementierung steht es frei, die Metriken nur zur Laufzeit vorzuhalten oder sie dauerhaft zu persistieren.

Service - Analytics Service

AnalyticsService
- metricsRepository: MetricsRepository - searchFactRepository: SearchFactRepository
+ evaluateTotal(abtest: ABTest, metricName: String): Map + evaluateTotal(searchEngine: SearchEngine, metricName: String): Double + evaluateTimeSeries(abtest: ABTest, metricName: String): Map + evaluateTimeSeries(searchEngine: SearchEngine, metricName: String): List + evaluateTimeInterval(abtest: ABTest, metricName: String, interval: TimeInterval): Map + evaluateTimeInterval(searchEngine: SearchEngine, metricName: String, interval: TimeInterval): Double + findZeroResultTerms(abtest: ABTest): List + findZeroResultTerms(searchEngine: SearchEngine): List + findZeroResultTerms(abtest: ABTest, interval: TimeInterval): List + findZeroResultTerms(searchEngine: SearchEngine, interval: TimeInterval): List

Abbildung 4.18: Service - Analytics Service in der Online Domain

Der Analytics Service bietet den Zugangspunkt zu allen Metriken. Dabei wird immer der Name der Metrik benötigt und entweder ein A/B-Test oder eine Search Engine. Der Rückgabewert unterscheidet sich zwischen den verschiedenen Anfragen. Für Anfragen über den allgemeinen Wert einer Metrik wird nur eine rationale Zahl zurückgegeben. Für Anfragen über den zeitlichen Verlauf einer Metrik wird eine Liste von „MetricBucket“-Objekten (siehe Abschnitt 4.4.2) zurückgegeben. Wenn die Anfragen für einen A/B-Test gestellt werden, werden die Rückgabewerte immer pro Testvariante berechnet. Die Rückgabewerte werden als Map zurückgegeben, wobei der Schlüssel der Name der Testvariante ist und der Wert in jedem Eintrag der eigentliche Rückgabewert der Metrik ist. Jeder Wert in der Map ist also entweder eine rationale Zahl oder eine Liste von „MetricBucket“-Objekten. Um Werte für einen bestimmten Zeitbereich zu berechnen, können die „evaluateTimeInterval“ Methoden

genutzt werden. Die Klasse, ihre Attribute und Methoden werden in Abbildung 4.18 dargestellt. Außerdem erlaubt der Analytics Service das Auflisten der Suchbegriffe, für welche kein Ergebnis gefunden wurde. Diese Suchbegriffe können entweder für eine Search Engine oder für einen A/B-Test mithilfe der „findZeroResultTerms“-Methode ausgegeben werden.

4.4.3 Domain Model - Integration

Das Integration Domain Model verwaltet alle externen Datenquellen, Suchfunktionen und Plugins. Plugins stellen Implementierungen für Datenquellen und Suchfunktionen bereit. Dabei können mehrere Plugins gleichzeitig geladen werden und auch dieselben Funktionalitäten implementieren, solange sich ihr Name und die Packages unterscheiden.

Entity - Plugin

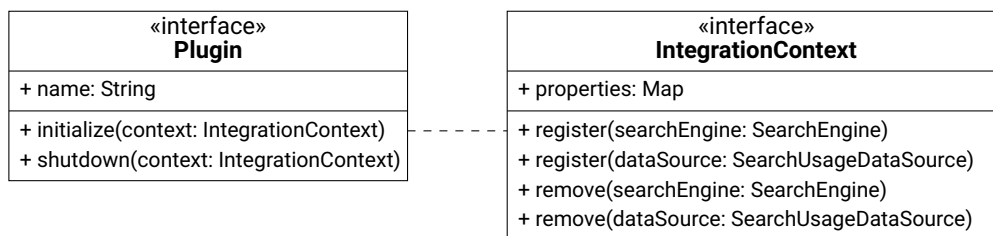


Abbildung 4.19: Entity - Plugin in der Integration Domain

Ein Plugin ist eine Einheit, welche externe Datenquellen und Suchfunktionen bereitstellen kann. Dabei übernimmt das Plugin die Verwaltung und besitzt auch eigene Einstellungsmöglichkeiten. Eine Plugin-Implementierung muss einen Konstruktor ohne Argumente besitzen, da diese Klasse generisch erstellt wird. Nach der Erstellung wird die „initialize“-Methode aufgerufen, welche ein „IntegrationContext“-Objekte als Parameter besitzt. Ein Plugin kann sich das „IntegrationContext“-Objekt vorhalten und sowohl „SearchEngine“-Objekte, als auch „SearchUsageDataSource“-Objekte registrieren und entfernen. Wenn die „shutdown“-Methode aufgerufen wird, muss ein Plugin alle Resource freigeben und gegebenenfalls existierende „SearchEngine“-Objekte und „SearchUsageDataSource“-Objekte aus dem Kontext entfernen. Die Klassenstruktur ist in ihrem Nutzungskontext in Abbildung 4.19 dargestellt.

Value Object - Integration Context

Das Integration Context Interface bietet einen Zugang zu den anwendungsspezifischen Funktionalitäten. Dazu gehören zum einen die Einstellungen, welche der Nutzer an der Anwendung vornehmen kann und welche auch von den Plugins verwendet werden sollten und zum anderen die Funktionalität „SearchEngine“- und „SearchUsageDataSource“-Objekte für die Verwendung zu registrieren. Über dieses Interface können die registrierten Objekte jederzeit auch wieder entfernt werden (z. B. wenn eine Konfiguration einer Suchfunktion gelöscht wird). Das Integration Context Interface muss von der Anwendungsschicht implementiert werden,

da hier viele technische Komponenten verwendet werden (z. B. Einstellungen aus den Umgebungsvariablen, Dateien, etc.). Wenn in der Implementierung bereits eine Klasse für den Anwendungskontext existiert, sollte diese das Integration Context Interface implementieren. Das Interface, seine Attribute und Methoden werden in Abbildung 4.19 dargestellt.

Entity - Search Usage Data Source

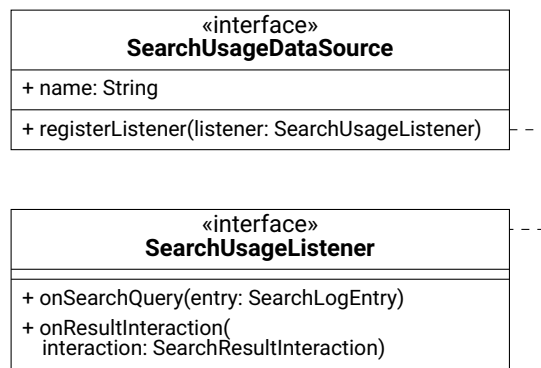


Abbildung 4.20: Entity - Search Usage Data Source in der Integration Domain

Eine Search Usage Data Source ist der Ursprung der Daten für die Auswertung des Nutzerverhaltens. Dabei kann ein Search Usage Listener registriert werden, welcher Informationen zu allen Suchen und Interaktionen erhält. Dieses Entwurfsmuster wird als „Observer Pattern“ bezeichnet[siehe GHJ97]. Dieselbe Implementierung einer Search Usage Data Source kann sowohl Informationen über Suchanfragen, als auch Informationen zu Interaktionen der Nutzer bereitstellen. Diese Daten können aber auch von mehreren unterschiedlichen Implementierungen bereitgestellt werden.

Value Object - Search Usage Listener

Das Search Usage Listener Interface muss von den Komponenten implementiert werden, welche die Informationen aus der Search Usage Data Source erhalten wollen. Die zwei Methoden des Interfaces werden mit den entsprechenden Daten von der Search Usage Data Source aufgerufen. Eine Instanz eines Search Usage Listener kann dabei an mehreren Search Usage Data Sources gleichzeitig registriert sein. Die „onSearchQuery“-Methode wird aufgerufen, wenn ein Nutzer eine Suchanfrage ausgeführt hat. Die „onResultInteraction“-Methode wird aufgerufen, wenn ein Nutzer mit einem Ergebnis interagiert hat. Das Interface und seine Methoden werden in Abbildung 4.20 dargestellt.

Entity - Search Log Entry

Ein „SearchLogEntry“-Objekt repräsentiert eine Suchanfrage eines Nutzers und enthält die dafür zurückgegeben Suchergebnisse. Dabei wird jedes Objekt anhand des „searchId“-Attributes einer Suche zugewiesen und ist über das „sessionId“-Attribut auch implizit einem Nutzer zuzuordnen. Die Session-ID kann hierbei als Pseudonym für einen Nutzer angesehen werden. Außerdem enthält jedes Objekt eine Referenz auf die genutzte Search Engine und alle für diese Suche aktiven Varianten von A/B-Tests. Diese werden in anderen Teilen der Anwendung

SearchLogEntry
+ searchId: String + sessionId: String + searchEngine: SearchEngine + testVariants: Set + searchTerm: String + parameters: Map + offset: Int + pageSize: Int

Abbildung 4.21: Entity - Search Log Entry in der Integration Domain

gegebenfalls für Auswertungszwecke benötigt. Die Kerninformationen zu einer Suchanfrage sind in den Attributen „searchTerm“, welcher den vom Nutzer angegebenen Suchbegriff speichert, „parameters“, welches alle weiteren Parameter der Suchanfrage enthält, „offset“ und „pageSize“, welche die Seite bzw. Position in der Ergebnisliste enthalten, vorhanden. Die Klasse und ihre Attribute sind in Abbildung 4.21 dargestellt.

Entity - Search Result Interaction

SearchResultInteraction
+ searchId: String + sessionId: String + searchEngine: SearchEngine + testVariants: Set + documentId: String + resultRank: Int + parameters: Map

Abbildung 4.22: Entity - Search Result Interaction in der Integration Domain

Eine Search Result Interaction repräsentiert eine Aktion mit genau einem Suchergebnis. Wenn ein Nutzer mit mehreren Suchergebnissen interagiert, gibt es für jede Interaktion eine Instanz der Entity. Damit die Interaktion einer Suche bzw. einem Nutzer zugewiesen werden kann, enthält die Klasse die „searchId“- und „sessionId“-Attribute. Die Session-ID kann hierbei als Pseudonym für einen Nutzer angesehen werden und ermöglicht eine implizite Zuordnung zwischen einer Interaktion und einem Nutzer. Außerdem enthält jedes Objekt eine Referenz auf die genutzte Search Engine und alle für diese Suche aktiven Varianten von A/B-Tests. Das Dokument, mit welchem der Nutzer interagiert hat, ist über die „documentId“ referenziert und über das „resultRank“ Attribut ist die Position des Dokuments in der Ergebnisliste erkennbar. Damit auch zusätzliche Informationen gespeichert werden können (z. B. die Art der Interaktion), enthält die Klasse das „parameters“-Attribut, in welchem Werte anhand eines Schlüssels hinterlegt werden können. Die Klasse und ihre Attribute sind in Abbildung 4.22 dargestellt.

5 Referenzimplementierung des Architekturvorschlags

5.1 Allgemeines

Die hier vorgestellte Referenzimplementierung implementiert die in Kapitel 4 vorgestellte Architektur. Ziel der Referenzimplementierung ist es, zu zeigen, dass eine Implementierung der Architektur generell möglich ist. Es werden daher nicht alle Funktionen vollständig implementiert. Die Referenzimplementierung soll außerdem dazu dienen, einen Eindruck zu vermitteln, wie ein RQMS funktionieren kann und welche Funktionen bereitgestellt werden können. Somit ist die Referenzimplementierung eine Grundlage, die zur iterativen Weiterentwicklung des RQMS und für weitere Forschung genutzt werden kann. In der hier vorgestellten Version sollte diese Implementierung nicht produktiv eingesetzt werden, ohne entsprechende Anpassungen vorzunehmen.

5.1.1 Programmiersprachen

Als primäre Programmiersprache für das RQMS wurde Java gewählt, eine objektorientierte Programmiersprache. Für eine auf Domain-Driven Design basierende Architektur eignet sich eine objektorientierte Sprache, da das Domain Model gut durch Klassen und Objekte repräsentiert werden kann. In der verwendeten Version 17 der Sprache sind auch viele nützliche Features enthalten, sodass sie sich zur alternativen Programmiersprache Kotlin nur unwesentlich unterscheidet. Außerdem erlaubt Java eine schnelle Feature-Entwicklung und erzeugt plattformunabhängige Artefakte[ora21]. Für die Unit-Tests wurde Groovy als Programmiersprache genutzt, weil diese Sprache mit Java kompatibel ist und da das Testing Framework Spock[BN] nur Groovy unterstützt. Im User Interface wurde TypeScript verwendet, weil das Framework Angular[KD16] nur diese Programmiersprache unterstützt. Eine direkte Verwendung von Angular in Java ist nicht möglich und die Sprachen sind auch nicht direkt kompatibel.

5.1.2 Framework

Spring Boot

Für die Entwicklung des RQMS wird Spring Boot genutzt. Spring Boot vereinfacht die Nutzung des Spring Frameworks, welches wiederum eine vereinfachte Nutzung von Java, Java EE und anderen Third Party-Bibliotheken anstrebt[vgl. NWW12a]. Dabei basiert Spring auf den Prinzipien der Dependency Injection, aspektorientierten Programmierung und einheitlicher Fehlerbehandlung[Joh03]. Dabei ist Spring Boot kompatibel zu den Anforderungen des

Domain-Driven Designs, da es auf Grundlage des Domain-Driven Designs entwickelt wurde [vgl. Dro20, Domain-Driven Design in a Spring application]. Außerdem ist es mit Spring Boot sehr einfach möglich, eine zustandslose Anwendung zu entwickeln, welche dann hochverfügbar betrieben werden kann.

Spock

Als Testframework für Unit Tests wird Spock genutzt, da dieses Framework es erlaubt, Anforderungen als Code zu formulieren. Somit ist es einfach, für Menschen verständliche Testfälle aus den Anforderungen zu erstellen. Spock ist dabei durch den sogenannten „JUnit runner“ kompatibel mit den meisten IDEs und „Build Tools“ [NB07].

Angular

Für das User Interface wird das Framework Angular genutzt, dieses Framework ist komponentenbasiert und bietet einen großen Funktionsumfang an. So werden standardmäßig Funktionen unter anderem für das Routing in der Anwendung, Client-Server-Kommunikation und Formular Management bereitgestellt. Durch eine vollständige Integration von Code und „Build Tools“ erlaubt es Angular, User Interfaces schnell zu entwickeln, welche auch mittel- bis langfristig performant sind [KD16].

5.1.3 Build Tools

Zur Erstellung der Anwendung, Verwaltung von Bibliotheken und Abhängigkeiten und der Ausführung von Tests werden verschiedene „Build Tools“ genutzt. Im Folgenden sind die verschiedenen Anwendungen aufgelistet und für jede Anwendung wird ihr Einsatzbereich beschrieben.

Gradle

Gradle ist ein flexibles „build automation tool“, welches verschiedenste Arten von Anwendungen bauen kann. Am häufigsten wird Gradle für JVM-basierte Anwendungen genutzt, da Gradle selbst auch eine JVM zur Ausführung benötigt [MWB07b]. Im RQMS wird Gradle eingesetzt, um das Backend zu bauen und zu testen. Dabei wird das Feature der Gradle-„Multi-project builds“ [vgl. MWB07a] genutzt, um die in Abbildung 4.2 dargestellte Modularisierung umzusetzen. Das umfassende Projekt heißt „root“ und alle Module sind direkt als Projekte darunter angelegt. Im „rqms-api-general“-Projekt wird das User Interface gebaut, dieses ist im Gegensatz zur Backend-Logik nicht in Gradle-Projekte ausgelagert, sondern wird durch ein eigenes „Build Tool“ erstellt. Gradle übernimmt dabei durch das Angular Gradle Plugin [Kun20, vgl.] die Ausführung dieses „Build Tools“. Somit wird das User Interface auch gebaut, wenn Gradle aufgerufen wird.

NPM

Die Abhängigkeiten des User Interfaces werden mit NPM[Sch10] verwaltet. Dieser Package Manager wird häufig für ECMAScript bzw. TypeScript-Anwendungen genutzt. NPM übernimmt für die UI aber keine weiteren Aufgaben eines „Build Tools“.

Angular Compiler

Der Angular Compiler muss genutzt werden, um Anwendungen mit dem Angular Framework[KD16] zu erstellen. Dabei wird der Angular Compiler in NPM integriert, was wiederum von Gradle aus aufgerufen wird. Der Angular Compiler erstellt aus dem TypeScript Code, den HTML- und den CSS-Dokumenten eine Webseite, welche im Browser genutzt werden kann.

5.1.4 Bibliotheken

Das RQMS benutzt viele verschiedene Bibliotheken, um die gewünschte Funktionalität bereitzustellen, im Folgenden sind die wichtigsten aufgeführt. Es wird auch kurz erläutert, warum und wo diese verwendet werden.

PF4J - Plugin Framework 4 Java

Das RQMS soll dynamisch durch Plugins erweitert werden (siehe Unterabschnitt 3.5.3). Dazu wird in dieser Implementierung das Framework PF4J[Høy+12] verwendet. Dieses Framework bietet eine direkte Integration mit Spring Boot an[vgl. Sui14]. Daher können die Plugins Objekte im Kontext des RQMS bereitstellen. Das PF4J kann mehrere Plugins gleichzeitig laden. Diese müssen in einem Ordner als „zip“ oder „jar“ vorliegen. Plugins können auch aktiviert oder deaktiviert werden, ohne dass der Programmcode aus dem Ordner entfernt werden muss. Die Plugins können auch mit Gradle gebaut werden und müssen eine Manifest-Datei enthalten, welche das Plugin beschreibt[vgl. Høy+12].

Groovy

Die in Unterabschnitt 4.4.1 beschriebenen Assertion Tests erlauben es den Nutzern, Bedingungen als Code zu definieren. In dieser Implementierung können diese Bedingungen als Groovy-Skript implementiert werden. Dazu muss diese Implementierung die Groovy Runtime als Abhängigkeit einbinden. Wenn die Groovy-Skripte ausgeführt werden, können Objekte in den Skriptkontext übergeben, dort verwendet und verändert werden[vgl. Kin20]. Wegen dieser einfachen Einbindung wurde Groovy als Sprache für die Assertion Tests gewählt und kann auch von Plugins für die Erweiterung der Funktionalität des RQMS verwendet werden[vgl. Str03].

5.1.5 Source Code

Diese Implementierung nutzt für die Verwaltung des Source Codes den Dienst „GitHub“, welcher von Microsoft bereitgestellt wird. Dort ist der gesamte Source Code des RQMS zugänglich und auch alle bisherigen Änderungen können dort abgefragt werden. Der Dienst „GitHub“ stellt den Source Code über eine Web-Oberfläche und über das Versionsverwaltungssystem Git bereit. Mit einem Git-kompatiblen Client kann jeder sich den Source Code herunterladen, anpassen und allen anderen zur Verfügung stellen. Somit kann diese Referenzimplementierung von anderen einfach als Grundlage für eine Weiterentwicklung genutzt werden. Der Source Code des RQMS ist unter <https://github.com/raynigon/rqms> öffentlich verfügbar.

5.1.6 Packaging

Damit das RQMS einfach zu nutzen ist, muss es als fertiges Paket zur Installation bereitgestellt werden. Dabei ist für eine JVM-basierte Anwendung das Paketformat „jar“ üblich. Dieses Paketformat wird von Gradle (bzw. dem Spring Boot Gradle Plugin) automatisch erstellt und kann einfach ausgeführt werden, wenn auf dem Rechner eine JVM installiert ist. Für die Installation des RQMS auf einem System ohne installierte JVM werden ein Pakete im Format „deb“ oder „rpm“ bereitgestellt. Diese erlauben es, die benötigten Abhängigkeiten zu definieren, sodass diese Abhängigkeiten automatisch installiert werden. Die wichtigste Abhängigkeit des RQMS ist die JVM, welche in einer passenden Version installiert sein muss. Das RQMS nutzt das OSPackage Gradle Plugin[Kru11], um sowohl „deb“- als auch „rpm“-Pakete zu erstellen. Docker Images sind ein sehr häufig genutztes Paketformat. Sie erlauben es, eine Anwendung mit allen Abhängigkeiten als Paket bereitzustellen, sodass dieses direkt ohne Installation ausgeführt werden kann. Das RQMS nutzt das JIB Gradle Plugin[Goo17], um ein Docker Image mit der passenden JVM zu erstellen. Wenn das Docker Image in einem Orchestrierungssystem, wie zum Beispiel Kubernetes, verwendet werden soll, ist es üblich, auch Helm Charts oder Kustomize-Dateien bereitzustellen, welche die Konfiguration der Anwendung im Orchestrierungssystem definieren.

5.1.7 Deployment

Damit das RQMS genutzt werden kann, wird eine PostgreSQL-Datenbank benötigt. Das RQMS selbst kann mit mehreren Instanzen gestartet werden, diese sollten sich hinter einem HTTP Load Balancer befinden. Jede Instanz benötigt mindestens zwei Gigabyte Arbeitsspeicher und einen vollen CPU-Kern mit mindestens 2 GHz Taktfrequenz. Um das RQMS optimal nutzen zu können, sollten vier Gigabyte Arbeitsspeicher und vier CPU-Kerne bereitgestellt werden. Der Speicherplatzverbrauch der Datenbank hängt von der Menge der Nutzungsdaten und deren Speicherdauer ab. Mindestens sollten die Datenbank zehn Gigabyte Speicherplatz besitzen. Für ein Produktivsystem sollte die Datenbank ca. 100 Gigabyte Speicherplatz nutzen können.

Wenn das RQMS mit mehreren Instanzen deployed wird, muss darauf geachtet werden, dass alle Instanzen dieselben Plugins besitzen. Ansonsten ist der fehlerfreie Betrieb des RQMS

nicht möglich. Ein Deployment als Docker Container oder in einem Orchestrierungssystem ist mit dem RQMS möglich, es kann dabei auch als zustandslose Anwendung deployed werden. Wichtig ist nur, dass jede Instanz Zugriff auf einen beschreibbaren Ordner für temporäre Dateien besitzt. Dieser wird von der JVM, dem Spring Boot Framework und weiteren Abhängigkeiten für verschiedene Aufgaben benötigt. Der Ordner darf für jeden Neustart der Instanz leer sein, muss aber während der Laufzeit der Anwendung persistent sein.

5.2 Umsetzung der Architekturschichten

Die Module, welche in der Architektur vorgeschlagen werden (siehe Abschnitt 4.3), wurden genauso als Gradle-Projekte umgesetzt. Für jedes Modul existiert ein Gradle-Projekt, welches entsprechend von den unterliegenden Projekten abhängt. Diese sind in Abbildung 5.1 dargestellt. Jedes Gradle-Projekt hatte dieselbe Gruppen-ID, nur der Projektname ist unterschiedlich.

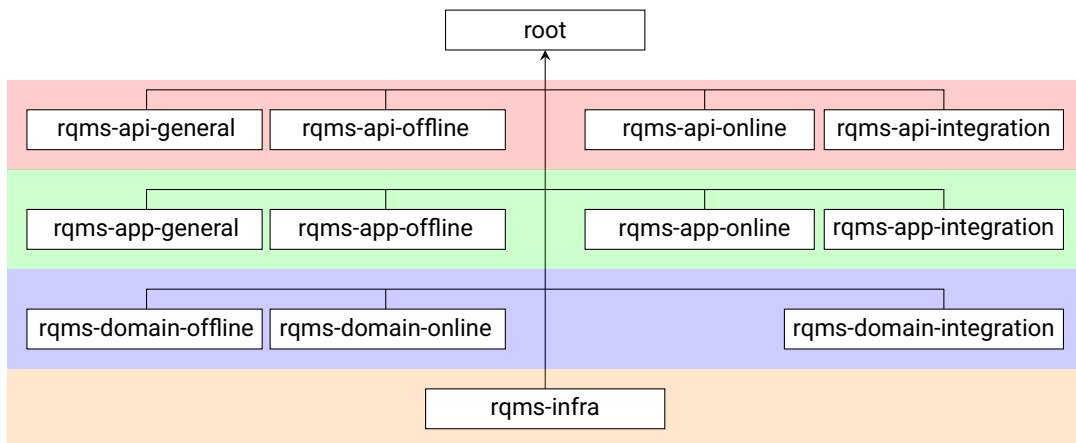


Abbildung 5.1: Gradle Module des RQMS in den entsprechenden Architekturschichten

In einigen Bereichen musste die Architektur aus technischen Gründen abgeändert werden, im Folgenden wird für jede Schicht erläutert, wie diese implementiert wurde und wo von der Architektur abgewichen wurde.

5.2.1 Präsentationsschicht

Die Implementierung der Präsentationsschicht ist zweigeteilt, zum einen das User Interface, welches mit Angular entwickelt wurde und zum anderen die API, welche in Java mit Spring Boot entwickelt wurde. Durch den Einsatz von Spring Boot kann die API einfach implementiert werden, allerdings muss sich an die Vorgaben des Frameworks gehalten werden, damit die Implementierung als Ganzes einfach bleibt. Einige Funktionen müssen dadurch aber komplexer implementiert werden, als es ohne Framework nötig ist.

Die Architektur sieht keine dedizierte Schicht für das User Interface vor, die Präsentationsschicht enthält sowohl die API als auch das User Interface. Da das User Interface als eigene Anwendung entwickelt wurde, kann es technisch nicht in denselben Modulen existieren wie

die API. Das User Interface als Anwendung ist selbst auch in die folgenden vier Module aufgeteilt:

- **root**: entspricht dem „rqms-api-general“ der Architektur
- **offline**: entspricht dem „rqms-api-offline“ der Architektur
- **online**: entspricht dem „rqms-api-online“ der Architektur
- **integration**: entspricht dem „rqms-api-interagratiion“ der Architektur

Das „root“-Modul hängt dabei von allen anderen Modulen ab und kann somit ein einheitliches User Interface bereitstellen. Die Logik ist aber auf die Module verteilt, sodass das „root“-Modul nur das Hauptmenü und die allgemeine Anwendungslogik bereitstellen muss. Alle Endpunkte werden in den Modulen selbst definiert und die Module werden nach Bedarf geladen. Somit ist die Anwendung beim initialen Laden kleiner und kann schneller verwendet werden.

Die Implementierung der API entspricht dem Architekturvorschlag. Für jedes Modul in dieser Schicht existiert ein Gradle-Projekt, welches entsprechend von den Projekten aus der Anwendungsschicht abhängt. Das „rqms-api-general“-Modul implementiert generische Funktionalität und liefert die Ressourcen des User Interface aus. Dabei ist es wichtig, dass alle Pfade, die nicht aufgelöst werden können, das User Interface ausliefern, da dieses gegebenenfalls für den Pfad eine Funktion bereitstellt. Die anderen Module enthalten die Controller-Klassen, welche die API-Endpunkte bereitstellen, Klassen für die Data Transfer-Objekte und Mapper-Klassen welche aus den Domain Model Objekten Data Transfer-Objekte erstellen. Die Data Transfer-Objekte werden zu JSON-Dokumenten serialisiert bzw. aus JSON-Dokumenten deserialisiert. Zur Fehlerbehandlung enthalten dieses Module sogenannte „ControllerAdvice“-Klassen, welche für die Behandlung von Exceptions-Methoden bereitstellen können. Jede Methode behandelt dabei eine spezielle Exception und erstellt eine entsprechende Antwort für den Client. Die behandelten Exceptions können im jeweiligen Modul, oder auch in den darunterliegenden Schichten erzeugt werden. Das „rqms-api-general“-Modul implementiert einen Fallback „ControllerAdvice“, welches für alle unbehandelten Exceptions aufgerufen wird.

5.2.2 Anwendungsschicht

Die Anwendungsschicht ermöglicht eine Verknüpfung der Logik aus den verschiedenen Modulen der Domainschicht. Dafür stellt die Anwendungsschicht übergeordnete Services bereit. Außerdem kann die Anwendungsschicht zukünftig für komplexere Exceptions genutzt werden, und somit als Adapter zwischen der Präsentationsschicht und der Domainschicht dienen. Diese Implementierung hält die Anwendungsschicht so klein wie möglich, da die Logik durch die Domainschicht abgebildet werden soll. Wie auch in der Architektur definiert, besteht die Anwendungsschicht aus vier Modulen, welche jeweils von den entsprechenden Modulen der Domainschicht abhängen.

5.2.3 Domainschicht

Die Domainschicht bildet die Architektur bestmöglich ab, dabei wurden die Module aus dem Architekturvorschlag als Gradle-Projekte umgesetzt. In jedem Modul sind die Arten von Klassen bzw. Interfaces in verschiedene Pakete unterteilt. Jedes Modul besitzt ein eigenes Präfix für die genutzten Packages, die Suffixe, welche die Arten der Klassen bzw. Interfaces sind aber in allen Packages gleich. Die folgenden Package-Namen werden genutzt:

- **aggregations**
- **entities**
- **factories**
- **helpers**
- **repositories**
- **services**
- **valueobjects**

Bis auf das „helpers“-Package, enthalten die Packages jeweils die entsprechende Art der Klassen bzw. Interfaces. Das „helpers“-Package wird benötigt, damit Funktionen, welche technischer Natur sind, aber im Domain Modul vorhanden sein müssen, entsprechend getrennt vom Domain Model gespeichert werden können. Da das Spring Boot Framework sich auch am Domain-Driven Design orientiert, können die Konzepte der Entities, Repositories und Services mit Framework-Funktionen umgesetzt werden. Entities werden mithilfe des „jakarta persistence“-Frameworks, welches von Spring Boot eingebunden wird, repräsentiert. Dabei muss für jede Entity eine Klasse erstellt werden, welche durch Annotations gekennzeichnet wird. Die Attribute dieser Klasse können dann jeweils einer Spalte in einer Datenbanktabelle zugeordnet werden, sodass die Klasse als ganzes in der Datenbank repräsentiert ist. Diese Technik wird als object-relational mapping (ORM) bezeichnet[vgl. KS10, S. 69]. Mithilfe von Repositories können diese Klassen erzeugt werden, indem ihre Attribute aus der Datenbank geladen werden. Die Entity-Klassen können auch mit einem Repository in der Datenbank persistiert werden. Ein Repository wird nur als Interface definiert und leitet immer vom „Jpa-Repository“ Interface ab, welches grundlegende Methoden bereitstellt. Die Implementierung des Interfaces wird automatisch zur Laufzeit bereitgestellt. Durch das Definieren eigener Methoden, können komplexere Abfragen erstellt werden. Services haben in Spring Boot dieselbe Aufgabe wie im Domain-Driven Design, sie sollen übergeordnete Funktionalitäten kapseln. Dafür benötigen sie häufig Repositories, um auf die entsprechenden Entities zugreifen zu können. Für Factories, Aggregations und Value Objects muss eigene Logik erstellt werden. Für diese Klassen sind die Aufgaben in der Implementierung wie folgt verteilt: Value Objects sind nach Möglichkeit als Java Records implementiert, somit sind sie unveränderlich und stellen automatisch Methoden zum Auslesen der Werte bereit. In dieser Implementierung enthalten Value Objects nur einfache oder gar keine Logik, sie dienen vorrangig zur Datenhaltung. Aggregations kapseln Entities und ihre Value Objects, sodass diese von außen manipuliert werden können, ohne dass Invarianten verletzt werden. Aggregations besitzen dabei nicht nur eine Referenz auf die Entity, sondern können diese auch mithilfe eines Repositories persistieren. Wenn eine Aggregation Value Objects oder Entities zur Verfügung stellt, welche mithilfe einer Factory erzeugt werden, enthält die Aggregation auch die Referenz auf die entsprechenden Factories. Somit ist eine Aggregation immer in der Lage, eine Entity mit ihren Abhängigkeiten zu verwalten. Factories haben in dieser Implementierung eine besondere

Rolle, sie ermöglichen es, eine konkrete Implementierung für Interfaces zu erstellen. Somit können Plugins ihre Implementierungen an einer Factory registrieren und diese Implementierungen können in der ganzen Anwendung genutzt werden. Factories bieten deshalb auch die Möglichkeit, alle konkreten Implementierungen aufzulisten.

Da die Entities direkt die relationale Datenbankstruktur referenzieren, wird diese Struktur auch im jeweiligen Domain Modul definiert. Jedes Modul besitzt dabei ein eigenes Datenbankschema. Diese Datenbankschemata werden über SQL Code erzeugt. Im Datenbankschema muss für jede Klasse eine Tabelle mit SQL erzeugt werden, welche die Attribute als Spalten enthält. Diese SQL-Anweisungen werden im Ordner „db/migrations“ im jeweiligen Gradle-Projekt hinterlegt. Somit ist auch die Datenbankstruktur immer pro Bounded Context definiert. Der SQL Code wird von Flyway[Ltd22b] ausgeführt, welches in der Infrastrukturschicht genauer erläutert wird(siehe Unterabschnitt 5.2.4).

5.2.4 Infrastrukturschicht

Die Infrastrukturschicht beinhaltet, wie in der Architektur beschrieben, die technischen Funktionen des RQMS. Zu diesen Funktionen zählen die Datenbankbindung, das Plugin Management, die Anbindung einer Metrikdatenbank (bzw. die Bereitstellung der Metriken für Prometheus[Aut22]) und die Einbindung der Groovy Runtime, welche für die Assertion Tests benötigt wird.

Die Datenbankbindung wird durch den PostgreSQL-Treiber bereitgestellt. Damit aber parallel mehrere Operationen auf der Datenbank ausgeführt werden können und nicht für jede Operation eine neue Verbindung aufgebaut werden muss, verwendet Spring Boot einen Connection Pool. Je nach Umgebung muss dieser Connection Pool entsprechend konfiguriert werden. Für die Migration der Datenbankschemata wird Flyway[Ltd22b] genutzt. Flyway ermöglicht es, SQL-Anweisungen auszuführen, um die Datenbank in den gewünschten Zustand zu überführen. Dabei werden Anweisungen in Dateien zusammengefasst, deren Dateiname einer Versionsnummer entspricht. Wenn nicht nur eine Migrationsdatei, sondern eine Datei zum Zurückrollen einer Migration existiert, ermöglicht Flyway das Up- und Downgrade der Datenbank. Flyway erkennt automatisch, welche Migrationen noch nicht ausgeführt wurden und führt die fehlenden Migrationen beim Start der Anwendung durch[Ltd22a].

Das Plugin Framework PF4J[Høy+12] erlaubt das Laden von Plugins in der Laufzeit der Anwendung. Im Gegensatz zur Architektur musste das Plugin Interface in die Infrastrukturschicht verschoben werden, damit es von PF4J genutzt werden kann. Alle von Plugins bereitgestellten PF4J Extensions können nach dem Laden der Plugins als Spring Beans verwendet werden. Für das Laden der Plugins wird ein Plugin Manager benötigt, welcher auch von der Infrastrukturschicht bereitgestellt wird.

Damit das RQMS auch zur Laufzeit überwacht werden kann, wird das von Spring Boot eingebundene Interface für Prometheus[Aut22] verwendet. Dieses erlaubt es, in der gesamten Anwendung Metriken zu definieren, welche dann Prometheus zur Verfügung gestellt werden. Ein Beispiel für eine Metrik ist die Ausführungszeit der Groovy-Skripte, welche mithilfe

dieses Logik erfasst wird. Spring Boot definiert einige Standardmetriken (z. B. Anzahl der Log-Einträge), aber die Metriken mit dem meisten Mehrwert werden in der Business-Logik erfasst. Daher ist es auch notwendig, dieses Interface in der Infrastrukturschicht bereitzustellen, damit es in allen überliegenden Schichten verwendet werden kann.

Zur Ausführung der Assertion Tests wird die Groovy Runtime benötigt. Diese wird als Implementierung des „ScriptEvaluator“ Interfaces als Spring Bean bereitgestellt. Das Interface besitzt nur eine Methode, welche das auszuführende Skript als Text, den Namen der Kontextvariable und den Inhalt der Kontextvariable als Parameter erhält. Der Rückgabewert dieser Methode ist ein Wahrheitswert, kann also nur die Werte „wahr“ oder „falsch“ annehmen. Nur wenn ein Fehler auftritt, wird eine entsprechende Exception geworfen.

5.3 Umsetzung der Use Cases

Eine Implementierung eines RQMS soll alle bekannten Use Cases (siehe Unterabschnitt 3.3.2) bestmöglich umsetzen. Im Folgenden wird diskutiert, inwieweit diese Use Cases umgesetzt werden konnten, wie die Architektur dabei unterstützt hat, diese Use Cases zu implementieren und wo die Architektur für diese Implementierung hinderlich ist.

5.3.1 Technische Use Cases

UC1 - Einfache Bereitstellung

Das einfache Bereitstellen des RQMS erfolgt über Docker. Dabei kann der Nutzer eine Docker Compose-Datei auf einem Rechner erstellen und mit Docker ausführen. Das RQMS wird dabei mit allen Abhängigkeiten gestartet und kann direkt genutzt werden. Das benötigte Docker Image wird wie in Unterabschnitt 5.1.6 beschrieben, für jedes Release des RQMS gebaut und bereitgestellt. Dieser Use Case (siehe Abschnitt 3.3.2) ist damit vollständig implementiert.

UC2 - Authentifizierung über externes System

Durch die Nutzung von Spring Boot kann auch das Modul Spring Security genutzt werden. Die Implementierung unterstützt durch die Einbindung von Spring Security die Authentifizierung mithilfe von OIDC. In den Einstellungen des RQMS müssen nur die URL für die OIDC Content Discovery[Sak+14], die Client-ID und das Client-Secret für den OIDC Provider hinterlegt werden. Für die Entwicklung und das Testen kann ein OIDC Mock Server genutzt werden (z. B. Open ID Connect Server Mock[Ale18]). Der Use Case (siehe Abschnitt 3.3.2) ist daher implementiert.

UC3 - Hochverfügbarkeit

Die Hochverfügbarkeit des RQMS wird erreicht, indem die Anwendung selbst zustandslos arbeitet. Alle benötigten Zustände werden über die Datenbank zwischen den Anwendungsinstanzen geteilt. Wenn mehrere Instanzen des RQMS auf verschiedenen Hardware-Systemen

(am besten in unterschiedlichen Regionen) gestartet werden und die Datenbank auch hochverfügbar ist, ist das gesamte RQMS hochverfügbar und der Use Case (siehe Abschnitt 3.3.2) damit vollständig implementiert.

UC4 - Sicherungsfähigkeit

Das RQMS selbst ist einfach sicherungsfähig, da es PostgreSQL als Datenbank nutzt und alle Zustandsdaten dort persistiert. Diese Datenbank erlaubt es, den Zustand zu einem Zeitpunkt zu sichern und später wiederherzustellen. Alle installierten Plugins sollten in ein eigenes Docker Image kopiert werden und sind somit in der Docker Registry gesichert. Daher können sowohl Plugins als auch alle Zustandsdaten wiederhergestellt werden und dieser Use Case (siehe Abschnitt 3.3.2) ist vollständig implementiert.

5.3.2 Relevance Test Management

UC5 - Manuelle Relevanztests

Das RQMS erlaubt das Erstellen und Durchführen von Relevanztests. Dabei kann ein Nutzer

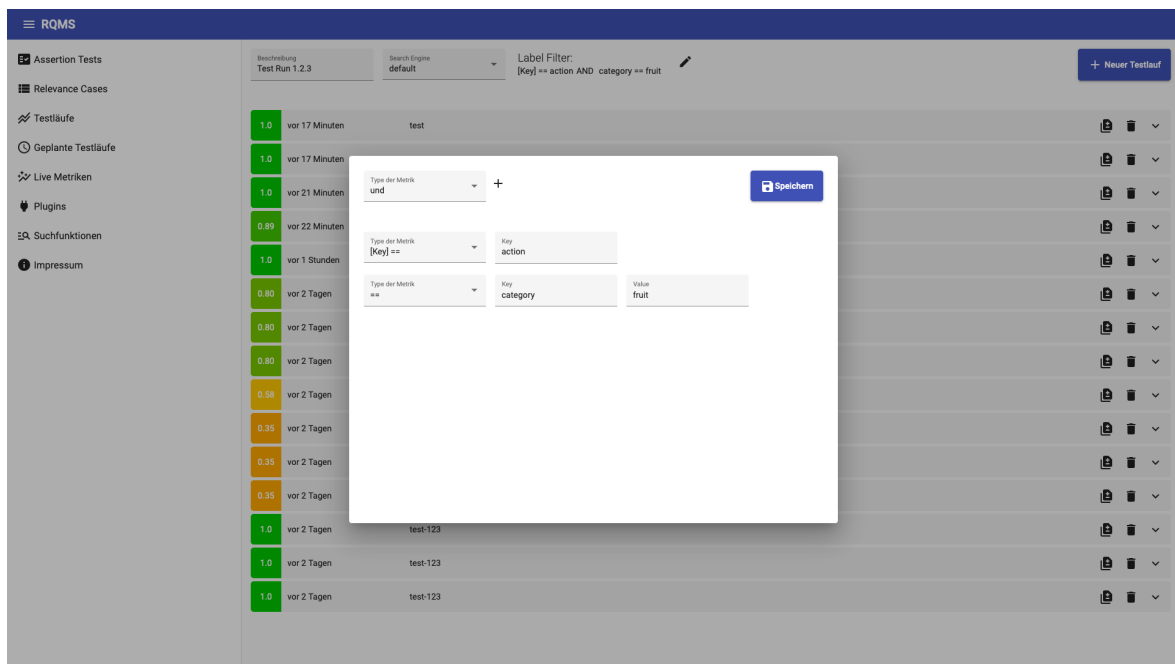


Abbildung 5.2: Screenshot der Testlaufübersicht im RQMS mit geöffnetem Label Filter Editor

Relevance Cases anlegen, welche eine Judgment List beinhalten. Für einen Relevance Case kann dann eine Metrik ausgewählt werden, um einen Relevance Score zu berechnen. Wenn ein Testlauf durchgeführt wird, muss eine Menge von Relevance Cases ausgewählt werden, die in diesem Testlauf enthalten sind und für diese wird mithilfe der ausgewählten Metrik ein Relevance Score errechnet. Die Logik zur Durchführung der Tests ist im Domain Model des RQMS implementiert. Durch die Kapselung ist es einfach, diese Logik zu entwickeln und zu testen. Allerdings erschwert die Trennung die Persistierung der Daten. Der ursprüngliche Use

Case (siehe Abschnitt 3.3.2) ist damit im Kern umgesetzt. In Abbildung 5.2 ist der geöffnete Label Filter Editor zu sehen. In diesem Dialog lässt sich der Label Filter definieren, welcher bei der Ausführung die Relevance Cases selektiert.

UC6 - Judgment List erstellen

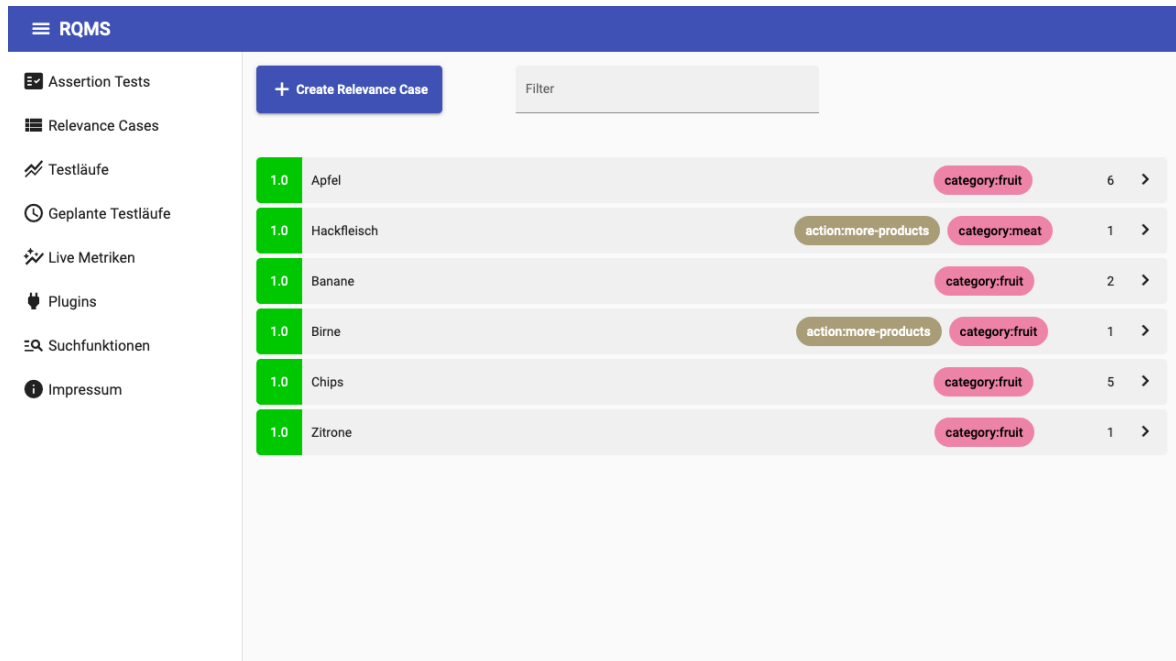


Abbildung 5.3: Screenshot der Relevance Cases-Übersicht im RQMS

Die Architektur erlaubt die Verwaltung von Judgment Lists in Relevance Cases. Die durch Plugins bereitgestellte Suchfunktion ermöglicht das Suchen von Dokumenten, sodass ein Relevance Case einfach erstellt werden kann. In Abbildung 5.3 ist die Übersichtsseite der Relevance Cases zu sehen, auf welcher sich der Knopf zum Erstellen eines neuen Relevance Cases befindet. Wenn ein Nutzer diesen Knopf drückt, wird er in den Relevance Case Editor (siehe Abbildung 5.4) weitergeleitet und kann dort einen neuen Relevance Case anlegen. Dieser Use Case (siehe Abschnitt 3.3.2) ist somit umgesetzt.

UC7 - Judgment List bearbeiten

Relevance Cases, welche im RQMS die Judgment Lists beinhalten, können im Relevance Case Editor (siehe Abbildung 5.4) bearbeitet werden. Die Nutzer haben die Möglichkeit, den Namen des Relevance Cases, seine Labels, die verwendete Metrik, die genutzte Suchanfrage und natürlich die erwarteten Ergebnisse anzupassen. Durch Drag and Drop können die Ergebnisse in ihrer Position verändert, neue Ergebnisse hinzugefügt oder Ergebnisse gelöscht werden. Der Use Case „UC 7 - Judgment List bearbeiten“ (siehe Abschnitt 3.3.2) ist damit vollständig im RQMS implementiert.

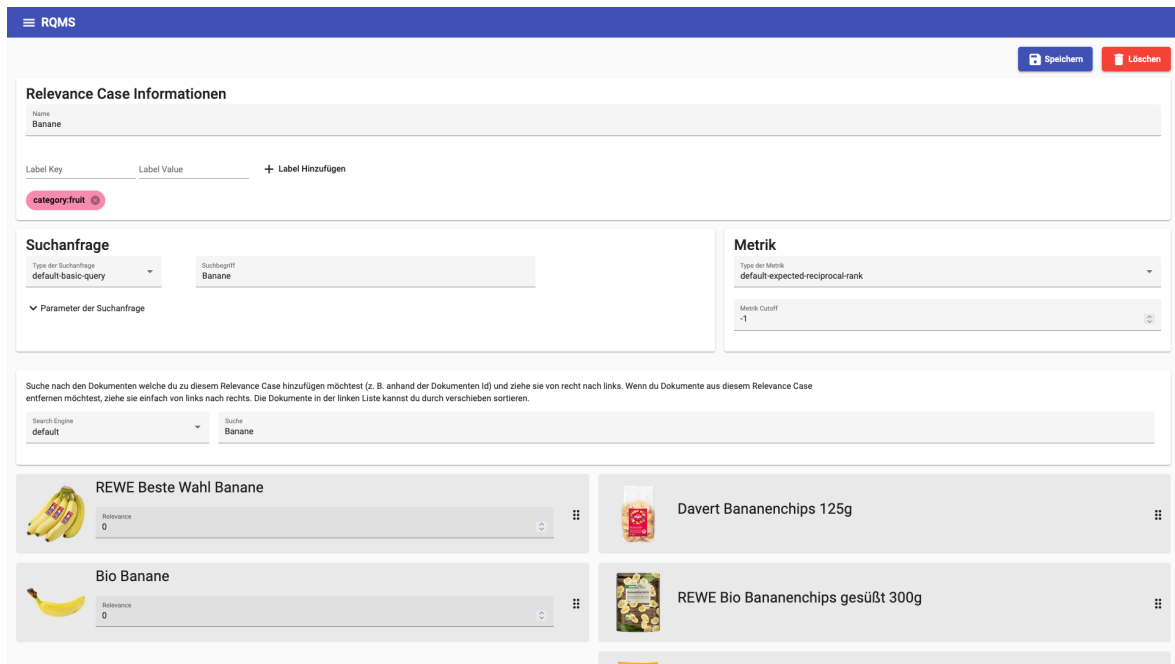


Abbildung 5.4: Screenshot des Relevance Case Editors im RQMS

UC8 - Judgment List löschen

Im Relevance Case Editor (siehe Abbildung 5.4) gibt es einen Knopf, um den aktuellen Relevance Case zu löschen. Wenn ein Relevance Case gelöscht wird, werden alle Informationen inklusive der Suchanfrage und der erwarteten Ergebnisse aus dem RQMS entfernt. Das Löschen eines Relevance Cases ist im Use Case „UC 8 - Judgment List löschen“ beschrieben, dieser ist damit vollständig implementiert.

UC9 - Automatische Relevanztests

Automatische Relevanztests können durch einen Test Schedule ausgeführt werden. Dabei führt das RQMS zu allen definierten Zeitpunkten die ausgewählten Relevance Cases aus. Durch Plugins ist es aber auch möglich, automatisch Test Schedules anzulegen oder selbstständig Tests auszuführen. Die im Use Case (siehe Abschnitt 3.3.2) beschriebene Logik kann also durch ein Plugin umgesetzt werden oder alternativ können Test Schedules genutzt werden, um die Tests zeitbasiert auszuführen.

UC10 - Judgment List Vorschläge

Die Architektur trennt die verschiedenen Domain Models durch Bounded Contexts. Diese Trennung erschwert die Umsetzung von domainübergreifenden Features, sowie der Generierung von Relevance Cases aus Nutzungsdaten. Durch Logik in der Anwendungsschicht lassen sich die Bounded Contexts aber so verküpfen, dass diese Features trotzdem umgesetzt werden können. Der Wechsel in den Schichten erzwingt dann auch eine Umwandlung und nötigt den Entwickler zur Prüfung der zu übertragenden Daten. Diese Prüfung entspricht dem von Eric Evans beschriebenen „Anti Corruption Layer“ [Eva04, S. 364ff], sodass die Architektur die

Entwickler implizit zur Einhaltung ihrer Richtlinien benötigt. Aktuell fehlt in der Anwendungsschicht noch Logik, um die beiden Domains zu verknüpfen. In der Business-Logik ist der Use Case (siehe Abschnitt 3.3.2) umgesetzt.

UC11 - Assertion Test Management

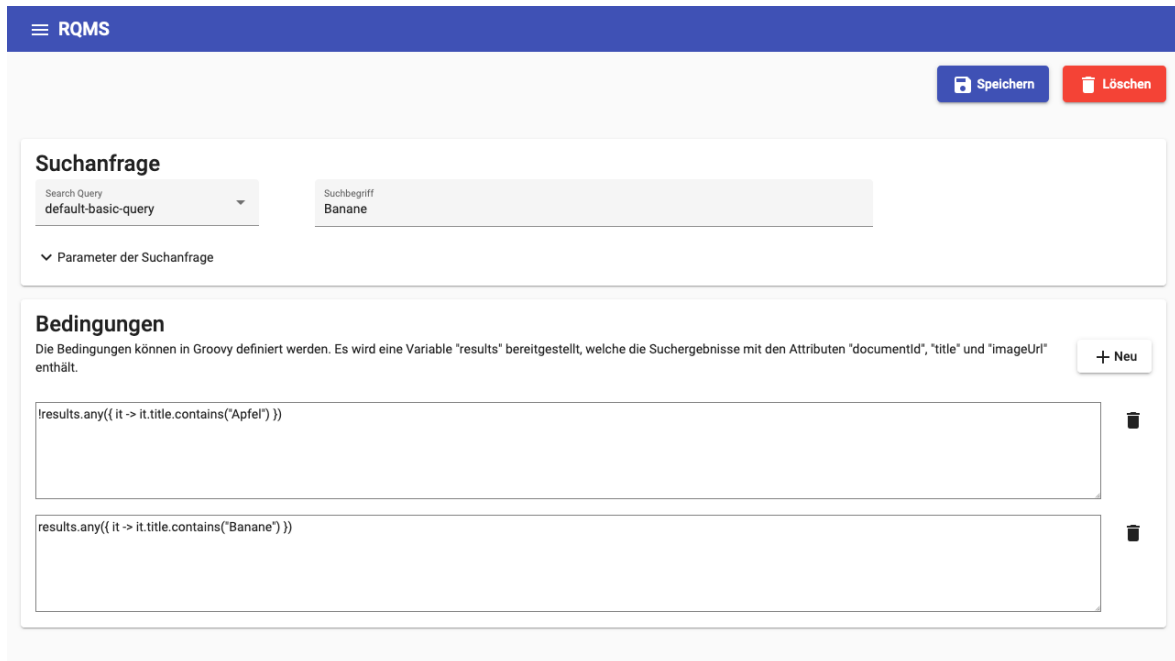


Abbildung 5.5: Screenshot des Assertion Test Editors im RQMS

Das RQMS erlaubt das Erstellen und Durchführen von Assertion Tests. Dabei kann ein Nutzer Assertion Test anlegen und die Bedingungen in Groovy[Kin20] definieren. Bevor die Relevance Tests ausgeführt werden, führt das RQMS immer alle Assertion Tests aus. Jeder Assertion Test muss eine Suchanfrage und mindestens eine Bedingung enthalten. Eine Bedingung kann auf die Variable „results“ zugreifen, welche eine Liste von „SearchResult“-Objekten (siehe Abschnitt 4.3.4) enthält. Diese Liste kann genutzt werden, um Bedingungen auf dem Titel, der Dokumenten ID oder der URL des hinterlegten Bildes zu definieren. Wenn ein Plugin eine Implementierung des „SearchResult“ Interfaces bereitstellt, welche zusätzliche Eigenschaften besitzt, können diese direkt in den Bedingungen genutzt werden. Durch diese flexible Lösung ist der Use Case UC 11 - Assertion Test Management (siehe Abschnitt 3.3.2) vollständig umgesetzt.

5.3.3 Online-Metriken

UC12 - Erstellen eines A/B-Tests

Das RQMS erlaubt es, A/B-Tests zu verwalten, entweder über das User Interface oder über ein Plugin, welches die A/B-Tests automatisch anlegt, aktualisiert und löscht. Das RQMS kann dabei A/B-Tests mit beliebig vielen Varianten behandeln, allerdings muss ein A/B-Tests

immer mindestens zwei Varianten besitzen, wovon eine Variante die Kontrollgruppe repräsentiert. Die Architektur gibt hierbei einen guten Rahmen für die Umsetzung vor, besonders die Behandlung von Varianten kann einfach umgesetzt werden. Aktuell fehlt im User Interface noch die Möglichkeit, die A/B-Tests zu verwalten. In der Business-Logik ist der Use Case (siehe Abschnitt 3.3.2) umgesetzt.

UC13 - Auswerten eines A/B-Tests

Durch eine Vielzahl an Metriken, die auch von Plugins erweitert werden kann, ermöglicht das RQMS die Auswertung von A/B-Tests. Dabei ist es möglich, die Metriken über den Zeitverlauf, für ein Intervall oder als Gesamtwert zu berechnen. Für die Berechnung wird immer eine A/B-Testvariante benötigt. Aktuell fehlt im User Interface noch die Möglichkeit, diese Werte abzufragen. In der Business-Logik ist der Use Case (siehe Abschnitt 3.3.2) umgesetzt.

UC14 - Fehlende Ergebnisse erkennen

Die Nutzungsdaten, welche im RQMS gespeichert sind, erlauben es, eine Liste aller Suchbegriffe zu ermitteln, für welche keine Ergebnisse gefunden wurden. Diese Liste kann mithilfe des Analytics Service (siehe Abschnitt 4.4.2) ausgegeben werden. Auf Grundlage dieser Liste können dann die fehlenden Ergebnisse erkannt werden. Aktuell fehlt im User Interface noch die Möglichkeit, diese Werte abzufragen. Der Analytics Service setzt diesen Use Case (siehe Abschnitt 3.3.2) in der Business-Logik um.

UC15 - Beurteilung Qualitätsentwicklung

Zur Beurteilung der Qualitätsentwicklung stellt das RQMS dieselben Metriken bereit, welche auch für die Auswertung von A/B-Tests verwendet werden können. Statt einer Testvariante muss eine „SearchEngine“ angegeben werden, für welche die Qualitätsentwicklung dargestellt werden soll. Aktuell fehlt im User Interface noch die Möglichkeit, diese Werte abzufragen. In der Business-Logik ist der Use Case (siehe Abschnitt 3.3.2) umgesetzt.

6 Fazit

6.1 Allgemein

Die Arbeit hat die Anforderungen an ein RQMS aufgezeigt und eine Architektur definiert. Mit dieser Architektur kann eine RQMS entwickelt werden, welches allen Anforderungen genügt. Als Basis für diese Entwicklung kann die Referenzimplementierung genutzt werden, welche in Kapitel 5 vorgestellt wurde. Damit eine erfolgreiche Entwicklung und Einführung eines RQMS in einer Organisation gelingt, sollten die in Kapitel 2 erwähnten Voraussetzungen erfüllt sein. Somit bietet diese Arbeit eine Grundlage für die Entwicklung und Einführung eines RQMS in Organisationen.

6.2 Technisch

Die vorgestellte Architektur orientiert sich am Domain-Driven Design, was viele Vorteile in der Entwicklung mit sich bringt. Ein großer Vorteil ist die starke Trennung durch die Layered Architecture und Bounded Contexts, durch welche eine parallele Entwicklung ermöglicht wird. Allerdings benötigt Domain-Driven Design auch einen starken Rückhalt im Entwicklungsteam. Daher ist es für eine Weiterentwicklung von Vorteil, wenn alle Teammitglieder auch Domain-Driven Design kennen und anwenden. Diese Notwendigkeit ist der große Nachteil des Domain-Driven Design. Es besteht die Gefahr, dass ein Entwicklungsteam, welches nicht Domain-Driven Design praktiziert, die starke Trennung der Layered Architecture und Bounded Contexts auflöst und dadurch nur noch die Komplexität des Domain-Driven Designs ohne seine Vorteile weiterbesteht. Diese Situation kann sogar soweit führen, dass die Anwendung neu entwickelt werden muss. Daher ist es aus technischer Sicht notwendig, dass alle zukünftig an der Entwicklung beteiligten Personen ein Verständnis für Domain-Driven Design besitzen und dieses gerne anwenden. Andernfalls sollte eine neue Architektur entwickelt werden, mit welcher das Entwicklungsteam effektiv arbeiten kann.

Die Architektur selbst ist, wie in Kapitel 5 beschrieben, geeignet, um alle Use Cases des RQMS umzusetzen. Die Referenzimplementierung ist eine gute Grundlage, kann aber noch verbessert werden. Insbesondere die Nutzung der von Spring Boot bereitgestellten Features kann noch erhöht werden, gerade im Bereich der Persistierung. Durch die Nutzung von PF4J und Spring Boot sollte es sehr einfach sein, Message Queues wie z. B. Kafka und auch Search Engines wie z. B. Elasticsearch durch Plugins anzubinden.

6.3 Organisatorisch

In den Kapiteln Kapitel 2 und Kapitel 3 werden die organisatorischen Rahmenbedingungen beschrieben, für welche das RQMS entwickelt wurde. Für einen Einsatz in einer Organisation sollte diese eine agile Arbeitsweise nutzen (z. B. Scrum), eine Suchfunktion besitzen, deren Inhalte sich dynamisch ändern oder deren Nutzer komplexe Anfragen stellen und genug Teamkapazität besitzen, um das RQMS auch effektiv nutzen zu können. Ein Team, welches das RQMS effektiv nutzen will, sollte ein bis zwei Relevance Engineers, einen Content Curator, zwei oder mehr Software Engineers, einen Product Owner und einen Scrum Master besitzen. Somit ist sichergestellt, dass genug Personen das RQMS regelmäßig nutzen und die Suchfunktion weiterentwickeln können.

Diese vergleichsweise hohe Anforderung an die Teamgröße lässt erkennen, dass ein RQMS vor allem in großen Organisationen effektiv genutzt werden kann. Dabei spielt auch die Nutzung der vom RQMS verwalteten Suchfunktion eine wichtige Rolle. Wenn es sich für ein Unternehmen nicht lohnt, genug Personal für die Verbesserung der Suche bereitzustellen, sollten Alternativen zur Nutzung eines RQMS in Betracht gezogen werden. Die alternativen Produkte, wie sie in Abschnitt 3.4 aufgelistet sind, können dann auch einen Mehrwert bieten. Das RQMS erweitert die Funktionalität dieser Produkte und ist daher besonders für Organisationen geeignet, welchen der Funktionsumfang der anderen Anwendungen nicht mehr ausreicht.

6.4 Ausblick

Die in dieser Arbeit aufgezeigten Anforderungen an ein RQMS sind durch Desk-Research-Analysen entstanden. Wenn ein RQMS unter Berücksichtigung dieser Arbeit entwickelt wurde, sollten weitere Analysen folgen, die zusammen mit den Stakeholdern die Anforderungen an das RQMS verbessern. Erst durch einen iterativen Entwicklungsprozess können alle Anforderungen einer Organisation, den Nutzern des RQMS und den Endnutzern der Suchfunktion an das RQMS ermittelt werden. Daher ist weitere wissenschaftliche Forschung notwendig, um diese Anforderungen zu ermitteln und in die Architektur für das RQMS zu integrieren.

7 Verzeichnisse

7.1 Abkürzungsverzeichnis

API	Anwendungsschnittstelle
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
LDAP	Lightweight Directory Access Protocol
OIDC	OpenID Connect
RQMS	Relevance Quality Management System
SaaS	Software as a Service
SAML	Security Assertion Markup Language
SQL	Structured Query Language

7.2 Glossar

A/B-Test

Ein A/B-Test ermöglicht das Bereitstellen einer Funktionalität für eine vordefinierte Benutzergruppe. Dabei wird die Nutzergruppe, welche die neue Funktionalität nutzt, mit der Nutzergruppe verglichen, welche die alte Funktionalität nutzt. Durch die Erfassung von Metriken ist es möglich, nach einem definierten Zeitraum eine Aussage zu treffen, ob die neue Funktionalität gegenüber der alten Funktionalität eine Änderung des Nutzerverhaltens erzeugt. Ein A/B-Test kann dabei mehr als nur eine Testgruppe enthalten. In der Theorie ist es möglich, dass ein A/B-Test beliebig viele Nutzergruppen enthält, aber in der Praxis muss eine Nutzergruppe eine Mindestgröße aufweisen, damit ein vergleichbares Nutzerverhalten vorhanden ist[siehe Sch20, S. 14ff].

Annotations

Annotations sind eine Form von Metadaten. Sie liefern Daten über ein Programm, die nicht Teil des Programms selbst sind[Ora22].

Anwendungsschnittstelle (API)

Eine Anwendungsschnittstelle (im Englischen „Application programming interface“) ist ein Teil eines Software-Systems, welcher anderen Systemen eine Anbindung an das Software-System ermöglicht.

Aspektorientierte Programmierung

Das Programmierparadigma der aspektorientierten Programmierung (AOP) ermöglicht es, generische Funktionalität unabhängig von den Implementierungen auszulagern. Die Geschäftslogik und die technische Logik einer Anwendung werden dabei getrennt. Ein Beispiel ist die Transaktionsverwaltung in Spring Boot, welche durch eine Markierung an einer Methode aktiviert wird und eine neue Datenbanktransaktion für jeden Methodenaufruf erstellt.

Assertion Test

Ein Assertion Test definiert eine oder mehrere Bedingungen, welche getestet werden können. Jeder dieser Bedingungen kann entweder erfüllt oder nicht erfüllt sein. Diese Art von Test zeichnet sich durch das binäre Ergebnis aus und ist der Gruppe der Offline-Metriken zugeordnet[siehe Sch20, S. 14ff].

Bot

Das Wort Bot stammt vom englischen Wort „Robot“ ab und beschreibt eine Programm, welches sich wiederholende Aufgaben (fast) vollautomatisch abarbeitet. Ein Bot benötigt dabei keine Interaktion mit einem menschlichen Benutzer.

Content Curator

Der Content Curator hat die inhaltliche Verantwortung, die Relevanz der Suchfunktion zu verbessern. Dabei benötigt der Content Curator ein tiefes Domänenwissen und eine gute Kommunikation zu den Domain Experts. Der Content Curator hat die Aufgabe, die Relevanz der Suchergebnisse anhand des Nutzerverhaltens zu bewerten und inhaltliche Verbesserungspotenziale zu erkennen. Diese setzt er gemeinsam mit einem Relevance Engineer um.

Crawler

Ein Crawler bzw. Webcrawler ist eine Anwendung, welche Webseiten im Internet durchsucht und analysiert. Häufig werden Crawler von Suchmaschinen zur Indexierung von Webseiten genutzt. Crawler gehören zur Kategorie der Bots, die (fast) vollautomatisch wiederholende Aufgaben abarbeiten.

Cron Expression

Eine Cron Expression ist ein Ausdruck, welcher angibt, in welchen Zeitintervallen eine Aktion ausgeführt werden soll. Der Ausdruck besteht aus fünf Werten, welche die Minute, die Stunde, den Tag im Monat, das Monat und den Wochentag angeben.

Datenbankschema

Ein Datenbankschema beschreibt formal die Struktur von Daten. In der Datenbank umfasst ein Schema alle vom Schemaeigentümer erstellten Objekte. Dazu gehören üblicherweise Tabellen, Ansichten (Views) und Sequenzen.

Dependency Injection

Die Dependency Injection ist ein Entwurfsmuster in der objektorientierten Programmierung, welches dazu genutzt wird, um die Abhängigkeiten eines Objekts zur Laufzeit zu verwalten. Alle Abhängigkeiten für ein zu erzeugendes Objekt sind an einem zentralen Ort hinterlegt und werden bei der Erzeugung des Objekts übergeben, anstatt vom Objekt selbst erzeugt zu werden.

Docker

Docker ist eine Anwendung zur Containervirtualisierung. Ein Docker Container umfasst neben dem auszuführendem Programm auch alle Abhängigkeiten. Somit lassen sich Anwendungen als Docker Container leicht bereitstellen, transportieren und installieren.

Docker Compose

Docker Compose ist ein Werkzeug, um mehrere Anwendungen als Docker Container zu definieren und zu betreiben. In der Docker Compose Konfigurationsdatei werden nicht nur die Container konfiguriert, sondern auch die Abhängigkeiten zwischen den Containern definiert. Somit können einfache Netzwerkstrukturen und Anwendungsabhängigkeiten erstellt werden.

Docker Image

Ein Docker Image enthält alle Dateien und Anweisungen, welche benötigt werden, um einen Docker Container zu starten.

ECMAScript

ECMAScript ist der standardisierte Kern der Skriptsprache JavaScript. Ursprünglich wurde JavaScript für die Erzeugung von dynamischen Inhalten in Browsern entwickelt. Heutzutage wird JavaScript aber auch für die Entwicklung von Server-, Desktop- und Microcontroller-Anwendungen genutzt.

Exception

Eine Exception ist ein Verfahren, bei welchem Informationen über einen Programmzustand - häufig einen Fehlerzustand - an eine andere Komponente im Programm zur Weiterbehandlung übergeben werden. Dabei ist sichergestellt, dass sich das Programm jederzeit in einem definierten Zustand befindet.

Extreme Programming

Extreme Programming ist eine Methode der agilen Softwareentwicklung, welche das Lösen der Programmieraufgabe priorisiert. Die formalisierten Prozessschritte haben im Extreme Programming eine geringe Bedeutung. Extreme Programming ist ein agiles

Vorgehensmodell, da sich den Anforderungen des Kunden in iterativen Schritten annähert wird.

Git

Git ist eine Anwendung zur Versionsverwaltung von Dateien. Git wurde ursprünglich von Linus Torvalds entwickelt und speichert die Daten verteilt ab. Es existiert kein zentraler Server, sondern alle Nutzer besitzen eine vollwertige Kopie des gesamten Versionsverlaufs der Daten. Git wird sehr häufig zur Versionierung von Source Code genutzt.

Helm Chart

Ein Helm Chart ist eine Vorlage für die Installation und Auslieferung einer Anwendung in Kubernetes. An einem Helm Chart können nutzerspezifische Einstellungen vorgenommen werden, welche die Installation der Anwendungen anpassen (z. B. Anzahl der Instanzen, Zugangsdaten). Der Package Manager kann mit einem Helm Chart auch automatische Updates übernehmen.

HTTP Load Balancer

Ein HTTP Load Balancer verteilt Anfragen auf mehrere Anwendungsserver, welche diese Anfragen beantworten. Dabei ist es wichtig, dass der HTTP Load Balancer die Anfragen intelligent verteilt. Zum einen darf kein Anwendungsserver überlastet werden, zum anderen sollten alle Anfragen eines Nutzers immer auf denselben Anwendungsserver verteilt werden, damit dieser die Nutzerdaten vorhalten kann. Die gesamte Arbeit soll durch diese Verteilung effizienter gestaltet werden.

Integrierte Entwicklungsumgebungen (IDE)

Eine integrierte Entwicklungsumgebung stellt Entwicklern eine einheitliche Oberfläche zur Verfügung, mit welcher die Entwickler ohne Medienbrüche arbeiten können. Eine integrierte Entwicklungsumgebung bündelt eine Sammlung von Anwendungen, welche üblicherweise zur Entwicklung verwendet werden und verknüpft die Informationen aus diesen Programmen. Eine integrierte Entwicklungsumgebung bietet durch die verknüpften Informationen einen wesentlichen Mehrwert für die Entwickler. Die Entwickler können zum Beispiel im Code Referenzen zu Datenbankobjekten nutzen, um diese zu untersuchen.

Judgment List

Eine Judgment List besteht aus einer Abfrage und einer Liste an erwarteten Ergebnissen. Der Relevance Engineer und der Content Curator übernehmen die Verwaltung der Judgment Lists, sie erstellen, verändern und löschen diese nach Bedarf. Mithilfe einer Judgment List und einer Metrik kann eine Suchfunktion für die in der Judgment List hinterlegten Abfrage evaluiert werden[siehe Sch20, S. 15].

Kafka

Apache Kafka (kurz Kafka) ist eine Software zur Verarbeitung von Datenströmen. Kafka kann Datenströme speichern, verarbeiten und Drittsystemen bereitstellen. Das verteilte Transaktions-Log ist der Kern von Apache Kafka und ermöglicht es, alle Funktionen skalierbar anzubieten.

Kanban

Kanban ist eine Methode, die ihren Ursprung in der Produktionsprozesssteuerung hat. Auch in der Softwareentwicklung kann Kanban genutzt werden und zählt zu den agilen Vorgehensmodellen. In der Softwareentwicklung soll Kanban die Anzahl der Aufgaben, an welchen parallel gearbeitet wird, reduzieren. Diese Reduktion soll die Qualität steigern und den Fokus der Entwickler auf die wichtigste Aufgabe lenken. Alle Aufgaben in Kanban sind priorisiert und werden der Reihe nach abgearbeitet. Neue Aufgaben können in Kanban nur angefangen werden, wenn alle alten Aufgaben abgeschlossen sind. Somit ist sichergestellt, dass sich zu jedem Zeitpunkt nur eine definierte Menge an Aufgaben in der Entwicklung befinden.

Kotlin

Die Programmiersprache Kotlin ist statisch und typisiert. Der vom Kotlin Compiler erzeugte Bytecode kann auf der JVM ausgeführt werden. Daher ist Kotlin zu Java bzw. Groovy kompatibel. Kotlin kann außerdem auch in JavaScript-Quellcode oder mittels LLVM in Maschinencode übersetzt werden.

Kubernetes

Kubernetes ist ein Orchestrierungssystem zur Verwaltung von Container-Anwendungen. Kubernetes wurde 2014 von Google veröffentlicht und ist in der Programmiersprache Go geschrieben. Häufig wird der Name Kubernetes auch mit „K8s“ abgekürzt. Seit 2015 ist Kubernetes Teil der Cloud Native Foundation.

Kustomize

Kustomize ist ein Werkzeug zur Verwaltung von Kubernetes-Konfigurationen. Kustomize vermeidet dabei Vorlagendateien und erlaubt es, bestehende Konfigurationen durch Operationen zu verändern.

Lightweight Directory Access Protocol (LDAP)

Das Lightweight Directory Access Protocol (LDAP) dient zur Abfrage und Änderung von Informationen dezentraler Verzeichnisdienste. Das Netzwerkprotokoll wird in der Praxis von fast jedem Produkt, welches mit Nutzerdaten arbeitet, implementiert.

Minimal Viable Product

Ein Minimal Viable Product ist die erste funktionsfähige Version eines Produkts, welche nur den minimal nötigen Funktionsumfang bietet. Der Funktionsumfang muss groß genug sein, um einen brauchbaren Nutzen für die Nutzer zu erzeugen, sodass diese das

Produkt auch einsetzen. Ziel eines Minimal Viable Product ist es, möglichst schnell aus Nutzerfeedback zu lernen, das Produkt auf die Nutzer zu optimieren und Fehlentwicklungen aufgrund von falschen Anforderungen zu vermeiden.

Object-relational mapping (ORM)

Das object-relational mapping (ORM) ermöglicht das Persistieren von Objektzuständen in der Datenbank. Dabei umfasst object-relational mapping alles vom Prozess, wie der Objektzustand auf die Datenbankspalten abgebildet wird, bis hin zur Frage, wie Abfragen über die Objekte durchgeführt werden können[vgl. KS10, S. 69].

OpenID Connect (OIDC)

OpenID Connect (OIDC) ist ein Framework zur Authentifizierung. Dieses basiert auf OAuth 2.0 und wird von der OpenID Foundation überwacht. OpenID Connect definiert die Mechanismen, welche eine automatische Konfiguration und standardisierte Nutzung von OAuth 2.0 ermöglichen (z. B. OpenID Connect Discovery).

Orchestrierungssystem

Ein Orchestrierungssystem kombiniert mehrere Services zu einer Komposition. Diese Komposition beschreibt einen ausführbaren Geschäftsprozess, bei welchem jeder Service nur einen eingeschränkten Sichtbereich hat und für alle Prozesse in diesem Bereich entscheidet. Ein Beispiel für ein Orchestrierungssystem ist Kubernetes.

Pair Programming

Pair Programming ist eine Arbeitstechnik, die sich häufig bei agilen Vorgehensweisen zur Softwareentwicklung findet. Sie ist ein wichtiger Bestandteil von Extreme Programming und soll die Softwarequalität steigern. Bei der Entwicklung von Quellcode arbeiten zwei Entwickler an einem Arbeitsplatz, wobei ein Entwickler Code schreibt und der andere diesen liest. Beim Lesen des Quellcodes soll der zweite Entwickler sich Gedanken über die Problemstellung machen, Probleme und Potenziale im Quellcode erkennen und diese sofort ansprechen. Durch das direkte Gespräch kann der Quellcode angepasst werden und die Softwarequalität steigt. Üblicherweise wechseln die Entwickler in einem definierten Zeitintervall die Rollen.

Persona

Persona (pl. Personae oder Personae) stammt aus dem angelsächsischen Sprachraum und bezeichnet Aspekte eines Menschen, der anderen präsentiert oder durch diese wahrgenommen wird bzw. eine Rolle oder eine Figur, die durch einen Autor oder Schauspieler eingenommen wird[Har09a, S. 29].

PF4J Extension

Eine „Extension“ ist eine konkrete Implementierung eines „Extension Points“. Um die Funktionalität einer Anwendung zu erweitern, muss diese einen sogenannten „Extension Point“ definieren. Dies ist eine Schnittstelle oder abstrakte Klasse, die ein bestimmtes Verhalten für eine „Extension“ definiert[Høy+12].

PostgreSQL

PostgreSQL ist ein objektrelationales Datenbankmanagementsystem (ORDBMS), welches von einer Open Source Community entwickelt wird. Häufig wird der Name auch mit „Postgres“ abgekürzt.

Precision

Precision ist eine Metrik, welche ausdrückt, wie viele relevante Dokumente in der Menge aller gefundenen Dokumente existieren.

Product Owner

Der Product Owner ist dafür verantwortlich, den Wert des Produkts zu maximieren, das aus der Arbeit des Scrum-Teams entsteht. Die Art und Weise, wie dies geschieht, kann vom Unternehmen, den Scrum-Teams und Einzelpersonen sehr unterschiedlich sein.

Proof of Concept

Ein Proof of Concept (PoC) ist eine Meilenstein im Projektmanagement. Mit einem PoC soll gezeigt werden, dass ein Vorhaben prinzipiell machbar ist. In der Softwareentwicklung ist ein PoC häufig eine Anwendung, welche die gewünschte Funktionalität prototypisch implementiert.

Rank

Der Rank ist die Position eines Dokuments in der Ergebnisliste. Dabei ist irrelevant auf welcher Seite das Dokument angezeigt wird, die Zählung des Ranks beginnt immer auf der ersten Seite mit dem ersten Dokument.

Recall

Recall ist eine Metrik, welche ausdrückt, wie viele Dokumente aus der Menge der relevanten Dokumente gefunden wurden.

Relevance Engineer

Ein Relevance Engineer entwickelt eine intelligente Suchfunktion, welche die Anforderungen der Nutzer und der Organisation versteht[TB16, S. 2f]

Relevance Quality Management System (RQMS)

Ein Relevance Quality Management System ist eine Anwendung, welche Mittel zur Verfügung stellt, um die Relevanz der Suchergebnisse in einer Suchfunktion zu verbessern.

Scrum

Scrum ist ein einfaches Rahmenwerk, das Menschen, Teams und Organisationen dabei hilft, durch adaptive Lösungen für komplexe Probleme Wert zu schaffen[SS20, S. 5].

Security Assertion Markup Language (SAML)

Die Security Assertion Markup Language (SAML) ermöglicht den Austausch von Authentifizierungs- und Autorisierungsinformationen. Das SAML-Framework besteht aus Profilen, aus dem SAML-Protokoll, aus SAML-Assertions und aus SAML-Bindings. Das Framework stellt Funktionen bereit, um sicherheitsbezogene Informationen zu beschreiben und zu übertragen. Die wichtigsten Anwendungsfälle für SAML sind Single Sign-on, verteilte Transaktionen und Autorisierungsdienste. Alle Anwendungsfälle sind auf die Nutzung in Webservices optimiert.

Spring Bean

Beans sind in Spring Objekte, welche vom Spring IoC-Container (Inversion of Control Container) verwaltet werden. Eine Bean wird von einem Spring IoC-Container instanziiert, zusammengestellt und verwaltet. Ansonsten ist eine Bean einfach eines von vielen Objekten in ihrer Anwendung[NWW12b].

Stakeholder

Stakeholder sind Personen(-gruppen), die Interesse am Entwicklungsprozess bzw. dessen Ergebnissen haben bzw. dadurch in irgendeiner Weise berührt sind[aus Har09a, S. 26].

TypeScript

Die Skriptsprache TypeScript wurde von Microsoft entwickelt, ist kompatibel zu ECMAScript und führt komplexere Sprachkonstrukte, wie Klassen, Vererbung, Module und anonyme Funktionen ein. Teile von TypeScript wurden auch in einen neuen ECMAScript Standard übernommen.

Use Case

Ein Use Case (Anwendungsfall) stellt ein Modell dar, das beschreibt, wie Benutzer eine Funktion eines interaktiven Systems zur Durchführung ihre Aufgabe(n) einsetzen[Har09a, S. 38].

7.3 Abbildungsverzeichnis

2.1	Search platform Architektur bei Zalando, aus „Berlin Buzzwords 2017“[WW20]	3
2.2	Learning to rank als Zentrum einer Relevanzgesteuerte Organisation, aus <i>Relevant search: with applications for Solr and Elasticsearch</i> , S. 277[TB16]	7
2.3	Acht Phasen Modell nach Kotter, aus <i>Leading change</i> , S. 37[Kot12]	11
3.1	Stakeholder Map für ein RQMS	18
3.2	Antje Bartz - Content Curator, aus Unsplash[Sol20]	23
3.3	Boris Reichert - Relevance Engineer, aus Unsplash[Pol18a]	24
3.4	Crista Gärtner - Software Engineer, aus Unsplash[Chr19]	25
3.5	Daniel Sturm - Product Owner, aus Unsplash[Dis19]	26
3.6	Edith Marquardt - Domain Expert, aus Unsplash[Lia20]	27

3.7	Florian Lemke - Manager, aus Unsplash[Sol19]	28
3.8	Screenshot eines Relevance Cases (bzw. Judgment List) in Quepid	48
3.9	Screenshot des Relevance Tuning Dialogs in App Search, aus <i>Elastic App search documentation</i> [Ela22]	50
4.1	Eine Übersicht der im Model-Driven Design verwendeten Begriffe, aus <i>Domain-driven Design - Tackling Complexity in the Heart of Software</i> , S. 65[Eva04]	55
4.2	Übersicht der Architekturschichten und Module des RQMS	64
4.3	Search Engine in der Infrastrukturschicht	66
4.4	Search Query in der Infrastrukturschicht	67
4.5	Search Result in der Infrastrukturschicht	67
4.6	Aggregation - Assertion Test in der Offline Domain	69
4.7	Aggregation - Relevance Case in der Offline Domain	70
4.8	Aggregation - Label Filter in der Offline Domain	71
4.9	Aggregation - Relevance Test Run in der Offline Domain	72
4.10	Entity - Test Run Schedule in der Offline Domain	73
4.11	Value Object - Metric und dazugehörige Implementierungen in der Offline Domain	75
4.12	Entity - A/B-Test in der Online Domain	77
4.13	Aggregation - Search Fact in der Online Domain	79
4.14	Service - Search Fact Consolidator in der Online Domain	79
4.15	Value Object - Time Interval in der Online Domain	80
4.16	Value Object - Metric Bucket in der Online Domain	80
4.17	Value Object - Metric mit Implementierungen in der Online Domain	81
4.18	Service - Analytics Service in der Online Domain	83
4.19	Entity - Plugin in der Integration Domain	84
4.20	Entity - Search Usage Data Source in der Integration Domain	85
4.21	Entity - Search Log Entry in der Integration Domain	86
4.22	Entity - Search Result Interaction in der Integration Domain	86
5.1	Gradle Module des RQMS in den entsprechenden Architekturschichten	91
5.2	Screenshot der Testlaufübersicht im RQMS mit geöffnetem Label Filter Editor	96
5.3	Screenshot der Relevance Cases-Übersicht im RQMS	97
5.4	Screenshot des Relevance Case Editors im RQMS	98
5.5	Screenshot des Assertion Test Editors im RQMS	99

7.4 Tabellenverzeichnis

3.1	Merkmale des Content Curator	19
3.2	Merkmale des Relevance Engineer	20
3.3	Merkmale des Software Engineers	20
3.4	Merkmale von Product Owners	21
3.5	Merkmale von Domain Experts	21

7.5 Literaturverzeichnis

- [22a] *Connected search*. Aug. 2022. URL: <https://lucidworks.com/products/connected-search/> (besucht am 04.09.2022).
- [22b] *Online Store*. Juni 2022. URL: <https://help.shopify.com/de/manual/online-store>.
- [22c] *Shopify Fulfillment Network*. Juni 2022. URL: <https://help.shopify.com/de/manual/shipping/shopify-fulfillment-network>.
- [Ado] Adobe. *Adobe Analytics für umfassendere Erkenntnisse*. URL: <https://business.adobe.com/de/products/analytics/adobe-analytics.html> (besucht am 13.07.2022).
- [Ale18] Alex. *OpenId Connect Server Mock*. Apr. 2018. URL: <https://github.com/Soluto/oidc-server-mock> (besucht am 26.10.2022).
- [Atl] Atlassian. *Jira tools für agiles projektmanagement*. URL: <https://www.atlassian.com/de/software/jira> (besucht am 13.07.2022).
- [Aut22] Prometheus Authors. *Overview: Prometheus*. 2022. URL: <https://prometheus.io/docs/introduction/overview/>.
- [BN] Leonard Brünings und Peter Niederwieser. *Spock Framework*. URL: <https://spockframework.org/spock/docs/2.3/index.html> (besucht am 11.10.2022).
- [Bru22] Daniel Brunner. *Technologieführer für statische Codeanalyse + Architekturprüfung*. Juni 2022. URL: <https://www.axivion.com/> (besucht am 12.07.2022).
- [Chr19] Christina. *Photo by Christina at Wocintechchat.com on unsplash*. Nov. 2019. URL: <https://unsplash.com/photos/QCgAK6uHZm8> (besucht am 26.10.2022).
- [CL06] Gordon V. Cormack und Thomas R. Lynam. „Statistical Precision of Information Retrieval Evaluation“. In: SIGIR '06 (2006), S. 533–540. DOI: 10.1145/1148170.1148262. URL: <https://doi.org/10.1145/1148170.1148262>.
- [Cla+08] Charles L.A. Clarke u. a. „Novelty and Diversity in Information Retrieval Evaluation“. In: SIGIR '08 (2008), S. 659–666. DOI: 10.1145/1390334.1390446. URL: <https://doi.org/10.1145/1390334.1390446>.
- [Coc98] Alistair Cockburn. „Basic use case template“. In: *Humans and Technology, Technical Report 96* (1998), S. 28.
- [Cod22] Codacy. *The fastest static analysis tool from setup to first analysis*. 2022. URL: <https://www.codacy.com/product> (besucht am 12.07.2022).
- [Col17] Louis Columbus. *McKinsey's state of Machine Learning and AI, 2017*. Juli 2017. URL: <https://www.forbes.com/sites/louiscolumbus/2017/07/09/mckinseys-state-of-machine-learning-and-ai-2017/>.

- [Con20a] OpenSource Connections. *Documentation*. 2020. URL: <https://quepid.com/docs/> (besucht am 16.08.2022).
- [Con20b] OpenSource Connections. *Why Quepid?* 2020. URL: <https://quepid.com/why-quepid/> (besucht am 16.08.2022).
- [Dis19] Austin Distel. *Photo by Austin Distel on unsplash*. Mai 2019. URL: <https://unsplash.com/photos/wD1LRb90eEo> (besucht am 26.10.2022).
- [Dro20] Oliver Drotbohm. *Domain-Driven Design and Spring*. Okt. 2020. URL: <http://static.odrotbohm.de/lectures/ddd-and-spring/#ddd.spring> (besucht am 02.11.2022).
- [EB17] Alaa Elhadba und Mikio Braun. „The modern architecture of search“. In: *Berlin Buzzwords 2017*. Zalando, Juni 2017. URL: <https://2019.berlinbuzzwords.de/17/session/modern-architecture-search.html> (besucht am 15.07.2022).
- [Ela22] Elastic. *Elastic App search documentation*. Aug. 2022. URL: <https://www.elastic.co/guide/en/app-search/current/> (besucht am 16.08.2022).
- [Eva04] Eric Evans. *Domain-driven Design - Tackling Complexity in the Heart of Software*. Boston: Addison-Wesley Professional, 2004. ISBN: 978-0-321-12521-7.
- [Fer+08] Eduardo B. Fernandez u. a. „The Secure Three-Tier Architecture Pattern“. In: *2008 International Conference on Complex, Intelligent and Software Intensive Systems*. 2008, S. 555–560. DOI: 10.1109/CISIS.2008.51.
- [GHJ97] Erich Gamma, Richard Helm und Ralph Johnson. *Design patterns elements of Reusable Object Oriented Software*. Prentice Hall, Juli 1997. ISBN: 9780201633610.
- [Gio+18] Fanny Barbara Giordano u. a. *The stakeholder map: A conversation tool for designing people-led public services*. Juni 2018. URL: <https://vbn.aau.dk/en/publications/the-stakeholder-map-a-conversation-tool-for-designing-people-led-> (besucht am 13.07.2022).
- [Goo17] Google. *Jib - Containerize your Gradle Java project*. Mai 2017. URL: <https://github.com/GoogleContainerTools/jib/tree/master/jib-gradle-plugin> (besucht am 26.10.2022).
- [Gui] Christopher Guindon. *Release - eclipse IDE: The Eclipse Foundation*. URL: <https://eclipseide.org/release/> (besucht am 13.07.2022).
- [Har09a] Prof. Dr. Gehard Hartmann. „Grundlagen des Systementwurfs“. Okt. 2009.
- [Har09b] Prof. Dr. Gehard Hartmann. „methoden und Vorgehensmodelle des Entwurfes von Benutzerschnittstellen interaktiver Systeme“. Nov. 2009.
- [Hos+19] Christine Hosey u. a. „Just give me what I want“. In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (2019). DOI: 10.1145/3290605.3300529.
- [Høy+12] Jan Høydahl u. a. *Plugin Framework 4 Java*. Okt. 2012. URL: <https://pf4j.org/doc/getting-started.html> (besucht am 26.10.2022).

- [HVA18] Bhole Rahul Hiranman, Chaptre Viresh M. und Karve Abhijeet C. „A Study of Apache Kafka in Big Data Stream Processing“. In: *2018 International Conference on Information , Communication, Engineering and Technology (ICICET)*. 2018, S. 1–3. DOI: 10.1109/ICICET.2018.8533771.
- [ISO15a] ISO. *Quality management systems – Fundamentals and vocabulary*. ISO 9000:2015. Geneva, Switzerland: International Organization for Standardization, 2015.
- [ISO15b] ISO. *Quality management systems — Requirements*. ISO 9001:2015. Geneva, Switzerland: International Organization for Standardization, 2015.
- [Jam+18] Pooyan Jamshidi u. a. „Microservices: The Journey So Far and Challenges Ahead“. In: *IEEE Software* 35.3 (2018), S. 24–35. DOI: 10.1109/MS.2018.2141039.
- [Jet] JetBrains. *Intellij idea: Die leistungsfähige und ergonomische Java-Ide von JetBrains*. URL: <https://www.jetbrains.com/de-de/idea/> (besucht am 13.07.2022).
- [Joh03] Rod Johnson. *Expert one-on-One J2EE design and development*. Wrox, 2003.
- [KD16] George Kalpakas und Pete Bacon Darwin. Sep. 2016. URL: <https://angular.io/guide/what-is-angular> (besucht am 11.10.2022).
- [Kel17] Hunter Kelly. *Real-time ranking with Apache Kafka’s streams Api*. Nov. 2017. URL: <https://engineering.zalando.com/posts/2017/11/real-time-ranking-kafka.html> (besucht am 26.10.2022).
- [Kin20] Paul King. „A History of the Groovy Programming Language“. In: *Proc. ACM Program. Lang.* 4.HOPL (Juni 2020). DOI: 10.1145/3386326. URL: <https://doi.org/10.1145/3386326>.
- [Kot12] John P. Kotter. *Leading change*. Harvard Business Review Press, 2012. ISBN: 9781422186435.
- [Kru11] Alan Krueger. *Gradle Linux Packaging Plugin*. Mai 2011. URL: <https://github.com/nebula-plugins/gradle-ospackage-plugin> (besucht am 26.10.2022).
- [KS10] Mike Keith und Merrick Schnicariol. „Object-Relational Mapping“. In: *Pro JPA 2: Mastering the Java™ Persistence API*. Berkeley, CA: Apress, 2010, S. 69–106. ISBN: 978-1-4302-1957-6. DOI: 10.1007/978-1-4302-1957-6_4. URL: https://doi.org/10.1007/978-1-4302-1957-6_4.
- [KS22] Jennifer Kutz und Danny Sullivan. *Our search liaison on 25 years of keeping up with search*. März 2022. URL: <https://blog.google/products/search/danny-25-years-of-search/> (besucht am 25.10.2022).
- [Kun20] Andrian Kunz. *Angular Gradle Plugin*. Feb. 2020. URL: <https://github.com/Clashsoft/Angular-Gradle> (besucht am 26.10.2022).
- [Lab] Grafana Labs. *Grafana: The Open Observability Platform*. URL: <https://grafana.com/> (besucht am 13.07.2022).

- [Lai20] Joel Laity. *Libc++'s implementation of std::string*. Jan. 2020. URL: <https://joellaity.com/2020/01/31/string.html> (besucht am 07.10.2022).
- [LD19] Sudarshan Lamkhede und Sudeep Das. *Challenges in Search on Streaming Services: Netflix Case Study*. 2019. arXiv: 1903.04638 [cs.IR]. URL: <https://arxiv.org/abs/1903.04638>.
- [Lia20] Samia Liamani. *Photo by Samia Liamani on unsplash*. Aug. 2020. URL: <https://unsplash.com/photos/jLc6cUhVjJA> (besucht am 26.10.2022).
- [Ltd22a] Red Gate Software Ltd. *Documentation - Flyway by Redgate*. 2022. URL: <https://flywaydb.org/documentation/> (besucht am 27.10.2022).
- [Ltd22b] Red Gate Software Ltd. *Flyway*. 2022. URL: <https://www.red-gate.com/products/flyway/> (besucht am 27.10.2022).
- [Luc] Lucidchart. *Stakeholder map example*. URL: <https://www.lucidchart.com/pages/templates/stakeholder-map-example> (besucht am 13.07.2022).
- [Max] Aaron Maxwell. *Application insights-übersicht - azure monitor*. URL: <https://docs.microsoft.com/de-de/azure/azure-monitor/app/app-insights-overview> (besucht am 13.07.2022).
- [Mic] Microsoft. *Microsoft Teams*. URL: <https://www.microsoft.com/de-de/microsoft-teams/group-chat-software> (besucht am 13.07.2022).
- [MK22] Marinus Martin und Friedrich Kühne. „Spotify: Test: Kosten: Unterstützte Geräte im überblick“. In: *Netzwelt* (Feb. 2022). URL: <https://www.netzwelt.de/spotify/testbericht.html> (besucht am 16.08.2022).
- [MWB07a] Adam Murdoch, Stefan Wolf und Rodrigo Bamboo. *Gradle multi-project build*. 2007. URL: https://docs.gradle.org/current/userguide/multi_project_builds.html (besucht am 26.10.2022).
- [MWB07b] Adam Murdoch, Stefan Wolf und Rodrigo Bamboo. *What is Gradle*. 2007. URL: https://docs.gradle.org/current/userguide/what_is_gradle.html (besucht am 26.10.2022).
- [NB07] Peter Niederwieser und Leonard Brünings. März 2007. URL: <https://spockframework.org/spock/docs/2.3/introduction.html> (besucht am 11.10.2022).
- [NWW12a] Stéphane Nicoll, Andy Wilkinson und Phil Webb. *Spring Boot*. Okt. 2012. URL: <https://spring.io/projects/spring-boot> (besucht am 26.10.2022).
- [NWW12b] Stéphane Nicoll, Andy Wilkinson und Phil Webb. *Spring Docs*. Okt. 2012. URL: <https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-introduction> (besucht am 26.10.2022).
- [Opt] Optimizely. *Entfalten Sie ihr digitales potenzial*. URL: <https://www.optimizely.com/de/> (besucht am 13.07.2022).
- [ora21] oracle. *JDK 17*. Sep. 2021. URL: <https://openjdk.org/projects/jdk/17/> (besucht am 11.10.2022).

- [Ora22] Oracle. *Lesson: Annotations*. 2022. URL: <https://docs.oracle.com/javase/tutorial/java/annotations/> (besucht am 12.11.2022).
- [Pol18a] Polargold. *Photo by Polargold on unsplash*. Dez. 2018. URL: <https://unsplash.com/photos/10wgD2ncpZ0> (besucht am 26.10.2022).
- [Pol18b] Axel Pols. *Scrum – König unter den Agilen Methoden*. Sep. 2018. URL: <https://www.bitkom-research.de/de/pressemitteilung/scrum-koenig-unter-den-agilen-methoden> (besucht am 25.10.2022).
- [Pro] Prometheus. *Prometheus - Monitoring System and Time Series Database*. URL: <https://prometheus.io/> (besucht am 13.07.2022).
- [Ree20] Alexander Reelsen. *Implementing a modern E-Commerce Search*. Juni 2020. URL: <https://spinscale.de/posts/2020-06-22-implementing-a-modern-ecommerce-search.html> (besucht am 15.07.2022).
- [RH06] Colin Robbins und Edward Hamilton. „Successfully deploying single sign-on (SSO) within an outsourced environment“. In: *Siemens Insight* (Aug. 2006). URL: [https://web.archive.org/web/20071013031507/http://www.insight.co.uk/files/whitepapers/Single%20Sign%20on%20\(White%20paper\).pdf](https://web.archive.org/web/20071013031507/http://www.insight.co.uk/files/whitepapers/Single%20Sign%20on%20(White%20paper).pdf) (besucht am 16.08.2022).
- [Sak+14] Nat Sakimura u. a. *OpenID connect Discovery 1.0*. Nov. 2014. URL: https://openid.net/specs/openid-connect-discovery-1_0.html.
- [Sch10] Isaac Z. Schlueter. *NPM*. Jan. 2010. URL: <https://www.npmjs.com/> (besucht am 26.10.2022).
- [Sch20] Simon Schneider. „Quality assurance for search results“. In: TH-Koeln, Okt. 2020.
- [Sen22] Sentry. *About sentry*. 2022. URL: <https://sentry.io/about/> (besucht am 12.07.2022).
- [Sla] Slack. *Slack*. URL: <https://slack.com/intl/de-de/> (besucht am 13.07.2022).
- [Slo21] Jodi Sloan. *Building Smarter Search Products: 3 Steps for Evaluating Search Algorithms*. Apr. 2021. URL: <https://shopify.engineering/evaluating-search-algorithms>.
- [Sol19] LinkedIn Sales Solutions. *Photo by linkedin sales solutions on unsplash*. Juni 2019. URL: https://unsplash.com/photos/pAtA8xe_iVM (besucht am 26.10.2022).
- [Sol20] LinkedIn Sales Solutions. *Photo by linkedin sales solutions on unsplash*. Mai 2020. URL: <https://unsplash.com/photos/sWxKwsgY57c> (besucht am 26.10.2022).
- [Son22] SonarSource. *Code quality and code security*. 2022. URL: <https://www.sonarqube.org/> (besucht am 12.07.2022).
- [SS20] Ken Schwaber und Jeff Sutherland. *The scrum guide*. Nov. 2020. URL: <https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf> (besucht am 13.07.2022).

- [Str03] James Strachan. *Groovy Integration Guide*. 2003. URL: <https://docs.groovy-lang.org/latest/html/documentation/guide-integrating.html> (besucht am 26.10.2022).
- [Sui14] Decebal Suiu. *PF4J - Spring Integration*. Juni 2014. URL: <https://github.com/pf4j/pf4j-spring> (besucht am 26.10.2022).
- [TB16] Doug Turnbull und John Berryman. *Relevant search: with applications for Solr and Elasticsearch*. Manning, 2016.
- [Tur22] Doug Turnbull. *Doug Turnbull's blog*. Juli 2022. URL: <https://softwaredoug.com/> (besucht am 16.08.2022).
- [Wri+22] Daniel Wrigley u. a. *Search relevance - solr, Elasticsearch, OpenSearch - Training and Consulting*. Aug. 2022. URL: <https://opensourceconnections.com/> (besucht am 16.08.2022).
- [WW20] Daniel Weinland und Maximilian Werk. „Berlin Buzzwords 2017“. In: *Neural Search in Practice*. Zalando, Juni 2020. URL: <https://2020.berlinbuzzwords.de/session/neural-search-practice-0>.



Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer oder des Verfassers selbst entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Köln, 24. Januar 2023

Ort, Datum

Rechtsverbindliche Unterschrift