ingenieur
wissenschaften
htw saar

Hochschule für
Technik und Wirtschaft
des Saarlandes
University of
Applied Sciences

Projektbericht

an der Hochschule für Technik und Wirtschaft des Saarlandes
im Studiengang Praktische Informatik
der Fakultät für Ingenieurwissenschaften

Persönliches Informationsportal „Elastifeed“ Backend

vorgelegt von
Jannik Schäfer
Matthias Riegler
Jorge Crespo Sueiro
Mark Martinussen

betreut und begutachtet von
Prof. Dr. Klaus Berberich

Saarbrücken, 20. September 2019

Inhaltsverzeichnis

1 Textentnahme - es-extractor

Mark Martinussen

Eine der wichtigsten Anforderungen an das Projekt war eine Funktion, die den relevanten Text aus einem beliebigen Artikel zu extrahiert. In diesem Abschnitt werden wir zuerst zwei formale Ansätze betrachten und dann im Anschluss sehen wie wir diese Funktion umgesetzt haben.

1.1 Problemstellung

Wir haben schnell festgestellt, dass es sinnvoll ist, das Extrahieren von Text nicht nur für Artikel im Internet zu betrachten, sondern für alle Websites. Zu diesem Schluss sind wir gekommen, weil das jede Website im Internet unterschiedlich aufgebaut ist und man kaum einen Artikel von einer anderen Website unterscheiden kann. Indem wir die Funktion auf beliebige Seiten des Internets ausweiten machen wir die Verwendung des Systems aus Nutzersicht einfacher, da dieser sich nicht damit befassen muss, ob eine Seite als „Artikel“ zählt. Weiterhin sparen wir uns aus Sicht der Entwickler und Administratoren die Notwendigkeit zu definieren welche Seiten erlaubte „Artikel“ sind. Insbesondere gibt es potenziell unendliche viele zulässige „Artikel“, alle einzutragen oder eine Regel dafür aufzustellen wäre vermutlich unmöglich. Im Folgenden soll Artikel gleichbedeutend mit einer beliebigen Website, die Text enthält, sein.
Wir definieren Text eines Artikels als relevant, wenn er einen inhaltlichen Nutzen hat. Dies sind zum Beispiel Überschriften, Unterüberschriften, inhaltlicher Text, Listen und ähnliches. Wir wollen vermeiden Teile wie Navigationselemente, Werbung, Kommentare und Verlinkungen zu extrahieren.
Mit dem Kontext unserer Architektur lässt sich die Aufgabe weiter konkretisieren. Die Aufgabe der Textentnahme ist es also, einen Link entgegenzunehmen und für diesen zu bestimmen, welche Teile dieser Website relevant sind. Die Architektur sollte zu der Microservice – orientierten Architektur passen, heißt, es sollte ein REST Interface geben und das Ergebnis muss so formatiert sein, dass es auch wieder gut von anderen Microservices verwendbar ist. Es dürfen also keine prozessspezifischen Adressen, Objekte oder Datenstrukturen zurückgegeben werden.

Soweit die Grundlagen – ein REST Interface zu implementieren ist relativ simpel und benötigt keine weitere theoretische Diskussion. Wir werden es ausschließlich im Abschnitt 1.4 über den schlussendlichen Aufbau betrachten. Viel wichtiger ist jedoch die Frage, wie wir nun bestimmen können welcher Teil einer Website relevant ist.

1.2 Theoretische Lösungsansätze

1.2.1 Ansatz 1: Druckansicht

Der erste Ansatz, den wir in Betracht gezogen haben, um den Inhalt eines Artikels zu extrahieren war die „Druckansicht“ der Seite zu verwenden. Hiermit ist diejenige Version des Artikels gemeint, welche erzeugt werden würde, wenn man mit einem Browser die Seite auf Papier ausdruckt. Wie uns aufgefallen ist, besitzen einige Seiten eine Druckansicht,

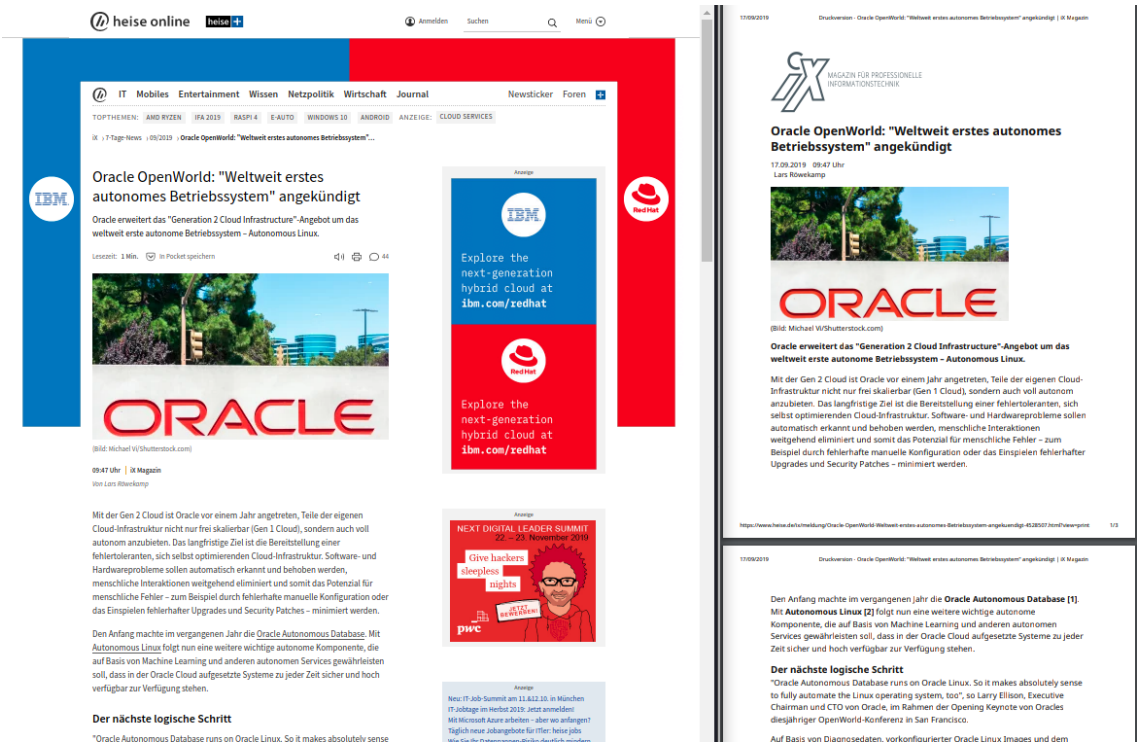

Abbildung 1.1: Ein Artikel und dessen reduzierte Druckversion

die sich von der dargestellten Version unterscheiden, oft indem Elemente wie Navigation, Bilder und Werbung entfernt sind und fast nur der Inhalt dargestellt wird. Beispielsweise die News – Seite `www.heise.de` besitzt für all ihre Artikel eine solche Druckversion. (siehe Abb. 1.1)
Man sieht wie die vollständige Website auf der linken Seite Navigationselemente und Werbung, die wir als nicht relevant ansehen enthält. Rechts sind dahingegen nur die relevanten Inhalte zu sehen. Falls wir also in der Lage sind, zu einem solchen Artikel vom Link auf diese Druckansicht zu schließen, könnten wir diese verwenden, um den relevanten Text einer Website zu bestimmen. Um diesen Lösungsansatz zu implementieren muss man mit zwei zentralen Fragen beschäftigen.

1. Wie erhält man die Druckansicht des Artikels, falls diese existiert?

2. Kann man bestimmen ob eine Website überhaupt eine reduzierte Druckansicht hat?

Ein Werkzeug zur Lösung der ersten Frage sind „headless“ Webbrowser. Der Begriff „headless“ beschreibt hierbei, dass ein Programm, welches normalerweise mit einer grafischen Benutzeroberfläche ausgeführt wird, komplett ohne diese rein algorithmisch gesteuert wird. Wir werden in Abschnitt **??** sehen, wie man einen headless Browser verwendet, da wir für das Rendern von Bildern die gleiche Herausforderung haben werden.

Aufgrund der zweiten Frage müssen wir leider feststellen, dass dieser Lösungsansatz nicht ausreichend ist. So gibt es keinen uns bekannten Weg, festzustellen ob ein Artikel eine reduzierte Druckansicht anbietet. Wie oben besprochen, ist jede Website unterschiedlich und es gibt keine fest definierte Schnittstelle. Weiterhin würde dieser Lösungsansatz eben auch nur bei Artikeln funktionieren, die eine solche Druckansicht anbieten, aber eben nicht, bei all jenen, die keine anbieten. Somit können wir unser Ziel mit diesem Ansatz nicht erreichen und müssen ihn leider verwerfen.

1.2.2 Ansatz 2: HTML DOM

Statt zu versuchen auf ein komplexes Feature, welches nur von bestimmten Seiten angeboten wird, zuzugreifen haben wir einen Ansatz entwickelt, welcher lediglich die HTML Datei einer Website benötigt. Eine jede Website wird durch eine HTML Datei beschrieben, die man ganz einfach mit dem Link herunterladen kann. Diese Datei besitzt ein wohl definiertes Format. Die Idee dieses Ansatzes ist es nun, sich ebenjenes Format, und insbesondere die Meta – Informationen, die es enthält, zu nutzen zu machen, um den relevanten Text zu bestimmen.

Der Aufbau einer HTML Datei oder HTML Dokument, wird durch das Hypertext Markup Language Document Object Model (HTML DOM) beschrieben: Ein Dokument besteht aus mehreren Elementen, wobei ein Element mit den spitzen Klammern `<name>` eingeleitet und mit `</name>` terminiert wird. Im Sinne des HTML DOM sind diese Elemente Objekte. Das heißt, sie besitzen Eigenschaften (Attribute), Methoden und Ereignisse. [4] Elemente die geschachtelt definiert sind, bilden eine Vererbungsbeziehung, wobei das oberste Objekt immer `document` genannt wird. Ein HTML Dokument beschreibt einen Baum (Abbildung 1.2), mit einem `head` für Metainformationen und einem `body` für den Inhalt. Dieser Inhalt kann dann weiter in Absätze, Überschriften, usw. unterteilt sein.

Nun können wir die Eigenschaften dieser Objekte auszunutzen. Beispielsweise gibt es das Attribut `document.body.textContent`, das allen angezeigten Text des Artikels enthält. Dieses beinhaltet jedoch auch den Text der Navigationselemente und Werbung. Folglich ist dieses Attribut alleine nicht ausreichend.

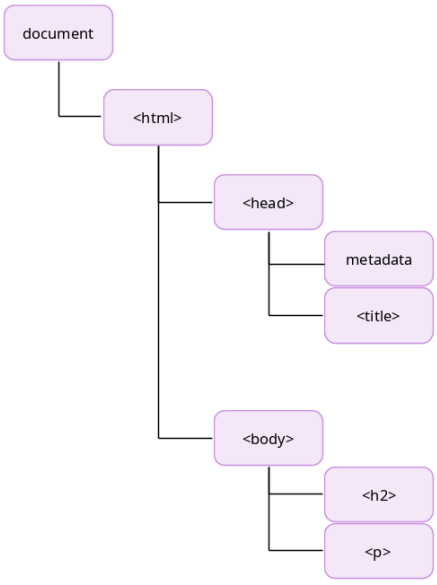

Abbildung 1.2: Struktur eines HTML Dokumentes

CSS Query Selectors

Zusätzlich zu seinen Attributen kann ein Element auch mit einer Klasse und einer ID dekoriert werden. Diese sind nicht direkt Teil von HTML, aber stattdessen Teil des CSS Stylings. So kann ein Webentwickler bestimmen, dass der Absatz, der den Haupttext enthält, eine bestimmte ID hat oder Teil einer Klasse ist. IDs, Klassen und HTML Elemente können dann mit sogenannten CSS Query Selectors abgefragt werden. Diesen Umstand können wir uns zunutze machen und selber diese Selektoren ausführen, um an die gewünschten Elemente zu kommen. Ein Auszug aus der Syntax Definition:

- `element element2` Wählt alle Elemente aus, deren Vorfahre ein HTML Element „element“ ist, und die selber ein Element vom Typ „element2“ sind.
- `element,element2` Wählt Elemente aus, die entweder element oder element2 sind.
- `.class` Wählt alle Elemente aus, deren Klasse „class“ ist.
- `#id` Wählt alle Elemente aus, deren ID gleich „id“ ist.
- `[attribut]` Wählt alle Elemente aus, die ein solche Atrribut besitzen.

Nun sind wir in der Lage, diese Query Selectors anzuwenden um den relevanten Text eines Artikels zu bestimmen. Beispielsweise können wir mit der Zeile

```
document.body.querySelectorAll(
        '.a-article-header__title,.a-article-header__lead,.article-content')
```

die relevanten HTML ELemente eines beliebigen Artikels von `www.heise.de` extrahieren. Die Formatierung bleibt hierbei erhalten, heißt wir können sogar Überschriften, Zitate, Aufzählungen und Tabellen unterscheiden und später unverändert anzeigen. Um den reinen relevanten Text zu erhalten würden wir von jedem Listenelement das `textContent` Attribut verwenden.

1.3 Mercury Web Parser

Wie wir in Abschnitt 1.2.2 besprochen haben, können wir mithilfe des HTML DOM und CSS Query Selectors zuverlässig den relevanten Text eines Artikels bestimmen. Damit wir nicht Selektoren für alle Websites der Welt definieren müssen, verwenden wir eine Library, die uns viel Arbeit abnimmt. Der Mercury Web Parser von Postlight Labs LLC[3] verwendet genau eine Menge von komplexen CSS Query Selectors und Eigenschaften des HTML DOM um für einen beliebigen Artikel den relevanten Text und mehr zu bestimmen. Der Mercury Parser kann mit nur einer URL oder zusätzlich dem schon heruntergeladenen HTML Dokument aufgerufen werden. In der Praxis verwenden rufen wir ihn nur mit der URL auf. Zusätzlich definieren wir den HTTP Header der Anfrage und mit welcher Formatierung (keine, HTML, Markdown) der relevante Text extrahiert werden soll.

```
1  import Mercury from '@postlight/mercury-parser'
2
3  const useragent = 'Googlebot/2.1 (+http://www.google.com/bot.html)';
4  async function runMercuryParser(url) {
5      data = await Mercury.parse(url, {
6          headers : { 'User-Agent' : useragent },
7          contentType: 'html'
8      });
9      return data;
10 }
```

Listing 1.1: Beispielhafter Aufruf des Mercury Web Parsers

Das Ergebnis ist ein JavaScript Object Notation (JSON) Objekt, die enthaltenen Felder sind in Tabelle 1.1 aufgelistet. Falls der Parser den Wert für ein Feld nicht bestimmen kann, ist er `null`.

1.3.1 Benutzerdefinierte Selektoren

Trotz der fortlaufenden Arbeit, die an dem Mercury Web Parser gemacht wird, haben wir feststellen müssen, dass er bei weitem nicht perfekt ist. Insbesondere, da er von einem englischsprachigen Team entwickelt wird, werden deutschsprachige Websites und Artikel oft nur unzureichend analysiert. Eine HTML Datei mit deutschen IDs und Klassen wie `.mitte` oder `#inhalt` ist für den Parser unverständlich. Glücklicherweise sind die Entwickler sich dieses Problems bewusst und bieten eine Lösung an.
Es gibt die Möglichkeit, für eine Domain die unzureichend erkannt wird, benutzerdefinierte

Feld	Beschreibung
`title`	Titel oder Überschrift der Website.
`content`	(Formatierter) Text, welcher als relevant angesehen wird. Mögliche Formate sind HTML, Markdown oder keins.
`author`	Autor oder Editor der Seite / Artikels.
`date_published`	Zeitpunkt zu dem die Website veröffentlicht wurde.
`lead_image_url`	URL des Vorschaubildes.
`dek`	Abstrakte Zusammenfassung des Artikels, die unter dem Titel steht.
`excerpt`	Ein kleiner Ausschnitt des Artikels, oft die ersten paar Sätze oder identisch mit dem `dek`.
`total_pages`	Anzahl der Seiten (HTML Dokumente) über die sich dieser Artikel erstreckt.
`next_page_url`	URL der nächsten Seite (HTML Dokument) des Artikels, falls der Artikel aus mehreren Seiten besteht.
`rendered_pages`	Anzahl der Seite (HTML Dokumente), die betrachtet wurden.
`url`	URL der Website.
`domain`	Domain der Website.
`word_count`	Anzahl der Wörter im relevanten Text auf der Seite.
`direction`	Vermutete Leserichtung des Artikels. Meistens „ltr“ oder „rtl“.

Tabelle 1.1: Rückgabewerte des Mercury Web Parsers

```
export const GolemExtractor = {
    title: {
        selectors : ['article header h1'],
    },
    author: {
        selectors : ['article header .authors .authors__name'],
    },
    date_published: {
        selectors : ['article header .authors .authors__pubdate'],
    },
    dek: {
        selectors : ['article header p'],
    },
    lead_image_url: {
        selectors : ['article header figure img'],
    },
    content: {
        selectors: ['article formatted'],

        // Clean out the div containers that will have ads.
        clean: ['div'],
    },
}
```

Listing 1.2: Definition eines benutzerdefinierten Selektors

Selektoren anzulegen. Intern heißen diese „Extraktoren“, wir werden diesen Begriff jedoch nicht verwenden, um Missverständnisse vorzubeugen. Bei diesen Selektoren handelt es sich ein JSON Dokument, bei dem jedem Feld eine Menge von CSS Query Selectors zugeordnet wird. [2] Beispielsweise ein benutzerdefinierter Selektor für die deutsche News – Seite `www.golem.de`.

1.4 Implementierung

Mit dem Mercury Web Parser sind wir in der Lage zuverlässig den relevanten Text eines Artikels zu extrahieren. Deshalb ist der es-extractor wenig mehr als ein Wrapper für den Mercury Web Parser, der dessen Funktionen als REST Interface zur Verfügung stellt. Der es-extractor ist komplett mittels JavaScript's `async/await` Syntax zur asynchronen Programmierung implementiert. Dies bedeutet, dass verschiedene Aufgaben gleichzeitig abgearbeitet werden können. Beispielsweise muss eine zweite Anfrage nicht darauf warten, dass die Seite der ersten Anfrage geladen ist, bevor sie selber anfangen kann, zu laden. In unseren Tests können 2 bis 3 Anfragen gleichzeitig bearbeitet werden, ohne dass es zu spürbaren Leistungsverlusten kommt.
Eine erfolgreiche Abarbeitung einer Anfrage resultiert in einer HTTP 1.1 Antwort mit dem Statuscode 200. `Content-Type` ist `application/json`, mit UTF-8 Zeichenkodierung. Das Datum ist ein JSON Objekt, welches dieselben Felder enthält wie die Antwort des Mercury Web Parser (Tabelle 1.1), mit Ausnahme des `content` Feldes. Das `content` Feld ist durch 2 Felder ersetzt worden:

Feld	Beschreibung
`raw_content`	Relevanter Text des Artikels, komplett ohne Textformatierung. Dieses Feld sollte genutzt werden um Suchanfragen und ähnliches zu starten.
`markdown_content`	Relevanter Text, der als Markdown [1] formatiert ist. Dieses Feld sollte verwendet werden, um den Text anzuzeigen, da er die ursprüngliche Form des Artikels beibehält. Er kann auch Links zu Bildern enthalten, die heruntergeladen werden müssen.

1.4.1 Schnittstellen

Bei Programmstart wird standardmäßig ein HTTP Server auf `http://localhost/`, Port 8080 gestartet. Dieser stellt 2 Endpunkte zu Verfügung, die beide nur auf einen HTTP 1.1 POST Request mit `Content-Type: application/json` reagieren.

/mercury/url Dieser Endpunkt erwartet ein JSON Objekt mit einem „url“ Feld. Die URL muss die URL der Website sein, von der der relevante Text bestimmt werden soll.

```
{"url" : "http://example.com"}
```

/mercury/html Dieser Endpunkt erwartet ein JSON Objekt mit einem „url“ Feld und einem „html“ Feld. Die URL muss die URL der Website sein, von der der relevante Text bestimmt werden soll und das HTML Feld muss dessen komplettes HTML enthalten.

```
{
 "url"  : "http://example.com" ,
 "html" : "<html><head><title>Example<\title>..."
}
```

1.4.2 Logging

Der es-extractor besitzt grundlegende Protokollierung, die zu `stdout` geschrieben werden. Der Logger basiert auf dem Winston Logger für NPM, man kann ihm einen neuen „Transport“ hinzuzufügen, um dafür zu sorgen, dass die Protokolle von außen erreichbar sind.
Die Ausgabe enthält immer zuerst das Log - Level (info, warn, error), den Tag und die Uhrzeit. Eine Ausgabe, die durch eine Anfrage an den Server erzeugt wurde, beinhaltet zusätzlich eine ID, die diese Anfrage eindeutig identifiziert. Schlussendlich kommt die Nachricht.

1.4.3 Fehlermeldungen

Falls es während der Laufzeit zu einem Fehler kommt, wird dieser Fehler zuerst mit einem entsprechenden Level protokolliert. Eine Anfrage die einen Fehler erzeugt, bekommt grundsätzlich immer eine HTTP 1.1 Antwort mit einem Statuscode der ungleich 200 ist. In Fällen wo der Fehler beispielsweise durch eine inkorrete URL oder HTML Dokument erzeugt wird, wird zusätzlich ein JSON Dokument mit einer Nachricht zurückgeschickt.

2 Rendern von Websites - es-scraper

Mark Martinussen

Literatur

[1] Sean Leonard. *The text/markdown Media Type*. RFC 7763. März 2016. DOI: 10.17487/RFC7763. URL: https://rfc-editor.org/rfc/rfc7763.txt.

[2] Postlight Labs LLC. *Custom Parsers README*. 2019. URL: https://github.com/postlight/mercury-parser/blob/master/src/extractors/custom/README.md (besucht am 19. 09. 2019).

[3] Postlight Labs LLC. *Mercury Web Parser*. 2019. URL: https://mercury.postlight.com/ (besucht am 19. 09. 2019).

[4] World Wide Web Consortium. *HTML Standard*. 2019. URL: https://html.spec.whatwg.org/ (besucht am 18. 09. 2019).

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

HTML DOM Hypertext Markup Language Document Object Model

JSON JavaScript Object Notation

Anhang