hochschule hof

University of Applied Sciences

Studienarbeit

Praktikum Software Entwicklung

Nachrichten-Suchmaschine

Fakultät Informatik - Studiengang Informatik

bei Prof. Dr. Richard Göbel

DieFünfFragezeichen

Daniel Beckstein

Johannes Franz

August Kraft

Philipp Marek

Jan Mothes

Inhaltsverzeichnis

1. Aufgabenstellung
2. Spezifikation
    2.1 Einleitung
    2.2 Programm
        2.2.1 Client
            2.2.1.1 Suche
            2.2.1.2 Suchergebnisanzeige
            2.2.1.3 Verwendete Browserversionen
        2.2.2 Server
            2.2.2.1 Crawler
            2.2.2.2 Elasticsearch
            2.2.2.3 Java-Server
3. Entwurf
    3.1 Einleitung
    3.2 RSSCrawler
        3.2.1 RSSFeeds
        3.2.2 Artikel
        3.2.3 Subscription / Notifications
    3.3 ESServer
    3.4 ESFeeder
        3.4.1 Neue Dateien Indentifizieren und Vorbereiten
        3.4.2 Verbindung zum ESServer
        3.4.3 Doppelte Artikel Herausfiltern
        3.4.4 Artikel hinzufügen
        3.4.5 Technologie
        3.4.5 Diagramme
    3.5 CustomerWebApp
        3.5.1 CWAInterface
        3.5.2 URL-Parameter als Such-Serialisierung
        3.5.3 Server
        3.5.4 Client
            3.5.4.1 GUI: Suche
            3.5.4.2 GUI: Suchergebnisanzeige
4. Programmdokumentation
    4.1 Github Projekt
    4.2 Ordnerstruktur
    4.3 Starten der Anwendung
        4.3.1 Serverseite
        4.3.2 Clientseite
5. Validierung
6. Selbständigkeitserklärung

1. Aufgabenstellung

Für das Nachrichtenarchiv soll eine Suchmaschine entwickelt werden, mit deren Hilfe die folgenden Suchkriterien unterstützt werden:

- Volltextsuche
- Zeitraum
- Thema (Politik, Sport, etc.)

Dazu sollen alle neuen Nachrichten über den Subscription-Mechanismus des Nachrichtenarchivs in den Index der Suchmaschine übernommen werden. Da häufig Quellen gleiche Nachrichten anbieten, sollen ähnliche Nachrichten nicht in den Index eingefügt werden. Hierzu soll ein effizienter Ansatz implementiert werden, der den Suchindex nutzt.
Als Ergebnis einer Anfrage soll eine Liste mit den zehn besten gefundenen Artikeln angezeigt werden. Für jeden Artikel wird hier die Datenquelle (e.g. Spiegel), das Publikationsdatum sowie der Titel angezeigt. Danach kann der Anwender schrittweise weitere Suchergebnisse abfragen. Für jedes Suchergebnis hat der Anwender drei Möglichkeiten:

- Der Anwender kann den Artikel im Internet durch einen Klick auf den Titel abrufen.
- Mit einem Klick auf das Wort „Cache“ kann der extrahierte Text abgefragt werden.
- Mit einem Klick auf das Wort „Similar“ wird eine weitere Suche angestoßen, die ähnliche Texte wie das Ergebnis liefert.

2. Spezifikation

2.1 Einleitung

Mithilfe der Volltextsuche ist es möglich verschiedene, durch RSS-Feeds aggregierte Artikel nach den Eigenschaften: Thema, Quelle, Zeitraum und Stichwörter/Volltext zu durchsuchen. Als Ergebnis bekommt der Nutzer eine Ergebnisliste mit jeweils der Überschrift, dem Veröffentlichungsdatum und einer Kurzzusammenfassung des gefundenen Artikels. Die Artikel sind nach Relevanz geordnet. Es ist bei jedem Artikel möglich den Inhalt der Nachricht durch einen Klick auf das Wort “Cache” zu extrahieren. Es ist außerdem möglich eine weitere Suche nach ähnlichen Artikeln durch einen Klick auf das Wort “Similar” anzustoßen.

2.2 Programm

2.2.1 Client

2.2.1.1 Suche

Es gibt eine Suchleiste für Volltext-/Stichwortsuche (ähnlich Google-Suchmaschine).

Eine Suche kann auf mit folgenden Kriterien beschränkt werden (Filter):

- Quelle (optional, mehrere auswählbar)
- Thema (mehrere auswählbar)
- Veröffentlichungsdatum (Zeitraum von - bis)

2.2.1.2 Suchergebnisanzeige

Es werden die zehn besten Ergebnisse angezeigt, die nach Relevanz sortiert sind. Mit dem “mehr” Button werden die nächsten zehn Ergebnisse angezeigt (“endless scrolling”).
Die Elemente bestehen aus einem Link zum Originalartikel und zwei Buttons für weitere Funktionen (siehe unten).

Folgende Metainformation müssen direkt in der Liste mit angezeigt werden:

- Titel
- Publikationsdatum
- Quelle
- Thema

Folgende Metainformationen können in der Liste oder im Cache-Bereich angezeigt werden:

- Volltext
- Autor

Der Cache-Bereich ist entweder über einen Link-Button erreichbar oder aufklappbar und beinhaltet den Fließtext.

Use-Cases:

Nach Stichwort suchen

> Original-Link aufrufen

> Text aus Cache anzeigen

> Nach ähnlichen Texten suchen

2.2.1.3 Verwendete Browserversionen

- Firefox Version 45
- Chrome Version 49

2.2.2 Server

2.2.2.1 Crawler

Ein Crawler sammelt unabhängig von der Benutzeroberfläche Nachrichten über RSS-Feeds, die vom Elasticsearch weiterverarbeitet werden.

Der Crawler bekommt eine Liste aus RSS-Links, mit deren Hilfe er die Links der Artikel heraussucht. Der Crawler erstellt anhand dieser Links XML-Dateien mit dem Inhalt der Artikel. Die XML-Dateien können dann in Elasticsearch eingelesen und verarbeitet werden.

"*Da häufig Quellen gleiche Nachrichten anbieten, sollen ähnliche Nachrichten nicht in den Index eingefügt werden. Hierzu soll ein effizienter Ansatz implementiert werden, der den Suchindex nutzt*" (Prof. Dr. Richard Göbel) -> Anbindung an Elasticsearch

2.2.2.2 Elasticsearch

Es wird die aktuelle Elasticsearch-Bibliothek verwendet, um die Stichwortsuche und die dazugehörige Datenhaltung zu realisieren. Elasticsearch stellt einen Webserver mit Rest-API zur Vergügung, mit dem die Benutzeroberfläche kommunizieren kann, um Suchergebnisse zu erhalten. Zu ähnliche Suchergebnisse werden von Elasticsearch nicht angeboten.

Elasticsearch bietet außerdem eine Java-API an, über die Daten eingefügt werden.

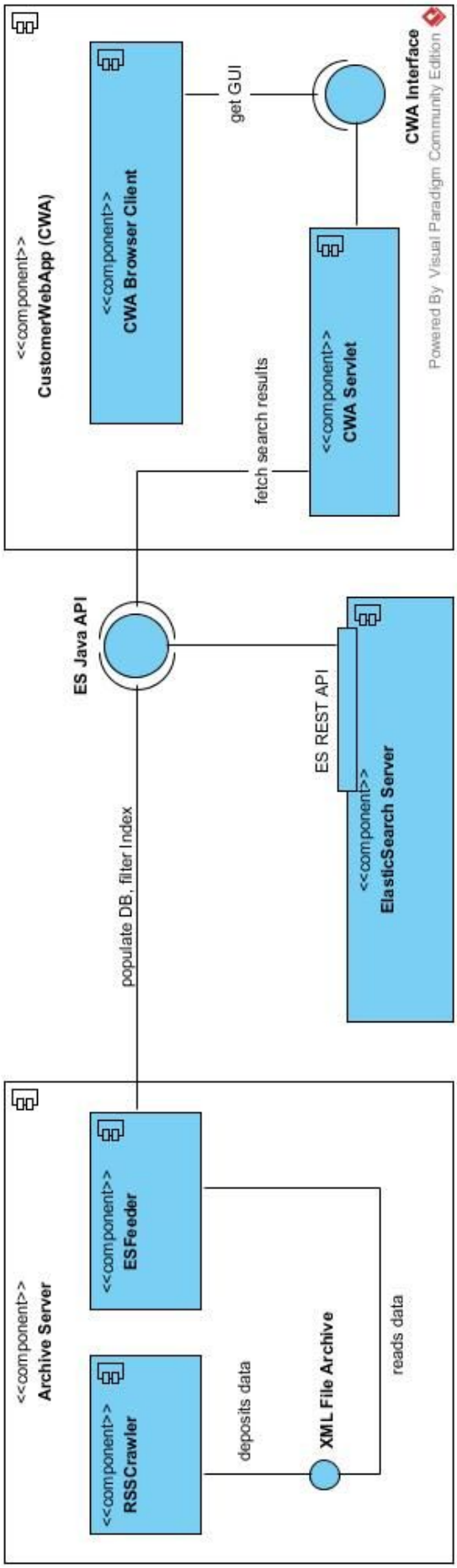
2.2.2.3 Java-Server

Die Webapplikation erfüllt die Servlet 3.1 Spezifikation und liefert die Benutzeroberfläche als Webseite aus. Die Applikation füllt außerdem die Elasticsearch-Datenbank mit den vom Crawler gefundenen Daten.

3. Entwurf

3.1 Einleitung

Die zu entwickelnde Software besteht aus 2 Applikationen (ESFeeder, CustomerWebApp), welche mit zwei weiteren vorhanden Applikationen interagieren muss (RSSCrawler, ESServer), um die Volltextsuche nach Artikeln zu ermöglichen.


<<component>>
CustomerWebApp (CWA)
<<component>>
CWA Browser Client
get GUI
CWA Interface
<<component>>
CWA Servlet
fetch search results
Powered By Visual Paradigm Community Edition
ES Java API
ES REST API
<<component>>
ElasticSearch Server
populate DB, filter Index
<<component>>
Archive Server
<<component>>
ESFeeder
<<component>>
RSSCrawler
deposits data
XML File Archive
reads data

3.2 RSSCrawler

Der Crawler wird benutzt, um mittels RSS-Links nach neuen Artikeln zu suchen, diese in einem Archiv als XML-Dateien zu speichern, und Nutzer der Artikel über neue Artikel zu informieren. Diese Anwendung ist vorhanden und muss nicht implementiert werden. Es werden lediglich die RSS-Feed-Links zum Testen des Systems ausgewählt.

3.2.1 RSSFeeds

Die Datei, in welcher die RSS-Feed-Links sowie die dazugehörige Sprache und das Topic definiert werden, ist folgendermaßen aufgebaut:

```
<sites xsi:schemaLocation=http://www.iisys.de/mmis/rssapp RSSFeeds.xsd
   xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
   xmlns="http://www.iisys.de/mmis/rssapp">
 <address language="de"
   link="http://www.faz.net/rss/aktuell/beruf-chance/"
   topic="career" ttl="2" country="Germany"/>
 <address language="de"
   link="http://www.faz.net/rss/aktuell/feuilleton/"
   topic="culture" ttl="2" country="Germany"/>
 <address language="de"
   link="http://www.faz.net/rss/aktuell/politik/"
   topic="politics" ttl="2" country="Germany"/>
</sites>
```

3.2.2 Artikel

Eine XML-Datei für einen einzelnen Artikel ist folgendermaßen aufgebaut (Ausschnitt):

```
<title>4000 Euro Zuschuss vom Staat: Ferngesteuerte Elektroautos in
Rekordzeit ausverkauft</title>
   <description>...</description>
   <link>http://www.der-postillon.com/2016/04/4000-euro-elektroauto.html
</link>
   <pubDate>Thu, 28 Apr 2016 06:13:00 +0200</pubDate>
   <guid isPermaLink="true">...</guid>
<author>Johannes Schlüter</author>
<ExtractedText>
      Berlin (dpo) - Dieses Angebot wollte sich niemand entgehen lassen:
      Nur Stunden nach dem Beschluss der Bundesregierung, den Kauf eines
      Elektroautos mit einer Prämie in Höhe von 4000 Euro zu unterstützen
      Vorräte an ferngesteuerten Autos aufgekauft.
</ExtractedText>
```

Die XML-Dateien werden in strukturierte Unterverzeichnisse gespeichert. Der Pfad zu solch einer Datei sieht dementsprechend so aus:

```
.../de/politics/Postillon/y2016/m4/d28/RSS-1700888097.xml
```

Relevant für uns sind folgende Informationen:

- Titel: Attribut “title”
- Veröffentlichungsdatum: Attribut “pubDate”
- Thema: zweite Unterordnerebene z.B. “politics”, Attribut “topic”
- Link: Domäne aus Attribut “link”
- Quelle: First- oder Second-Level-Domain des Links Attribut “source”
- Artikeltext: Attribut “ExtractedText”
- Autor: Attribut “author”

3.2.3 Subscription / Notifications

Der RSSCrawler verfügt über einen Subscription-Mechanismus, über den Nutzer des Archivs (in diesem Fall der ESFeeder) über neue Artikel informiert werden können (Notifications).
Dafür muss ein Pfad in einer Subscription-Konfigurationsdatei eingetragen werden. Auf diesem Pfad wird pro RSS-Link eine Notification-Datei angelegt, in der eine Liste zu den durch diesen Link gefundenen Artikel-Dateien angelegt wird.
In jeder Iteration des Crawlers werden komplett neue Dateien erstellt, einmal erstellte Dateien werden nicht mehr verändert.

3.3 ESServer

Der ESServer ist ein Elastic-Search-Cluster, welches unabhängig vom RSS-Crawler Artikel in einer Datenbank speichert. Diese Software ist vorhanden und muss ebenfalls nicht implementiert werden.
Der ESServer stellt zwei Indizes bereit: Den Suchindex, über den auf auf vorhandene Artikel zugegriffen werden kann und den Metadataindex, in dem Metadaten über die vorhandenen Artikel abgelegt werden.
Die Metadaten bestehen aus jeweils allen bisher in hinzugefügten Artikeln vorkommenden Thema-Feldern sowie Quelle-Feldern. Diese werden benötigt, damit im CWA (siehe Kapitel 3.5) nach diesen Feldern gefiltert werden kann. Dadurch wird verhindert, dass jedes mal alle Artikel im ESServer durchsucht werden müssen, um eine vollständige Liste dieser Felder zu erhalten.

3.4 ESFeeder

Der ESFeeder ist ein Java Kommandozeilenprogramm. Die Aufgaben des ESFeeders sind:

1. Neue XML-Dateien, die der Crawler erzeugt hat, für das Einfügen in den Suchindex des ESServers vorbereiten. Dies beinhaltet das Parsen der XML-Dateien, des Artikel-Pfades und die Konvertierung zu JSON.
2. Aufbauen einer Verbindung zum ESServer, Erstellung der Indizes falls noch nicht vorhanden.
3. Neue Artikel herausfiltern, wenn sehr ähnliche oder gleiche Artikel bereits im Suchindex vorhanden sind.
4. Neue Artikel in den Suchindex einfügen und Meta-Daten aus den neuen Dateien in den Metadataindex einfügen.

3.4.1 Neue Dateien Indentifizieren und Vorbereiten

Es wird der Subscription-Mechanismus des RSSCrawlers benutzt, um eine Liste aller neuen Artikel-Pfade zu erhalten. Da mehrere Notification-Dateien vorhanden sein können, müssen alle Pfade aus allen Dateien ausgelesen werden.
Anschließend werden die ausgelesenen Dateien gelöscht, um zu verhindern, dass bereits ausgelesene Artikel in einem späteren Durchlauf nochmal eingelesen werden.

Die neuen XML-Artikel-Dateien werden anhand der Pfade gefunden, geparst und zu JSON konvertiert. Die meisten der benötigten Informationen müssen aus dem Artikel-XML File extrahiert werden.
Das Thema jedes Artikels muss aus dem jeweiligen Dateipfad ausgelesen werden und die Quelle muss aus dem jeweiligen vollen Link extrahiert werden.

Aufbau eines Artikels siehe Kapitel 3.2.2. Artikel

Verantwortliche:

- jmothes: Auslesen & Löschen der Artikel-Pfade in den Notification-Dateien und die Bereistellung der Dateiinhalte
- dbeckstein, jfranz: Extrahieren von Informationen aus den Dateiinhalten (XML)
- jfranz: Extrahieren der nicht im XML enthaltenen Informationen aus dem Pfad und Formatieren von Informationen wenn benötigt (z.B. Datum)

- pmarek: Schreiben von JUnit-Tests für sämtliche Funktionalität

3.4.2 Verbindung zum ESServer

Verantwortlicher: akraft

Nachdem eine Verbindung zum ESServer hergestellt wurde, muss geprüft werden, ob die benötigten Indizes (Suchindex und Metadataindex) vorhanden sind sind und falls nicht, neu erstellt werden.

3.4.3 Doppelte Artikel Herausfiltern

Verantwortlicher: akraft

Es muss herausgefunden werden, ob im Suchindex bereits ähnliche Artikel vorhanden sind.
Dafür kann ein ElasticSearch query verwendet werden, der automatisch Artikel über einer festgeleten Ähnlichkeitsschwelle findet. Die Funktionsweise ist die gleiche wie die “Similar” Suche im CWA (siehe Kapitel 3.5 CustomerWebApp).
Die Ähnlichkeitsschwelle muss durch Ausprobieren festgelegt werden. Artikel, die z.B. zu großen Teilen den gleichen Artikeltext beinhalten, sollen herausgefiltert werden.

3.4.4 Artikel hinzufügen

Verantwortlicher: akraft

Bevor die Artikel letzendlich in den Suchindex eingefügt werden, müssen jeweils alle in den neuen Artikeln vorkommenden Themen-Felder sowie Quelle-Felder aufgelistet werden. Falls neue (im Metadataindex noch nicht vorhande) Felderinhalte vorhanden sind, müssen diese zu den vorhandenen hinzugefügt werden.
Anschließend werden die Artikel in den Suchindex eingefügt.

3.4.5 Technologie

Der ESFeeder soll als einfache Java-8 Applikation ohne User-Interface umgesetzt werden. Er sollte mit wenig Aufwand so geändert werden können, dass er periodisch ausführbar ist, auch wenn die Implementation nur durch manuelle Ausführung getestet wird.

Als IDE-unabhängiges Build-System wird Maven 2 eingesetzt.

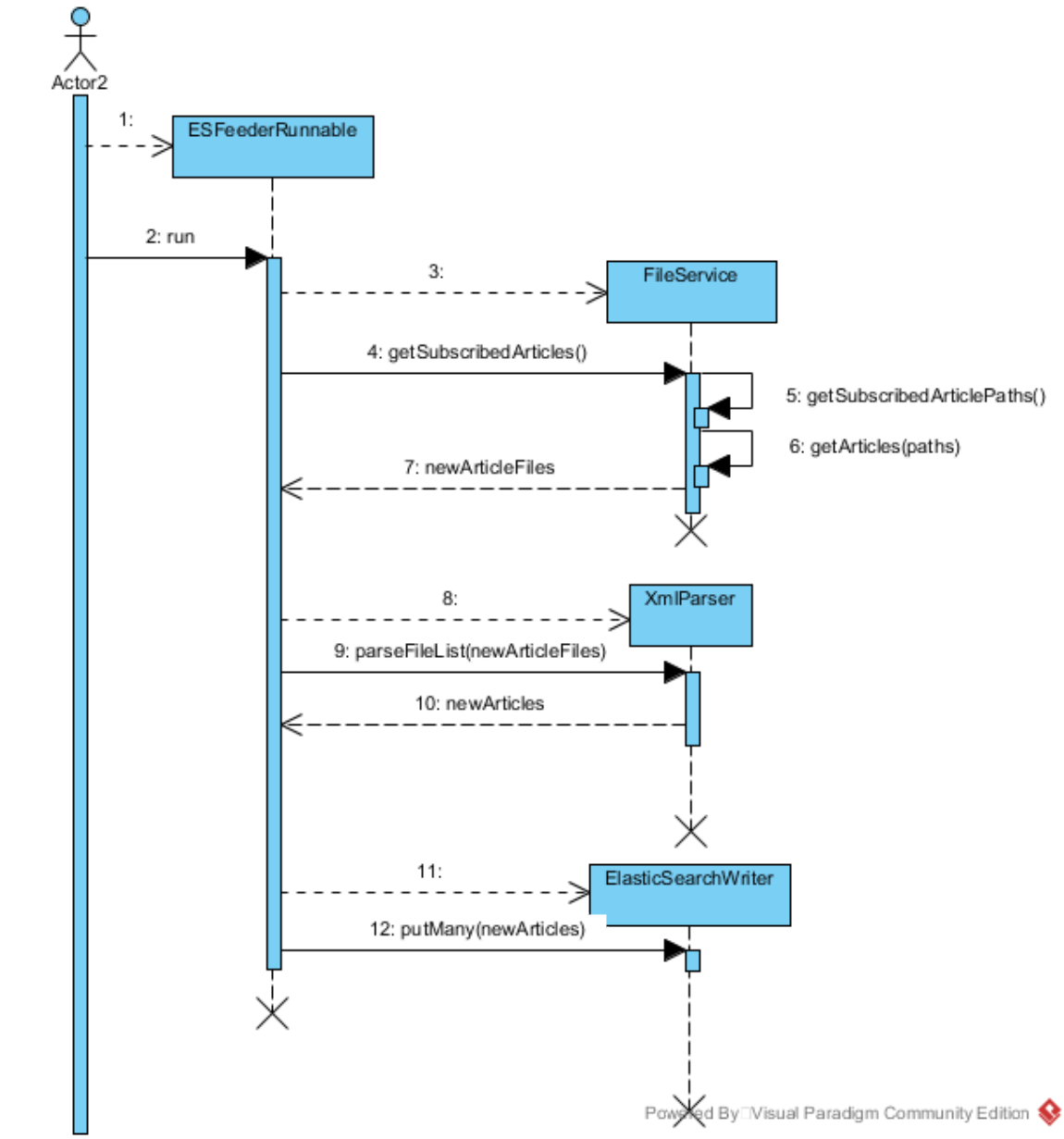
Verantwortlicher:

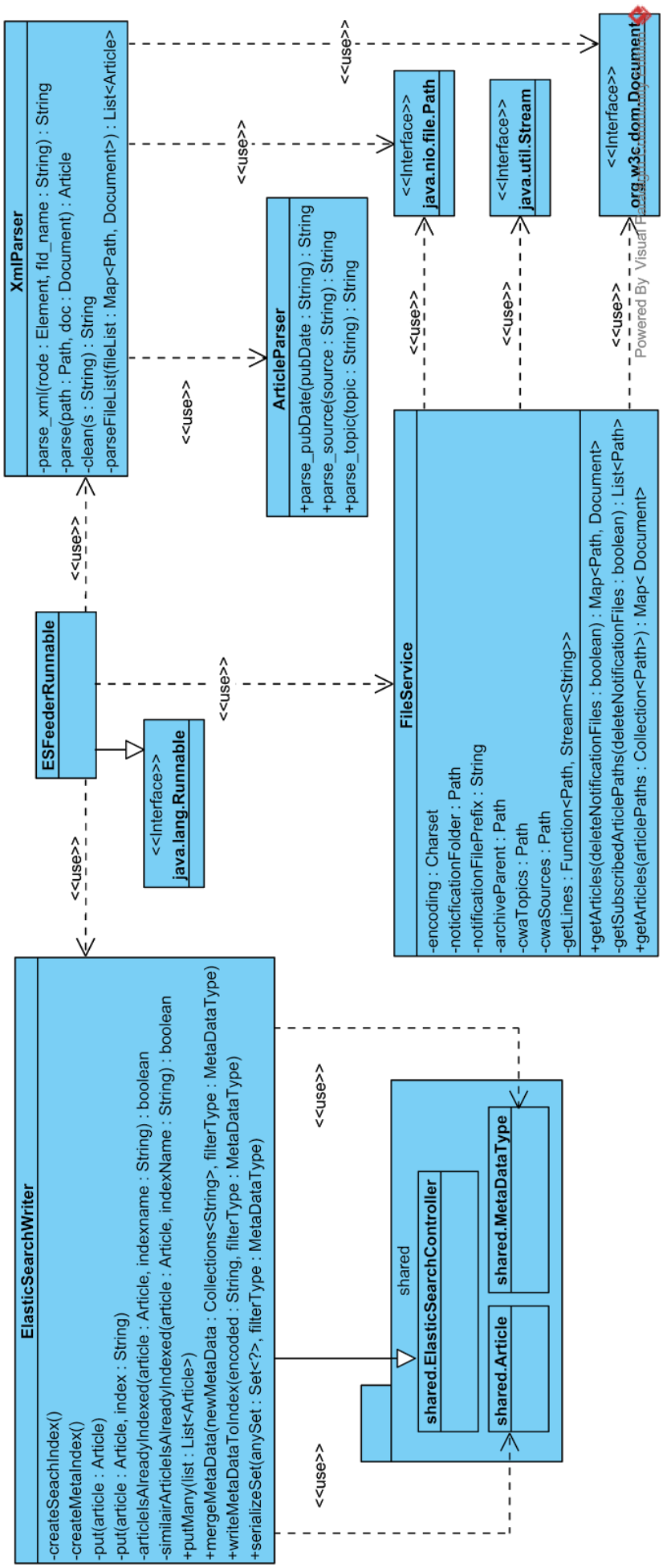
- jmothes: Konvertierung zu Maven Projekt
- pmarek: ESFeeder

3.4.5 Diagramme

Verantwortlicher:

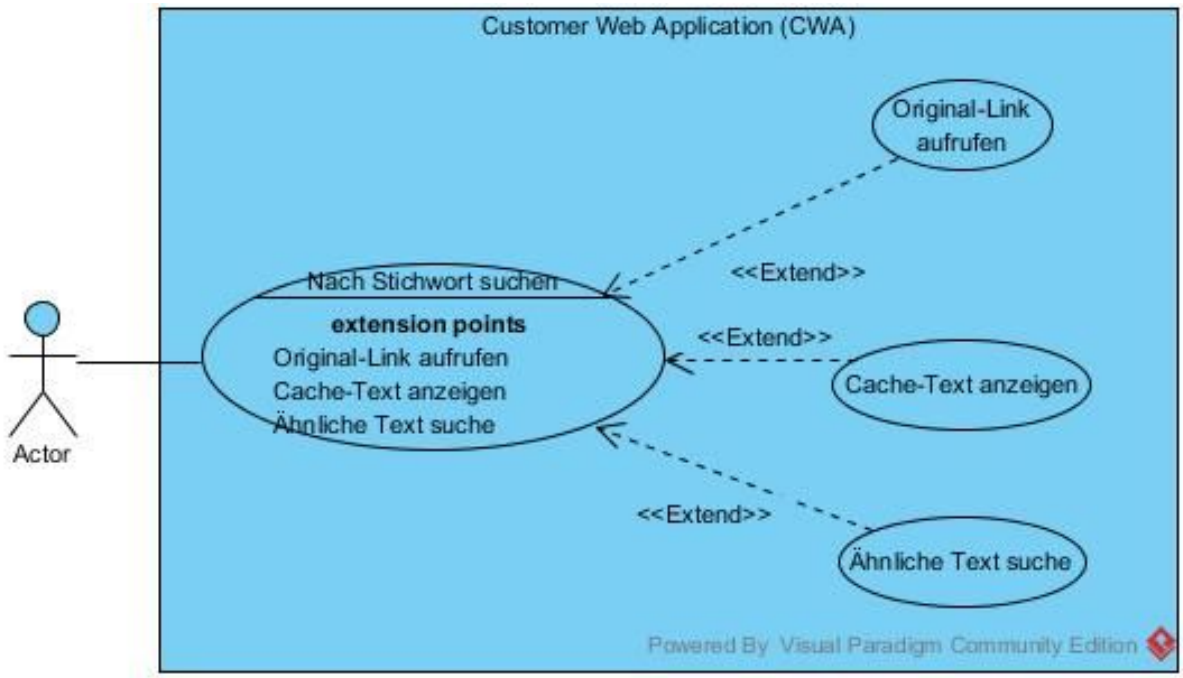
- pmarek: Erstellen der Diagramme




XmlParser
-parse_xml(rode : Element, fld_name : String) : String
-parse(path : Path, doc : Document) : Article
-clean(s : String) : String
-parseFileList(fileList : Map<Path, Document>) : List<Article>
ArticleParser
+parse_pubDate(pubDate : String) : String
+parse_source(source : String) : String
+parse_topic(topic : String) : String
<<Interface>>
java.nio.file.Path
<<Interface>>
java.util.Stream
<<Interface>>
org.w3c.dom.Document
ESFeederRunnable
<<Interface>>
java.lang.Runnable
FileService
-encoding : Charset
-noticficationFolder : Path
-notificationFilePrefix : String
-archiveParent : Path
-cwaTopics : Path
-cwaSources : Path
-getLines : Function<Path, Stream<String>>
+getArticles(deleteNotificationFiles : boolean) : Map<Path, Document>
-getSubscribedArticlePaths(deleteNotificationFiles : boolean) : List<Path>
+getArticles(articlePaths : Collection<Path>) : Map< Document>
ElasticSearchWriter
-createSeachIndex()
-createMetaIndex()
-put(article : Article)
-put(article : Article, index : String)
-articleIsAlreadyIndexed(article : Article, indexname : String) : boolean
-similairArticleIsAlreadyIndexed(article : Article, indexName : String) : boolean
+putMany(list : List<Article>)
+mergeMetaData(newMetaData : Collections<String>, filterType : MetaDataType)
+writeMetaDataToIndex(encoded : String, filterType : MetaDataType)
+serializeSet(anySet : Set<?>, filterType : MetaDataType)
shared
shared.ElasticSearchController
shared.MetaDataType
shared.Article
<<use>>
Powered By Visual Paradigm

3.5 CustomerWebApp

Bei der CustomerWebApp richten wir uns nach folgendem Use-Case-Diagramm:



Die CustomerWebApp (CWA) bietet eine GUI für Nutzer an, die die über den Suchindex des ESServers verfügbaren Artikel durchsuchen wollen. Die App besteht aus einem Client-Teil und einem Server-Teil.

Der Client-Teil ist eine Webseite und wird vom Server an den Browser des Clients ausgeliefert. Anschließend kommunizieren Client und Server über das CWAInterface, um dem Nutzer die jeweiligen Suchergebnisse zu liefern.

Der Server baut eine Verbindung zum Suchindex des ESServers auf, um Suchanfragen nach Artikeln ausführen zu können.

3.5.1 CWAInterface

Das CWAInterface besteht aus 3 HTTP-JSON-Anfragen:

1. Suchanfrage für normale Suche
2. Suchanfrage um ähnliche Artikel zu einem bereits angezeigten Artikel zu finden
3. Anfrage um die Metadaten abzufragen, nach denen der Benutzer die aktuellen Suchergebnisse filtern kann.

Die Suchanfragen sollen alle Parameter in der URL übergeben, um konsistent mit dem Seitenaufruf selber zu bleiben (siehe 3.5.2 Such-Serialisierung).

Die Suchergebnisse sollen in einer Antwort als JSON zuückgegeben werden, sodass die Ergebnisse vom Client dynamisch in die Ergebnisseite eingebaut werden können.

Exakte Spezifizierung der HTTP-Requests:

1.

GET AJAX Request "/getArticles/search"

- request parameter:
  - searchString (length restricted to 512 chars), topic filter, date filter (optional), source filter, search result range
- answer body:
  JSON with search result html (rendered on the server)
- onanswer:
  - update current page link to reflect search result range
  - append new results

URL-Beispiel:

```
/getArticles/search?query=helloworld&range=0-10
&topics=politics;economy
&sources=www.faz.net;www.spiegel.de&from=20-04-2010
```

2.

GET AJAX Request "/getArticles/similar"

- request parameter:
  - id of selected search result item, search result range
- answer body:
  JSON with search results or "no article for this id found" error
- onanswer:
  - update current page link to reflect similar search and result range
  - clear current result and append new results

URL-Beispiel:

`/getArticles/similar?id=1234&range=0-20`

3.

GET AJAX Request "/getMetadata"

- answer body:
  JSON with list of all existing topics and sources

Verantwortliche:

- jmothes: Codierung / Dekodierung (URL) und Validierung der Anfragen (Client -> Server)
- dbeckstein: Codierung / Dekodierung (JSON) und Validierung der Antworten (Server -> Client)

3.5.2 URL-Parameter als Such-Serialisierung

Die URL einer Suchanfrage enthält alle relevanten Parameter, die benötigt werden, um die Suche durchzuführen und entspricht damit einer serialisierten Form einer Suchanfrage.

Diese URLs werden nicht nur in den CWAInterface-AJAX-Anfragen verwendet, sondern werden auch an die URL der Seite selbst angehängt. Dies ermöglicht die volle Nutzung des Zurück-Buttons des Browsers sowie die Speicherung einer Suchanfrage als Browser-Lesezeichen zum späteren Abrufen der exakt gleichen Anfrage mit aktuelleren Ergebnissen.

Dafür muss der Client, wenn er eine neue Seite initialisiert, nur die URL der aktuellen Seite analysieren, um den Suchtyp herauszufinden und kann die URL-Parameter der eigenen Seite direkt verwenden, um eine CWA-Interface-Suchanfrage zu tätigen. In der Praxis muss lediglich "`/getArticles`" vor der aktuellen URL eingefügt werden, wenn diese nicht "leer" ist, um die CWA-Interface-Anfrage-URL zu erhalten.

Beispiele:

1. "/" (leer)
   Der Client initialisiert die Seite und führt keine Suchanfrage durch.

2. "/search?query=helloworld&range=0-10&topics=&sources="
   Der Client initialisiert die Seite und führt anschließend automatisch eine Suche mit folgender URL aus:
   "/getArticles/search?query=helloworld&range=0-10&topics=&sources="

Verantwortliche:

- jmothes: Controller-Klasse serverseitig
- dbeckstein: Funktionalität clientseitig

3.5.3 Server

Der Server übersetzt die Anfragen des Clients zu Anfragen an den ESServer. Dafür müssen die Parameter, welche in der Anfrage-URL enthalten sind, ausgelesen und validiert werden.

Anschließend muss eine Verbindung zum ESServer aufgebaut werden und die Parameter in eine ESServer-Anfrage eingebaut werden. Dabei muss beachtet werden:

Die normale Suchanfrage soll so konfiguriert werden, dass der Benutzer sich einen Tippfehler in seinem Suchtext erlauben kann (Fuzzy Suche).
Bei der Suche nach ähnlichen Artikeln muss sichergestellt werden, dass der zugehörige Artikel zur übergebenen Artikel-ID existiert.

Der Server wird mit Hilfe des Spring MVC Frameworks als Java-Servlet umgesetzt.
Vorausgesetzte Java Version: 8
Vorausgesetzte Servlet Spezifikations-Version: 3.0

Eingesetzter Technologien & Frameworks:

- Maven (Build Tool)
- Spring 4 MVC (MVC Framework)
- Thymeleaf (HTML Templating)

Verantwortliche:

- akraft: Verbindung zum ESServer, Konvertierung der Anfrage-Parameter zu ESServer Anfragen.
- jmothes: Erstellung des Maven-Projekts, Konfiguration von Spring & Thymeleaf

3.5.4 Client

Der Client besteht aus einer HTML-Seite mit einer Suchmaske und einem Bereich für die Anzeige der Suchergebnisse. Beim Initialisieren der Seite muss das CWA-Interface genutzt werden, um die vorhanden Metadaten abzufragen und diese in der Suchmaske als Filter anbieten zu können.

Eingesetzter Technologien & Frameworks:

- HTML5
- SASS (CSS extension, SCSS syntax)
- Typescript (Javascript extension)
- Node.js + NPM + Gulp (Build Tool)

Verantwortliche:

- dbeckstein, jfranz: Erstellen der HTML-Template, SCSS (Seitendesign) und Implementierung der sonstigen benötigten clientseitigen Funktionalität: Extrahieren von Usereingaben aus der Seite, Einfügen der Suchergebnisse in die Seite
- jmothes: Gulp-Buildscript für Transpilierung von Typescript zu Javascript & SASS zu CSS

3.5.4.1 GUI: Suche

Haupt-GUI-Elemente, die es dem Nutzer ermöglichen, nach einem Artikel zu suchen:

- Suchbegriff: Textfeld
- Die erweiterte Suche wird mit einem klick auf den Button “Filter” aufgeklappt:
    - Metadaten: Quelle: Multiselect (evtl. Toggle Buttons)
    - Metadaten: Thema: Searchselect & Multiselect
    - Zeitraum: zwei Datepicker (von, bis)

Der Nutzer kann nicht nach einem leeren Begriff suchen.

business

A third of gas stations in France running dry

http://rss.cnn.com/r/8d7efa044e/0Lmoney0Bcnn0

25.05.2016

Related: France may give workers right to ignore emails at home The strikes

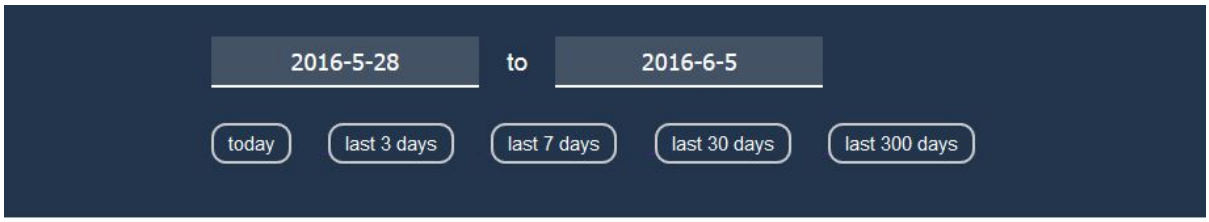
3.5.4.2 GUI: Suchergebnisanzeige

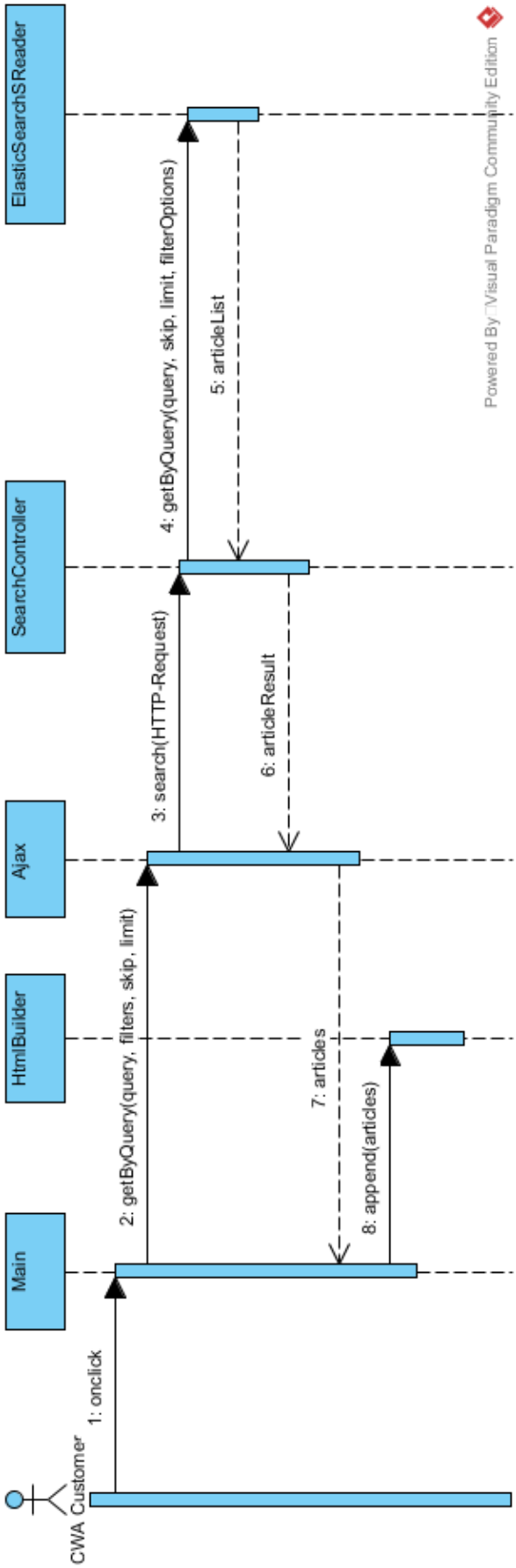
Interaktion mit der Ergebnisliste:

Es wird eine begrenzte Anzahl an Ergebnis-Elementen angezeigt, beim Click auf einen Button werden weitere Ergebnisse nachgeladen und an die vorhandene Liste angehängt. Der Button kann benutzt werden, bis keine weiteren Ergebnisse mehr vorhanden sind.

Interaktion mit einem einzelnen Ergebnis:

- Klick auf Titel bewirkt Weiterleitung auf den Link der Originalen Artikelquelle
- Klick auf Cache klappt den ArtikelText aus
- Klick auf "*Similar"* zeigt eine neue Ergebnisliste mit ähnlichen Artikeln




ElasticSearchSReader
SearchController
Ajax
HtmlBuilder
Main
CWA Customer
1: onclick
2: getByQuery(query, filters, skip, limit)
3: search(HTTP-Request)
4: getByQuery(query, skip, limit, filterOptions)
5: articleList
6: articleResult
7: articles
8: append(articles)
Powered By Visual Paradigm Community Edition

4. Programmdokumentation

4.1 Github Projekt

Das ganze Projekt steht unter der “GNU GENERAL PUBLIC LICENSE” für interessierte zum Download bereit. Dort befindet sich zusätzlich eine kurze Zusammenfassung und alle Dateien die zum Betrieb der Software notwendig sind:

https://github.com/Katharsas/PSE

4.2 Ordnerstruktur

doc/

RSSCrawler/

ESF_CWA_Shared/
- pom.xml - Maven build Konfigurationsdatei
- src/
  - main/
    - java/
      - shared/
        - Article.java - Article Klasse
        - ArticleId.java - ArticleId Klasse
        - ElasticSearchController.java - Schnittstelle zu ElasticSearch Server

ESFeeder/
- pom.xml - Maven build Konfigurationsdatei
- notifications/
  - Subscription Ordner: konfiguriert im RSSCrawler, enthält notification Dateien
- src/
  - main/
    - java/ - java source code
  - test/
    - java/ - java unit tests

CustomerWebApp/
- pom.xml - Maven Konfigurationsdatei
- doc/
  - Frontend Developer Guide - Explains how to build Javascript and CSS
- src/
  - main/
    - java/
      - java server code (compiled to WEB-INF/classes folder when deplyed as .war)
      - cwa/ - ApplicationConfiguration - Spring config
        - controller/ - Spring controller classes
        - service/ - Services used by the controllers
    - resources/
      - java server code resources (copied to WEB-INF/classes folder when deplyed as .war)
      - logback.xml - logger settings
    - scss/ - client source (siehe Frontend Developer Guide)
    - typescript/ - client source (sieheFrontend Developer Guide)
    - webapp/
      - WEB-INF/ - hidden server files
        - templates/
          - Thymeleaf HTML template source code (will be compiled to HTML dynamically when needed)
            - search.html - HTML Template der Website
        - web.xml - servlet configuration
      - resources/ - public server files
        - css/ - kompilierte scss files
        - js/ - kompilierte ts files
        - img/ - Bilder der Website
  - test/
    - java/ - java server unit tests

4.3 Starten der Anwendung

4.3.1 Serverseite

Zur Hilfestellung befindet sich eine `demo.txt` Datei im Hauptverzeichnis. Deren Inhalt ist hier ausführlicher dokumentiert:

Bevor der Server gestartet wird muss sicher gestellt sein, dass der Ordner `ESServer/data/elasticsearch/nodes/` leer ist. Den nodes Ordner kann man sich als Datenbank des ESServer vorstellen. Deshalb ist es ratsam bei einer neuen Installation frisch zu beginnen.

Nun kann der Server mit der Batch Datei `ESServer/bin/elasticsearch.bat` gestartet werden.

Stelle sicher, das keine leeren Notification Dateien im Ordner "`ESFeeder/notifications`" vorhanden sind. In den notification Dateien sind die Pfade von den neu gecrawlten XML Dateien gespeichert. Es muss sicher gestellt werden, dass diese Pfade existieren und diese XML Dateien vom Crawler in den Ordner `archive_dev` gespeichert wurden.

Im Ordner: `ESFeeder/`
Den Ordner `notification_presentation` zu `notification` umbenennen.

Im Ordner: `RSSCrawler/`
Den Ordner `archive_dev_presentation` zu `archive_dev` umbenennen.

Um unsere shared Klassen später zu verwenden, müssen diese zuerst compiliert werden:
Im Ordner: `ESShared/`

```
mvn clean install
```

Im Anschluss müssen nachfolgende Befehle in dieser Reihenfolge ausgeführt werden:

Im Ordner: `ESFeeder/`

```
mvn compile
```

Im Ordner: `ESFeeder/`

```
mvn exec:java
```

```
Die nachfolgende Funktion muss zwei mal ausgeführt werden um
sicherzustellen, dass der Index neu aufgebaut werden kann.
Im Ordner: ESFeeder/
mvn exec:java


Im Ordner: CustomerWebApp/
mvn tomcat7:run
```

4.3.2 Clientseite

Clientseitig lässt sich die Anwendung sehr einfach ausführen. Lediglich ein aktueller Browser ist notwendig. Wichtig ist nur, das die serverseitigen Dienste vorher erfolgreich gestartet wurden. Dazu wird folgender Link im Browser aufgerufen:

```
http://localhost:8080/CWA/
```

5. Validierung

Elasticsearch:

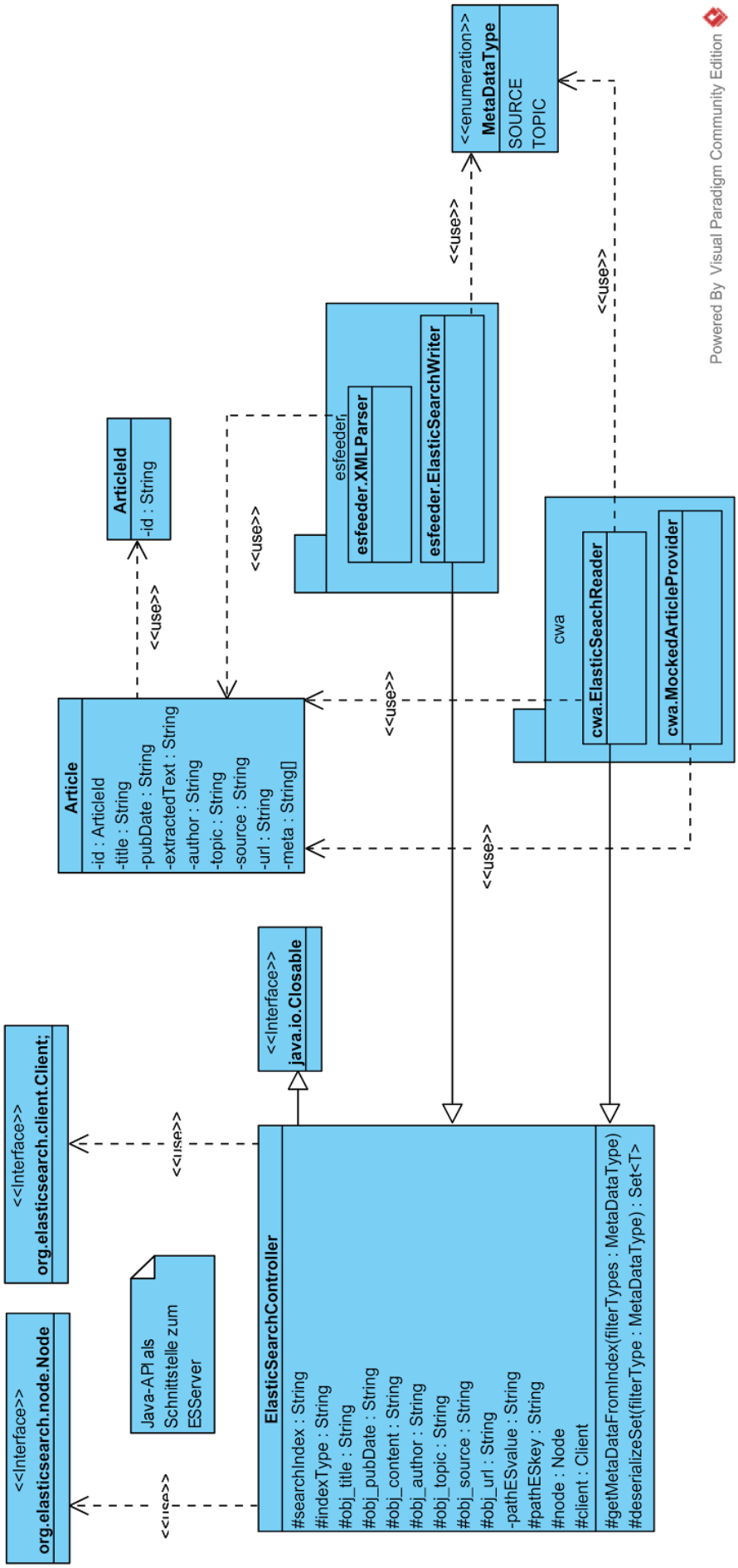
Da zum Start des Projekts unklar war, welche Funktionen ElasticSearch anbietet, wurde der ElasticSearch-JavaAPI-Code laufend angepasst und der ursprüngliche Entwurf für die betroffene Funktionalität wurde obsolet.
Die Java-Dokumentation von ElasticSearch erschwerte unsere Arbeit, weil diese in großen Teilen nicht oder nur mangelhaft existiert. Die meisten Informationen zur Funktionsweise der API mussten aus Beispiel-Code entnommen und durch eigenes Testen herausgefunden werden.

Ein Beispiel für nachträglich geänderte Funktionalität ist der Metadata-Index, welcher ursprünglich nicht vorgesehen war. Weiteres Beispiel ist das Entfernen doppelter Artikel. Für dieses Feature war ursprünglich eine eigene Klasse vorgesehen, da sich die Überprüfung jedoch viel leichter vor dem eigentlichen Hinzufügen umsetzen ließ, wurde diese in die put-Methode des ElasticSearchWriters integriert, welcher generell für das Hinzufügen von Daten zum ESServer verantwortlich ist.

Builds:

Sämtliche Java-Builds wurden mit Hilfe von Maven umgesetzt. Dies erwies sich als vorteilhaft, da somit das Kompilieren und Starten der Anwendung auch in unterschiedlichen IDEs (oder auch ohne IDE) sehr gut funktionierte. Ein Nachteil von Maven war die mangelnde Intuitivität des Build-Codes, was Dokumentation und Erklärungen nötig machte. Die jeweiligen Build-Befehle sind direkt im Build-Skript (`pom.xml`) des dazugehörigen Artefakts/Unterprojekts dokumentiert.

Ein besonders großer Vorteil, CWA mit Maven zu builden, bestand darin, einen embedded Tomcat Server zum Testen der Anwendung in den Build integrieren zu können, sodass vom Entwickler kein Server eingerichtet werden musste und auch die Deploy-Prozedur entfiel. Um CWA auf dem embedded Tomcat zu starten muss der Befehl "`mvn tomcat7:run`" ausgeführt werden (siehe `pom.xml`).


<<enumeration>>
MetaDataType
SOURCE
TOPIC
<<use>>
ArticleId
-id : String
esfeeder
esfeeder.XMLParser
esfeeder.ElasticSearchWriter
Article
-id : ArticleId
-title : String
-pubDate : String
-extractedText : String
-author : String
-topic : String
-source : String
-url : String
-meta : String[]
cwa
cwa.ElasticSeachReader
cwa.MockedArticleProvider
<<Interface>>
java.io.Closable
<<Interface>>
org.elasticsearch.client.Client;
<<Interface>>
org.elasticsearch.node.Node
Java-API als
Schnittstelle zum
ESServer
ElasticSearchController
#searchIndex : String
#indexType : String
#obj_title : String
#obj_pubDate : String
#obj_content : String
#obj_author : String
#obj_topic : String
#obj_source : String
#obj_url : String
-pathESvalue : String
#pathESkey : String
#node : Node
#client : Client
#getMetaDataFromIndex(filterTypes : MetaDataType)
#deserializeSet(filterType : MetaDataType) : Set<T>
Powered By Visual Paradigm Community Edition

Ursprünglich waren ESFeeder und CWA als Projekte ohne gemeinsam genutzten Code konzipiert worden. Da jedoch insbesondere Teile des ElasticSearch Java-API-Codes in beiden Projekten benötigt wurden, wurde das Shared-Artefakt eingeführt. Dieses beinhaltet Code, der sowohl vom ESFeeder als auch vom CWA benötigt werden, wodurch Code-Duplikation oder Kopieren von Code vermieden wurde.
Allerdings musste das Shared-Artefakt nach jeder Änderung in das lokale Maven-Repo installiert werden, damit die anderen Maven-Projekte die aktuelle Version nutzen konnten. Die Kompilierung und das installieren des Shared-Artefakts kann mit dem Befehl "`mvn install`" ausgeführt werden (siehe `pom.xml`).

Als Build-Tool für Typescript und SASS wurde Gulp aus dem Nodejs/NPM Ökosystem verwendet. Da Gulp nicht so verbreitet ist wie z.B. Maven, war die Integrierung der benötigten Transpiler etwas aufwendig. Die Compilezeit war mit unter 3 Sekunden im Vergleich zum Schreiben von nativem Javascript / CSS fast vernachlässigbar. Die Gulp-Builds sind in der Datei "`\CustomerWebApp\doc\DevGuide - Typescript.txt`" dokumentiert, die Build-Skript-Dateien heißen "`gulpfile.js`" (Build), "`package.json`" (Abhängigkeiten) & "`tsconfig.json`" (Typescript Transpiler Optionen).

6. Selbständigkeitserklärung

Hiermit erklären wir, dass wir die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt haben; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde nach unseren besten Kenntnissen bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

_____
Daniel Beckstein

_____
Johannes Franz

_____
August Kraft

_____
Philipp Marek

_____
Jan Mothes

Hof, den _____