**Universität Duisburg-Essen**

Virtueller Weiterbildungsstudiengang Wirtschaftsinformatik (VAWi)

Projektarbeit

**Aktuelle Techniken und Herausforderungen des Web-Scrapings am Beispiel von online Kundenbewertungen**

Current techniques and challenges in web scraping using the example of customer ratings

Vorgelegt der Fakultät für Wirtschaftswissenschaften der Universität Duisburg-Essen

Verfasser: **Krug, Tobias**

Obertorstr. 2

65520 Bad Camberg

Matrikelnummer: 3077532

Erstgutachter: Prof. Dr. Tim Weitzel, Universität Bamberg

Zweitgutachter: #betreuender/e Hochschullehrer/in (Name und Uni)

Abgabe : 18.08.2020 / WiSe 2020/21

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

1. Einleitung

2. Definition / Einordnung Webscraping

2.1. Was ist Web Scraping (inkl. Scraping vs Crawling vs Browserautomation)

2.2. Use Cases / Anwendungsfälle

2.2.1. Suchmaschinen

2.2.2. Preis-/Produktvergleiche bzw. Marktbeobachtung

2.2.3. Daten Aggregation / Datensatz zusammenstellen

2.3. Schwierigkeitsgrade / technische Grenzen

2.4. Anwendungsgrenzen Grenzen des Web Scraping

2.5. Rechtliche Aspekte

2.5.1. Vertragsrecht

2.5.2. Urheberrecht

2.6. …?

3. Prozess Webscraping / Prozessmodell für gezielte Quellenextraktion

3.1. Informationsbedarf definieren

3.2. Analyse Zielseite(n)

3.2.1 Robots.txt

3.2.2 Nutzungsbedingungen / teilweise im Impressum

3.2.3 Sitemap.xml (hier richtig angeordnet ???)

3.3. Auswahl Vorgehensmodell (Scraping-Service vs Eigenentwicklung)

3.4. Entwicklung Scrapingtool

3.5. Datenevaluierung

4. Technik des Webscrapings (am Beispiel Scraping Kundenbewertungen mittels JS+Puppeteer+Cheerio)

4.1. Auswahl Technik / Infrastruktur

4.2. erforderliche Komponenten – Einrichtung Webscraping-Tool von Grund auf

4.3. Programmierungsschritte

4.3.1. HTTP-Requests

4.3.2. Möglichkeiten der Zieldatenidentifikation und -selektion

4.3.3. Datenweiterverarbeitung / Speicherung

4.4. exemplarische Detaillösungen für typische Herausforderungen im modernen Web

4.4.1. Captchas

4.4.2. dynamische Webseiten / dynamischer Seiteninhalt

4.4.3. Pagination (recursion vs link-Selektion)

4.4.4. Web-Formulare

4.4.5. IP-Blocking - Throttling

5. Zusammenfassung

6. Abbildungen

7. Literaturverzeichnis

8. Eidesstattliche Versicherung

# Abbildungsverzeichnis

# Tabellenverzeichnis

Tab. 1: 17

*# Das Tabellenverzeichnis ist optional. Löschen Sie dieses Verzeichnis, wenn keine Tabellen in Ihrer Arbeit vorhanden sind.*

# Abkürzungsverzeichnis

|  |  |  |
| --- | --- | --- |
| **Abkürzung** |  | **Bedeutung** |
| API |  | Application Programming Interface (deutsch etwa: „Programmierschnittstelle“) |
| CSS  HTML  REST |  | Cascading Style Sheets (Definitions-Textdateien zum optischen Erscheinigungsbild von Webseiten)  Hypertext Markup Language  REpresentational State Transfer |
| SOAP  W3C |  | Simple Object Access Protocol  kurz für: World Wide Web Consortium |
|  |  |  |
|  |  |  |

# Einleitung

*Kapitel noch komplett zu bearbeiten*

* Bedeutung / Verbreitung / Beispiele
* Statistik zu % des Webtraffic durch Bots – entsprechend Gegenmaßnahmen
* Web Scraping auf dem Rückzug auch wegen APIs wie REST und SOAP?
* eigene Anwendung kurz vorstellen

# Definition / Einordnung Webscraping

### Was ist Web Scraping (inkl. Scraping vs Crawling vs Browserautomation)

*Absatz noch im Stadium Sammlung Zitate / Quellen*

*Definitionselement: WS um Daten einzusammeln, die nicht leicht zugänglich in anderen Formaten vorliegen [Quelle: Jarmul, Lawson 2017 – Python web scraping]*

*Definitionselement: Das Speichern der „gescrapten“ Daten in gängigen Datenbankformaten wie JSON, XML aber auch Excel-Dateiformaten ist elementarer Bestandteil von Web Scraping, da es die Analyse und Weiterverwendung der Daten erst ermöglicht. [Quelle: Chapagain 2019 – Hands-on web scraping with Python]*

*Definitionselement: Unmittelbar im Zusammenhang mit Web Scraping muss auch der Begriff „Web Automation“ gesehen werden. Für einige Autoren sind Web Automation Leistungsmerkmale wie insbesondere das Ausfüllen von Formularen Bestandteil von Web Scraping und insbesondere ein Vorteil bzw. auch abgrenzendes Merkmal gegenüber Suchmaschinen [Quelle: Mitchell 2018 – Web scraping with Python*

*Kernelement von WS: „vorher feststehende Zielseiten“ – Unterschied zu IR / Crawling, wo URL/URI zu Beginn des Prozesses gerade nicht bekannt ist.*

* *Kann der Aspekt der feststehenden WS-Zielseiten ein Aspekt für künftige Scraper-Entwicklung sein? Wie ist der Stand der Technik bei der Frage, ob WS sich im ersten Schritt Zielseiten analog Suchmaschinen selbst suchen?*
* *Unterstützen das populäre WS-Frameworks wie Scrapy[nachlesen], Beautiful Soup[nachlesen], Puppeteer [NEIN]*
* *Achtung: Verknüpfung zum Semantic Web: Wenn URI maschinell gesucht wird, wie kann dann DOM analysiert werden, um die einzelnen Scraping-Zielelemente zu identifizieren?*

### Use Cases / Anwendungsfälle

Die Extraktion von Daten aus dem HTML-Dokument der Webseite *(das sollte sich mit dem wesentlichen Teil der Definition von WS aus 2.1. decken)*kann in mehreren Szenarien sinnvoll und dabei anderen Vorgehensweisen überlegen sein. Entsprechend finden sich folgende Anwendungsfälle von Web Scraping umgesetzt.

### Suchmaschinen

*kritisch! Als letztes Element der Gliederungsebene nehmen und kritisch diskutieren ob bzw. welche Ähnlichkeit zu WS.*

*Kernelemente von Suchmaschinen: Crawling (URL-erfassen) und vor allem auch Ranking (Webseiten im Verhältnis zum Suchbegriff)*

*dagegen WS: HTML-innerText in anderes Datenformat überführen*

* Crawling-Aspekt von Information Retrieval mit Web Scraping vergleichen - was ist gleich, was ist anders. Ist IR eine Art Vorläufer für Web Scraping?

### Preis-/Produktvergleiche bzw. Marktbeobachtung

*Insbesondere Tracking von Einzelinformationen wie eben dem Preis auf einer oder mehrerer, feststehender Webseiten im Zeitverlauf als Anwendungsfall. Entweder im Sinne eines „Preisalarms“ (kaufen unterhalb Schranke) oder zur Marktbeobachtung, z.B. auch im Sinne des Wettbewerber-Monitorings (Preisverläufe nach Saisonalität oder anderer Einflussfaktoren, wenn der Anbieter nicht von sich aus Preisstaffelungen aufzeigt und Preise ohne Historie verändert)*

*[Quelle mit Anwendungsfall: Geoff Boeing, Paul Waddell, New Insights into Rental Housing Markets…, 2017, Journal of Planning Education an Research ]*

### Daten Aggregation / Datensatz zusammenstellen

*wissenschaftlich wichtigster Anwendungsfall – unstrukturierte z.B. Foren- oder Social Media-Daten zu strukturierten Tabellen / Datenbanken transformieren oder eigene Strukturierung vs Struktur des Anbieters („alle Studi-Bewertungen zur Hochschule Fresenius in einer Liste“ vs. je Studiengang eine Seite oder ähnliches)*

*Literatur-Beispiele:*

*Craigslist [Quelle: …]*

### Schwierigkeitsgrade / technische Grenzen

Aufwand beim Crawlen abhängig von auf der / den Zielseiten eingesetzten Techniken

|  |  |
| --- | --- |
| Technik auf Seite | Aufwand / Anspruch |
| kein JS / direkte Links | einfach – reine HTTP-Requests zum Erhalt vollständiges HTML,  Links aus sitemap.xml oder über Link-Syntax-Nachahmung |
| JS-Rendering und Link-Erzeugung nach Regelwerk | schwieriger – reine HTTP-Requests liefern nicht mehr allen Seiteninhalt  Link folgen z.B. über Click-Event in headless browser |
| SPA, Captchas, Link-Erzeugung auf Basis (unbekannter) API | schwer bis nicht mehr automatisierbar auswertbar |

Tabelle : Schwierigkeitsgrade durch verschiedene Webseiten-Features

### Anwendungsgrenzen Grenzen des Web Scraping

Webscraping bietet zweifellos die (technische) Möglichkeit riesige Datenmengen aus dem Frontend unzähliger Webseiten zu gewinnen. Die gewonnen Daten werden aber regelmäßig gewisse Charakteristika aufweisen, die ihre weitere Nutzung mitbestimmen bzw. auch begrenzen.

Durch das Scrapen einer Seite gewinnt man die zu diesem Zeitpunkt dort bereitgestellten Daten bzw. sofern ausreichend Kontext gegeben ist auch Informationen. Diese Daten bzw. Informationen sind aber statisch, eine etwaige Veränderung im Zeitverlauf kann nur durch wiederholtes Scrapen und paralleles Data Warehousing inklusive Historisierung erreicht werden. Zu den gescrapten Daten wird man regelmäßig auch keine sonstigen Metadaten wie Datum der Onlinestellung, ggfs voreingestellte Ablaufdatum oder regionale Beschränkungen eines Items bei überregionalen Seiten gewinnen können. Man erhält also ein „Standbild“ einer Webressource bzw. Internetpräsenz.

Im Gegensatz zu z.B. käuflich zu erwerbenden Marktforschungsdaten erhält man durch Scraping auch keine Transaktions-/Prozessdaten. Die gescrapten Daten werden regelmäßig keine Angaben dazu liefern können wie oft ein Produkt in einem Shop gekauft wurde, wie oft eine Anzeige auf einem Online-Portal geklickt wurde oder ähnliches, sofern die Zielseite keine entsprechenden Counter im Frontend zeigt und deren Zählerstände ebenfalls extrahiert werden können.

Diesen Limitationen wird man in der Regel durch sich wiederholende Scrapingzugriffe auf die Zielseite begegnen und so z.B. den Use Case „Preisalarm“ (bzw. Artikel wieder/erstmals verfügbar etc.) realisieren. Der Änderungszeitpunkt selbst kann dabei zwar nicht erfasst werden, jedoch kann zumindest unter technischen Gesichtspunkten in fast beliebig kurzen Zeitintervallen das gewünschte Seitenelement der Zielseite neu abgefragt werden. Eine dem Use Case angepasste Reaktion auf einen veränderten Value des extrahierten HTML-Elements muss dann auf der Serverseite des Scrapers realisiert werden, wo der Abgleich des zuletzt extrahierten Wertes mit dem neuen Wert erfolgen muss.

z.B.

* keine historischen Daten / nur Snapshot einer Seite zum aktuellen Zeitpunkt – Historisierung ggfs auf Seite des Scrapers bzw. in nachgelagerter Datenbank
* keine Transaktionsdaten – wie oft etwas geklickt, gekauft, genutzt wird kann kaum ermittelt werden, es sei denn über Counter auf der Webseite
* zu viel Daten? Nicht im gezielten Einsatz zur Beantwortung einer / weniger Fragestellungen anhand handverlesener Quellen

### Rechtliche Aspekte

Zahlreiche Rechtstreite mit verschiedenen Ergebnissen zwischen Website-Betreibern und Firmen oder Einzelpersonen die Web Scraper eingesetzt hatten um Inhalte von diesen Webseiten zu extrahieren, belegen, dass keineswegs Klarheit herrscht bezüglich der Frage, was in Sachen Scraping erlaubt ist und was nicht. Dies gilt sowohl im internationalen Kontext, als auch national auf Deutschland bezogen. Eine eingehende Beschäftigung mit diesem Thema zu Beginn eines jeden Scraping-Projektes ist daher unerlässlich.

Die ersten Anlaufstellen in diesem Themenkomplex sollten dabei nach wie vor zwei altbekannte Dokumente des Zielwebseitenbetreibers sein. Die robots.txt (auch bekannt als robots exclusion standard oder robots exclusion protocol einerseits und die Terms of Use bzw. Nutzungsbedingungen der Webseite andererseits. Damit wird bereits deutlich, dass man hier zwei Web Scraping-Ansätze hinsichtlich ihres Scopes, ihrer Zielquellen unterscheiden muss.

Ein Web Scraper, der zunächst nach gewissen Mustern Links zu verschiedenen Webseiten bzw. verschiedenen (Top-Level-) Domänes sammelt („crawling“) um dann im zweiten Schritt von dieser Vielzahl an Zielen Daten zu extrahieren, wird entsprechend zahlreiche robots.txt und terms of use zu berücksichtigen haben. Populäre Scraping Bibliotheken wie Scrapy für Python bieten entsprechende Einstellmöglichkeiten ( ROBOTSTXT\_OBEY = True )

* *Ist die robots.txt im Prozess des scrapens machinenlesbar? Versuchen multi-link crawler „on the fly“ robots.txt und terms of use zu berücksichtigen”?*

*Textteil:*

*Manche, auch jüngere Quellen behaupten direkt und ohne Einschränkung, Web Scraping wäre legal. [Quelle: Mitchell 2018 – Web scraping with Python]*

### Vertragsrecht

Unterschied Scraping nach Log-In / Scraping öffentlich zugänglicher Seite

### Urheberrecht

kurz auf Werksschutz nach nach § 2 Abs. 1 Nr. 1 UrhG eingehen und dann ausführlicher auf Datenbankschutz für Datenbanken i.S.v. § 87 a Abs. 1 S. 1 UrhG und Rechte i.S.v. §87b

Amazon.de verbietet in seinen Nutzungsbedingungen unter Punkt 3 die Verwendung von Robots oder ähnlichen Programmen zum systemmaischen Extrahieren von Daten, auch wenn das nur einmalig geschehen soll.

Das amerikanische Kleinanzeigenportal Craigslist (Stand Mai 2020 unter den 50 meistbesuchten Website von us-amerikanischen IP-Adressen, Alexa.com) war in der Vergangenheit mehrfach Gegenstand von veröffentlichten Web Scraping – Anwendungen [mind. 2 Quellen], hat aber inzwischen in seinen Nutzungsbedingungen das scrapen jeglicher Inhalte der Seite verboten [craigslist.org als Quelle].

* + 1. Datenbankherstellerrecht („sui generis Datenbankschutz“)

ist ein Recht zum Schutz von Investitionen in Datenbanken das auf der EU-Richtlinie xxx basiert und in Deutschland zunächst im Informations- und Kommunikationsdienste-Gesetz (IuKDG) codiert wurde. Das IuKDG wurde 2007 aufgehoben und durch Nachfolgeregelungen, insbesondere im Telemediengesetz abgelöst. Enthalten waren allerdings auch Regelungen die das Urheberrechtsgesetz (UrhG) ergänzten. Im UrhG finden sich aktuell mit den Paragrafen 4 („Sammelwerke und Datenbanken“) und 87a, der den Datenbankbegriff definiert und dabei die erforderlichen Investitionen zur Erstellung der Datenbank herausstellt. Die Investition muss „wesentlich“ sein [Quelle – Kommentar wählen]. In der juristischen Diskussion wird deutlich, dass eine technisch, methodisch wenig strukturierte reine Anhäufung von Daten nicht schutzwürdig im Sinne des UrhG sein muss, wohlstrukturierte relationale Datenbanken wie sie i.d.R. im Backend von Webanwendungen existieren aber ggfs. entsprechend geschützt sind, selbst wenn das einzelne Datum in der Datenbank z.B. einen für jeden frei zugänglichen Ursprung hat [hier muss Quelle kommen]. Ohne das Thema hier weiter zu vertiefen, muss sich der Softwareentwickler oder Data Scientist der Webseiten scrapen möchte darüber im Klaren sein, dass er regelmäßig Datenbankinhalte extrahieren wird, deren Ansammlung, Strukturierung und Aufbereitung für die Bereitstellung in einem Frontend mit schützenswertem Aufwand für den Webseitenbetrieber verbunden sein kann.

*Eingehen auf Weiterverwendung? Schaden nur, wenn gescrapte Daten kommerziell weiterverwendet werden?*

### …?

*wahrscheinlich nicht benötigt / kein weiterer Detailaspekt mehr machbar vom Umfang her*

# Prozess Webscraping / Prozessmodell für gezielte Quellenextraktion

### Informationsbedarf definieren

Sofern Web Scraping das Ziel hat spezifische Daten einer oder mehrerer Webseiten zu extrahieren um diese Daten in eine neue, für das Scraping-Projekt sinnvolle (Datenbank-)Struktur zu bringen, extrahiert man beim Web Scraping von einer oder mehreren Webseiten konkrete HTML-Elemente und erstellt keine Kopien kompletter Seiten. Diese spezifischen Elemente können in der Struktur der Webseite, dem DOM (Document Object Model) eindeutig identifiziert werden. Es bieten sich verschiedene Strukturelemente an, um die entsprechenden Inhalte zu spezifizieren. Zu nennen sind hier CSS-Selectoren, XPath-Pfadangaben, HTML-IDs und -Klassen.

In der konkreten Besprechung des Datenbedarfs mit der anfragenden Fachseite bietet es sich beispielsweise an, gemeinsam die Webseite zu analysieren und direkt mit einem CSS-Selektor live im Browserfenster die zu extrahierenden Elemente zu bestimmen.

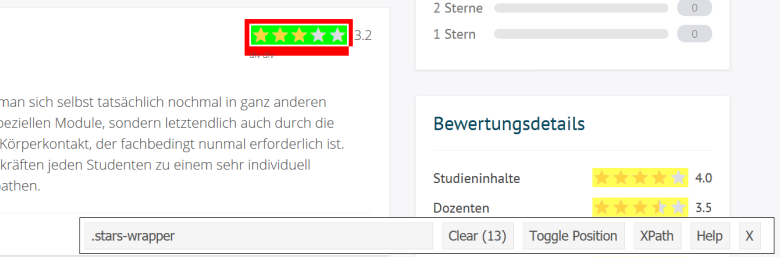


Abbildung 1: Selektion des Webseitendetails „.stars-wrapper“ mit Darstellung einer Studierendenbewertung eines Studiengangs auf [www.studycheck.de](http://www.studycheck.de)

Im Gegensatz zur Nutzung der Webentwicklertools der Browser wie dem „Inspector“ von Mozilla Firefox oder […***chrome-Tool benennen…]*** geht wesentlich weniger Bildschirmfläche verloren.

### Analyse Zielseite(n)

Neben dem Verständnis für die gewünschten Daten ist es wichtig zu verstehen was der Seitenbetreiber hinsichtlich automatisierter Datenextraktion erlaubt und welche technischen Merkmale der Zielseite das Scrapen beeinflussen können.

Hinsichtlich der Zulässigkeit sind die Robots.txt sowie die Nutzungsbedingungen bzw. der Code of Conduct der Seite zu beachten.

### Robots.txt

Ähnlich wie die sitemap.xml ist eine robots.txt keinesfalls für jede Webseite vorhanden, stellt aber einen stark verbreiteten Webstandard dar.

[Quelle: Chapagain 2019 – Hands-on web scraping with Python;

Quelle: https://developers.google.com/search/reference/robots\_txt?hl=de ]

Die robots.txt wird in der Regel auf root-Level, also in der URL direkt nach dem Host zur Verfügung gestellt (z.B. https://www.studycheck.de/robots.txt) und enthält Anweisungen für Crawler bzw. Bots welche Seiteninhalte erfasst werden dürfen und welche nicht. Da die robots.txt inzwischen eine große Rolle in der Search Engine Optimization („SEO“) spielt [Quelle: Drivas, Sakas et al. 2020 – Big Data Analytics for Search], werden nur ganz wenige kommerzielle Seiten keine entsprechende robots.txt aufweisen.

* *hier per Screenshot eine möglichst interessante robots.txt als Beispiel zeigen.*
* *Dann die verschiedenen Statements in je einem Satz erklären.*
* *Ausblick auf automatische Berücksichtigung durch Scraper? Muss eigentlich sein!*

#### Nutzungsbedingungen / teilweise im Impressum

#### 3.2.3 Sitemap.xml (hier richtig angeordnet ???)

Webseiten bieten häufig, aber nicht immer eine .html oder .xml Seite an, die alle bzw. alle für Suchmaschinen interessante Links / URL einer Domain enthält. Allerdings ist diese Seite in aller Regel nicht für den Aufruf im Browser durch einen menschlichen Besucher gedacht. Entsprechend ist die Seite in der Regel nicht sichtbar verlinkt und oft auch nicht offensichtlich mit „sitemap.xml“ bzw „.html“ benannt. Es lohnt den Versuch zu testen ob man die Sitemap leicht finden kann, da es sinnvoll sein kann diese Seite zu scrapen, die Links in Arrays zu speichern und dann für den iterativen Scrapingprozess der Unterseiten z.B. eines jeden Produktes oder sonstigen Elements einer Kategorie zu nutzen.

*Verweis auf Kapitel zu rechtlichen Fragen, wo gewissermaßen geklärt wird, was die Missachtung zur Folge haben kann, während hier nur beschrieben wird, was der Seitenbetreiber wünscht*

### Auswahl Vorgehensmodell (Scraping-Service vs Eigenentwicklung)

Bei der Frage wie die gewünschten Inhalte von der Zielseite extrahiert werden sollen, kann man neben der Anfrage beim Seitenbetreiber grundsätzlich 3 Herangehensweisen unterscheiden.

Erstens die Möglichkeit der Nutzung eines Webscraping-Dienstes wie bspw. ( BEISPIEL parsehub.com oder scrapehero.com) bei dem man ähnlich zum Vorgehen in der Datenbestimmung mit dem Bedarfsträger im eigenen Unternehmen in einer Weboberfläche die Zielobjekte markiert und den Rest dem Scraper des Dienstanbieters überlässt.

1. ***Einen Dienst ausprobieren und Grenzen / Probleme beschreiben***

Zweitens bieten einige Seiten APIs, also vordefinierte Schnittstellen zu ihren Seiten an.

1. ***(Twitter als Beispiel nehmen ???)***

Drittens: Eigenentwicklung unter Nutzung von „Tools“ wie Selenium oder Puppeteer

### Entwicklung Scrapingtool

***??? offen***

1. Grundsätze Softwareentwicklung ???
2. wichtiger Punkt, auch für den konkreten Fall: Deployment-Möglichkeiten wie PaaS (z.B. heroku) kurz skizzieren
3. Möglichkeiten der Automatisierung ansprechen

Ansonsten?

Da die Libraries bzw. Frameworks selbst relativ einfach anzuwenden sind, muss es hier wohl auch um Datenbankanbindung gehen. ABER WICHTIG: In dem Abschnitt geht’s nicht um technische Umsetzung!

### Datenevaluierung

1. Rückkopplungsschleife mit Fachseite zur Nutzbarkeit der Daten beschreiben
2. ggfs Erfassung weiterer Anforderungen wie z.B. timestamping beim Extrakt für Versionierung
3. findet sich in der Literatur ein Beispiel für extrem große Anwendungsfälle, in denen gescrapte Daten nicht mehr durch Menschen begutachtet werden können und automatisch weiterverarbeitet werden?

# Technik des Webscrapings (am Beispiel Scraping Kundenbewertungen mittels JS+Puppeteer+Cheerio)

Kurz Softwareentwicklungs-Modelle / Vorgehen ansprechen -> Waterfall vs. interativer Entwicklungsprozess

Waterfall: Wenn die Zielseite und der Informationsbedarf einfach und voraussichtlich zeitlich stabil in Einklang zu bringen sind, kann nach Waterfall-Modell vorgegangen werden, da keine größeren Rückkopplungsschleifen zur Fachseite erforderlich sein werden.

interativ / agil: Insbesondere bei komplexen Datenbeschaffungen von vielen und ggfs in Anzahl und Identität zunächst nicht (vollständig) bekannten Zielseiten wird es i.d.R. sinnvoll sein mit dem Bedarfsträger Rücksprache zu testweise gescrapten Daten zu halten und ggfs nachzujustieren.

### Auswahl Technik / Infrastruktur

Wenn man sich gegen die Nutzung eines kommerziellen Webscraping-Dienstes wie parsehub [parsehub.com], octaparse [octaparse.com] oder scrapehero [scrapehero.com] entschieden hat und eine individuelle Lösung erstellen möchte, stellt sich die Frage nach der zu verwendenden Programmiersprache, da es durchaus unterschiedliche Leistungsmerkmale gibt [hier Masterarbeit aus Schweden als Quelle]. Python in Kombinationen mit spezialisierten Frameworks ist eine aktuell häufig genannte Technologie. [Bücher, Artikel zitieren]. Allerdings bietet sich neben dem Einsatz einer höheren Programmiersprache wie Python auch der Einsatz von JavaScript und seinem Backend-Äquivalent Node.js an. JavaScript ist „die Sprache des Web“ [Quelle finden] und entsprechend gut geeignet um Webinhalte anzusprechen. JavaScript als Technologie kann dabei auch deshalb naheliegend sein, weil firmenintern die Mitarbeiter des Web-Dev-Teams die ersten Ansprechpartner für die Eigenentwicklung eines Scrapers sein könnten und diese Mitarbeiter immer mindestens mit JavaScript vertraut sein werden.

Allerdings wird regelmäßig der geplante Scope des Projektes Einfluss auf die Technologiewahl haben.

*Python mit Stärken im Handling von Big Data und Multithreading, JS eher für kleinere Projekte*

Ein Nachteil von JavaScript bzw. Node.js mit der Bilbiothek Puppeteer ist insbesondere der Umstand, dass Puppeteer immer einen Browser (Chrome) startet. Auch wenn dieser in der Standardeinstellung „headless“, also ohne User Interface läuft, verbraucht er im Vergleich zu browerlosen Technologien wie Selenium [Quelle Selenium Documantation] mehr Rechenleistung. Diesem Umstand kann allerdings leicht durch Nutzung von Cloudressourcen begegnet werden, sollte die eigene Hardware dem Workload nicht gewachsen sein oder nicht dafür blockiert werden.

[testweise Elastic Beanstalt mit Node-Programmcode befüllen?]

welche Sprache / welche Frameworks ?

Unterscheidung z.B. anhand von Multi- vs. Single-Threaded, also großes Projekt vs. kleines Projekt

### erforderliche Komponenten – Einrichtung Webscraping-Tool von Grund auf

* Installation node.js / ggfs Node Package Manager in 1-2 Sätzen beschreiben
* Linksammlung generieren -> Reg-Expression nutzen um alle Links eines gewissen Musters zu erhalten
* Require puppeteer und ggfs andere Libraries
* Instanziierung eines (headless) browsers und Navigation zu Zielseite(n)
* Zugriff auf HTML- (getElementById/Class), CSS- (querySelector/All)
* kurze Schilderung/Nennung verwendete Softwarekomponenten und deren Einrichtung

### Programmierungsschritte

Wichtigstes Programmierkonzept beim Einsatz von JavaScript auf Backend-Seite bzw. außerhalb seines ursprünglichen Einsatzzweckes in der Manipulation von User Interfaces von Webapplikationen ist die Asynchronität. JavaScript ist grundsätzlich „single threaded“, echte Parallelisierung durch gleichzeitiges, voneinander unabhängiges Abarbeiten von Aufgaben in verschiedenen Programmabläufen („Threads“) ist nicht möglich. JavaScript ist auch keine blockierende Sprache wie Java, die z.B. bei einem Aufruf einer Web-URL wartet bis das Ergebnis zurückgeliefert wird. JavaScript würde alles gleichzeitig bzw. sehr schnell von oben nach unten ausführen und Web- oder Datenbankaufrufe wären regelmäßig nicht rechtzeitig für Folgebefehle ausgeführt.

Für den Web Scraper im ANHANG XY wurde mit dem Konzept von Async – Await gearbeitet.

* ggfs kurz auf die anderen 2 häufig anzutreffenden Konzepte Callbacks und (rein) Promises eingehen

### HTTP-Requests

### Möglichkeiten der Zieldatenidentifikation und -selektion

Ein Webscraper soll spezifische Elemente einer Webseite, also eines HTML-Dokuments erfassen. Für die „Adressierung“ der Elemente bieten sich verschiedene Identifikatoren an. Direkt aus den Kerntechnologien des Web, HMTL und CSS stammen die Möglichkeiten HTML-Elemente z.B. über getElementByID oder getElementByClassName für HTML und querySelector und querySelectorAll für CSS anzusprechen. Die Verwendung der HTML-IDs und Klassen kann im <body> - Element der Webseite eingesehen werden, die CSS-Selektoren arbeiten nach genau dem gleichen Schema wie es auch im Styling, also in der CSS-Datei einer Webseite verwendet werden würde.

**let** pageHits **=** **document.**querySelector**(**'div .hits span'**).**innerText**;**

*Code-Beispiel x: Selektion über “querySelector()“, welcher das erste entsprechende Element zurück gibt*

**let** nbReviews **=**

Array**.**from**(document.**querySelectorAll**(**'span.report-count'**))**

**.**map**(**span **=>** span**.**innerText**);**

*Code-Beispiel x: Selektion über “querySelectorAll(), liefert eine Nodelist zurück, welche in ein Array transformiert mit den jeweiligen „innerText-Eigenschaften der Elemente gemappt werden kann*

Auch die XML Path Language (kurz: „XPATH“) kann zur Adressierung eines Elementes aus dem DOM verwendet werden.

**/html/body/**main**/**div**[**1**]/**section**[**1**]/**div**/**div**[**1**]/**div**[**3**]/**div**[**2**]/**div**/**div**/**div**[**3**]/**div**[**4**]/**div**[**2**]**

*Code-Beispiel eines XPATHs zu einem DOM-Element „tief“ in der HTML-Struktur des Body-Elements*

Allerdings ist der entsprechende CSS-Selektor oft kürzer und besser für Menschen lesbar.

div**.**card**-**row**:**nth**-**child**(**4**)** **>** div**:**nth**-**child**(**2**)**

*Code-Beispiel CSS-Selektor für das gleiche Element wie im vorausgegangenen XPATH-Beispiel (DOM-Elemente „Regelstudienzeit“ der URL: https://www.studycheck.de/studium/arztassistent/dhbw-1565)*

### Datenweiterverarbeitung / Speicherung

Insbesondere das Extrahieren von DOM-Elementen über die puppeteer-Methode querySelectorAll() liefert Elemente in Form einer Nodelist zurück, welche in ein Array überführt werden kann und es dann erlaubt Elemente (z.B. Bewertung 1, Bewertung 2,…) mit deren Eigenschaften wie dem .innerText (sofern Textinhalt und nicht z.B. Bilder) zu Wertpaaren zu verknüpfen. Diese Wertpaare sind dann in der richtigen Datenstruktur um in ein leicht verständlichen Tabellenschema mit mehreren Spalten gespeichert zu werden.

[hier muss Code-Beispiel für speichern als csv und speichern in MySQL-DB kommen]

### exemplarische Detaillösungen für typische Herausforderungen im modernen Web

### Captchas

* interessant aber fraglich wie sinnvoll bei Nutzung im Rahmen von WS: Captch-Solving Services wie 2captcha.com oder deathbycaptcha.com… ggfs beschreiben

### dynamische Webseiten / dynamischer Seiteninhalt

Moderne Webseiten nutzen immer mehr JavaScript um Inhalte dynamisch nachzuladen, also vom Server zur Verfügung zu stellen, während dem User erste Inhalte der aufgerufenen Seite schon angezeigt werden. Diese zuerst geladenen Inhalte sind das in der Regel wenige 100 KB große HTML-Gerüst der Seite in seiner CSS-Stylisierung. Die JavaScript-Funktionalitäten werden als letztes geladen [Gasston, Peter 2013, The Modern Web, Seite, Chapter 5.1 – online Buch von O’Reilly hat keine Seitenangabe ]

[auch Mitchell mit Hinweis auf Schwierigkeiten beim Scrapen wenn mit Tools wie Beautifoul Soup HTML abgegriffen werden soll, dass mit JS dynamisch verändert wird, konkret Redirect im Sinne von SPA, siehe Chapter 11]

[…]

http-Request Bibliotheken wie im Falle von JavaScript / Node.js „Axios“ oder Python-Bibliotheken wie „urllib“ oder „requests“ können nicht asynchron betrieben werden, sie können nicht ohne fest codierte Wartezeiten auf Serverantworten warten, wie das ein Browser kann.

[…]

Auch auf Ebene der Scraping-Bibliotheken wie Beautiful Soup oder Scrapy (Python) oder Cheerio (JavaScript) welche Methoden zur wunschgemäßen Selektion von HTML-Elementen anbieten, besteht die Herausforderung, dass z.B. noch nicht alle div-Container einer Klasse mit dem ersten HTTP-Request geladen wurden. Neue Techniken zur Ausbalancierung von möglichst kurzen Ladezeiten einer Seite und vielen, direkt verfügbaren Multimediainhalten (mit im Vergleich zu Text großen Datenvolumen) wie „lazy loading“ [https://developers.google.com/web/fundamentals/performance/lazy-loading-guidance/images-and-video/] verschärften die Problematik weiter, wenn aufgrund eines initialen Request schlicht nicht alle Elemente einer Seite geladen und zur Darstellung gebracht („rendering“) werden. Hier kommen sogenannte headless Browser wie sie in den Bibliotheken Selenium (für Python) oder Puppeteer (für Node.js) enthalten sind ins Spiel und hier bildet sich auch die Brücke zum Begriff der Browser Automation. Durch die Nutzung der vollen Browserfunktionalität kann das vollständige Laden bzw. Ausführen von JavaScript-Befehlen gesteuert werden.

Beispiel für Puppeteer-Befehle für Browser-Objekt bzw. Networkfunktionalität

### Pagination (recursion vs link-Selektion)

Viele Inhalte können auf Webseiten als Listen aufgeteilt auf mehrere Seiten verteilt angezeigt werden. Es entspricht guter Webdesignpraxis einzelne Seiten nicht zu lang werden zu lassen, also den User nicht dazu zu zwingen viele Bildschirmhöhen weit „runter“ zu scrollen um alle Listenelemente zu sehen. Außerdem bietet die Fortsetzung einer Liste von Items auf einer „Seite 2“ die Möglichkeit einen Fixpunkt für das erneute Laden von weiteren Elementen zu setzen. Die Beispielzielseite dieser Arbeit “studycheck.de” zeigt bspw. immer 4 studentische Bewertungen je Seite im Bewertungsteil eines Studiengangs an. Die Seite enthält jedes Mal 4 entsprechende div-Container mit der Klasse „card-report-…“, egal wie umfangreich die einzelne Bewertung ist.

ggfs Teilaspekt: Javascriptfunktion “window.location.asign()“-Method for redirecting to new URL… But what would be such a use case?

### Web-Formulare

Web-Inhalte können sich hinter Logins verbergen und es erforderlich machen, dass sich der Scraper zunächst anmeldet, bevor Zugang zu Daten gegeben ist. Ein Beispiel hierfür sind soziale Netzwerke wie Instagram oder Facebook, die den Zugriff auf Inhalte für nicht angemeldete Nutzer stark begrenzen und zur Anmeldung auffordern.

Durch die Möglichkeit der vollen Browserfunktionalität von headless Chrome in Puppeteer oder die ähnlichen Funktionen in Selenium ist es möglich User-Interaktionen wie Mouseover (triggern des DOM-Events „onfocus“) oder Tastaturbefehle (DOM-Events onkeydown, onkeypress, onkeyup) zu simulieren und entsprechend Buttos wie „Anmelden“ oder Submit-Buttons vom Loginformular „drücken“ zu lassen.

Hier muss ein Beispiel für das Ausfüllen kommen

### IP-Blocking - Throttling

Gewerbliche Webseitenbetreiber wie auch der Betreiber der Seite „studycheck.de“ könnten regelmäßig kein Interesse daran haben, dass Bots die nicht zu Werbeeinnahmen z.B. durch Suchmaschineneintragung beitragen, ihre Seiten mit einer hohen Anzahl von Requests belegen und damit Serverressourcen auslasten. Entsprechend können Web-Admins Regeln für das Blocken von besuchenden IP-Adressen insbesondere anhand von Requests pro Zeiteinheit festlegen, sogenanntes „rate limiting“ [Priebisch- O’Reilly Buch hinzugefügt am 26.06.]

Eine Möglichkeit IP-Blocking zu umgehen, liegt in der Nutzung von Proxyserver-Diensten die es erlauben in kurzen Zeitabständen ausgehenden Traffic über wechselnde IP-Adressen zur Zielseite weiterzuleiten. Die Zielseite kann dann ein Volumen von Zugriffen zumindest nicht mehr anhand der IP-Adresse identifizieren. Ein Beispiel für einen Open Source Dienst aus dieser Kategorie ist Privoxy (<http://www.privoxy.org/user-manual/index.html>). Da beim Einsatz von Proxyservern der ausgehende Netzwerktraffic beeinflusst wurde, ist innerhalb der Applikation nur die Anpassung der Dauer eines Scrapingdurchlaufs erforderlich, so dass innerhalb eines Durchlaufs IP-Neuzuweisungen wirksam werden.

Dieses Throttling ist auch abgesehen von IP-Rotationen sinnvoll, um idealerweise überhaupt nicht erst zur Belastung für den Zielserver zu werden.

ggfs eigene Teilüberschrift **„Tarnung Web Scraping“**

Teilaspekt von Verschleierung: Cookies verwerfen/löschen, Web Analysedienste wie Google Analytics täuschen [wie in „WS with Python“, Mitchell erwähnt]

# Zusammenfassung

Hier gemäß Titel der Arbeit einerseits

aktuelle Techniken resümieren

und andererseits:

Als zentrale Herausforderung in Sachen Web Scraping muss aus Sicht des Autors noch vor technischen Fragestellungen der rechtliche Rahmen genannt werden. Von Open Source - /Community-Webprojekten abgesehen, wird der Betreiber einer Webseite regelmäßig wenig Interesse daran haben, dass die von ihm bereitgestellten Inhalte abseits der verbreiteten Suchmaschinen gecrawlt und gar gescrapt werden. Zahlreiche Nutzungsbedingungen populärer Seiten machen entsprechende Verbote deutlich. In einschlägigen Tutorials zum Webscraping wird sich nach wie vor die Meinung vertreten oder zumindest dem Leser nahegelegt, dass ein zielseitenfreundliches, langsames Crawlen/Scrapen, ggfs. kombiniert mit IP-Rotation-Techniken von der Zielseite kaum erkannt oder jedenfalls nicht nachverfolgt werden kann.

*+ im deutschen Rechtsraum ggfs auch gar nicht strafbar, wie Artikel aus „Kommunikation&Recht“.*

Dazu passt auch, dass Webscraping-dienste wie bspw. ScrapeHero.com auf ihren Seiten bereits Verschleierungstaktiken wie IP-Rotation, Proxy-Services, rotating user agents empfehlen. Dies dürfte darin begründet liegen, dass wie gezeigt kaum noch populäre Webseiten Scraping ohne vorherige schriftliche Zustimmung erlauben. Dabei dürften die Seitenbetreiber regelmäßig kein Interesse weil keinen Nutzen daran haben, entsprechende Anfragen zu genehmigen. Während das Verbot des Zielseitenbetreibers damit eindeutig ist, wird die Frage der Rechtsverfolgung von Verstößen gegen dieses Verbot bzw. die entsprechenden Terms of Use / AGB / Nutzungsbedingungen eine Grauzone bleiben. *Damit wird Web Scraping in vielen Anwendungsfällen eine Frage der technischen Umsetzung unter der Voraussetzung des nicht entdeckt werdens bleiben* [anders formulieren, aber Aussage: Wenn Scrapen, dann technische Herausforderung die Sicherheitssysteme der Zielseite zu verarschen]

# Abbildungen

Alle Abbildungen und sonstigen Elemente, die in Verzeichnissen geführt werden, müssen gemäß der im Buchdruck üblichen Form erscheinen. Dies bedeutet im Einzelnen:

* Jedes Element ist mit einer aussagekräftigen Beschriftung sowie einer identifizierenden Nummer zu versehen.
* Bei der Nummerierung besteht, unter Wahrung der Einheitlichkeit, folgende Wahlmöglichkeit:
  + fortlaufend, z.B.: **Abb. 1**, **Abb. 2** (...) **Abb. 25** oder **Tab.** 1, **Tab.** 2, ...
  + relativ zum Kapitel, also z.B.: **Abb.** 1.1, **Abb.** 1.2 (...) **Abb.** 6.4
* Es wird ein Doppelpunkt verwendet.

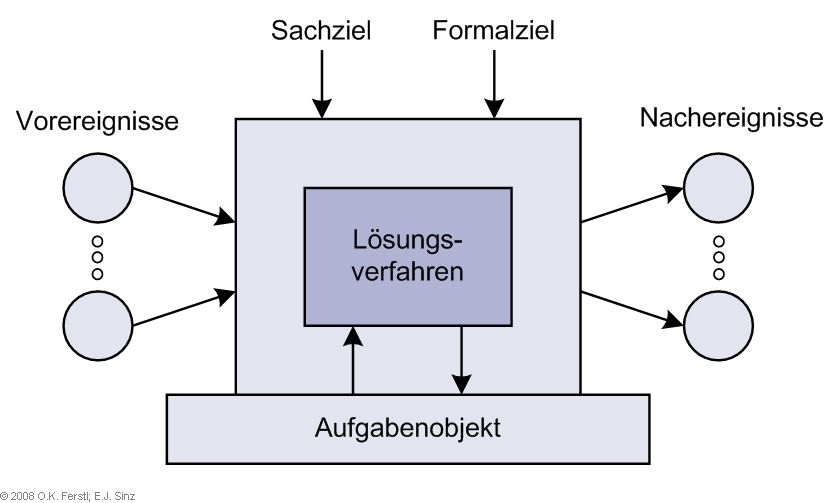


Abb. 1: Darstellung der Aufgabenstruktur (Ferstl und Sinz 2008, S. 96)

|  |  |  |
| --- | --- | --- |
| **Zeitpunkt der Lenkung** | **Aufgabenziele** | |
| Betriebswirtschaftliche Ziele | Technische Ziele |
| **Zeitdiskrete Lenkung** | Betriebswirtschaftlich orientierte Transaktionssyteme | Technisch orientierte Transaktionssysteme |
| **Zeitkontinuierliche Lenkung** | Betriebswirtschaftlich orientierte Führungssyteme | technisch orientierte Führungssyteme |

Tab. 1: Abgrenzung von Lenkungssystemen (Ferstl und Sinz 2006, S. 8)

* Für die Verweise im Text auf Abbildungen und Tabellen werden nur die entsprechenden Nummern verwendet, nicht dagegen Formulierungen wie „die folgende Abbildung“.

# Literaturverzeichnis

*Link zur Dokumentation von Scrapy (Quelle für automatisches Respektieren von robots.txt in der library*

*https://docs.scrapy.org/en/latest/topics/settings.html#topics-settings-ref*

*# [Buch]*

Ansoff HI (1988) The new corporate strategy. Wiley, New York

*# [Sammelwerk]*

Ansoff HI (1988) Mutmaßungen über die Zukunft des strategischen Managements. In: Henzler HA (Hrsg) Handbuch Strategische Führung. Gabler, Wiesbaden

*# [Zeitschrift]*

Ferstl OK, Sinz EJ (1991) Ein Vorgehensmodell zur Objektmodellierung betrieblicher Informationssysteme im Semantischen Objektmodell (SOM). WIRTSCHAFTSINFORMATIK 33(6):477–491

*# [Zeitschrift (mit DOI)]*

Sackmann S, Strüker J, Accorsi R (2006) Personalization in privacy-aware highly dynamic systems. Commun ACM 49(9):32-38. doi:10.1145/1151030.1151052

*# [Konferenzbeitrag (ohne Herausgeber, ohne Verlag)]*

Myaeng SH, Jang DH, Kim MS, Zhoo ZC (1998) A flexible model for retrieval of SGML documents. In: Proc 21st annual international ACM SIGIR conference on research and development in information retrieval, New York, S 138-145

*# [Konferenzbeitrag (ohne Herausgeber, mit Verlag)]*

Breitman K, Leite JCSP (2003) Ontology as a requirements engineering product. In: 11th IEEE international conference on requirements engineering. IEEE Comp Soc Press, Los Alamitos, S 309-319

*# [Konferenzbeitrag (mit Herausgeber, ohne Verlag)]*

Caruana R, Niculescu-Mizil A (2006) An empirical comparison of supervised learning algorithms. In: Cohen, WW, Moore A (Hrsg) Proc 23rd intern. conf. on machine learning, Pittsburgh, S 161-168

*# [Konferenzbeitrag (mit Herausgeber, mit Verlag)]*

Siakas KV, Maoutsidis D, Siakas E (2006) Trust facilitating good software outsourcing relationships. In: Richardson I, Runeson P, Messnarz R (Hrsg) Software process improvement: Proc 13th European conference. Springer, Heidelberg, S 171- 182

*# [Online-Quelle]*

Strassmann PA (1996) The value of computers, information and knowledge. http://www.strassmann.com/pubs/cik/cik-value.shtml. Abruf am 2007-01-20

# Eidesstattliche Versicherung

*# Verwenden Sie bitte diese Formulierung, wenn Sie einen Arbeit zur Anerkennung von beruflichen Kompetenzen, Projektarbeit, Masterarbeit oder Seminararbeit einreichen. Ergänzen Sie bitte den Titel ihrer Arbeit im dafür vorgesehenen Feld.*

*# Bitte schicken Sie die unterschriebene eidesstattliche Versicherung per Fax bzw. Briefpost separat an den Prüfungsausschuss der Universität Duisburg-Essen, falls Sie die Arbeit ausschließlich in elektronischer Form einreichen.*

Ich versichere an Eides statt durch meine Unterschrift, dass ich die vorliegende **#Projektarbeit / #Masterarbeit /#Seminararbeit „#Titel der Arbeit (deutsch)"** selbständig und ohne fremde Hilfe angefertigt und alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht habe, mich auch keiner anderen als der angegebenen Literatur oder sonstiger Hilfsmittel bedient habe. Die Arbeit hat in dieser oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

|  |  |  |
| --- | --- | --- |
| Ort, Datum |  | Unterschrift |