Ingenieurpraxis
Web Scraping

David Gasser
ge52noz@mytum.de
Technische Universität München

25. Mai 2021

Zusammenfassung

Das Ziel der Arbeit ist sowohl die Darlegung der Grundlagen des Web Scrapings als auch die Entwicklung eines eigenen Web Scrapers. Der erste Teil der Arbeit beinhaltet die Ausarbeitung einer Übersicht über das Thema und die dafür verwendeten Tools. Im Anschluss werden sowohl verschiedene Anwendungsgebiete aufgeführt als auch die rechtliche Lage des Web Scrapings dargelegt. Im zweiten Teil der Arbeit kommt es zu einer genaueren Betrachtung eines Web Scraping Tools namens Scrapy und zu der Umsetzung eines Projektes mit Hilfe dessen. In dem Projekt werden Sehenswürdigkeiten in der Stadt München von öffentlich zugänglichen Seiten gescrapt und anschließend korrekt kategorisch in prädefinierte Gebiete eingeordnet. Die dafür benötigte Vorgehens- bzw. Funktionsweise und der Aufbau eines Scraping Bots werden anschließend in dieser Arbeit näher beschrieben.

Keywords – Web Scraping, Web Harvesting, Screen Scraping, Data Mining, Web Data Extraction

1 Web Scraping - Übersicht

1.1 Einleitung

Die Bedeutung von Daten hat in den letzten Jahren rasant zugenommen, denn sie helfen uns Anwendungen zu personalisieren, Abläufe zu verbessern, neue Erkenntnisse zu erlangen und Vorhersagen zu treffen, wie z.B. im Bereich des Maschinellen Lernens. Die Anzahl an Daten ist dabei stets ein entscheidender Faktor, da die Realität mit jedem Datenpunkt besser approximiert werden kann bzw. jeder Datenpunkt einen verbesserten Einblick in die Realität gewährt. Die Optimierung des Zusammenhanges zwischen der Realität und dem menschengeschaffenen Artefakt resultiert schließlich immer in der Verbesserung des Ergebnisses, wodurch eine klar positive Korrelation von Daten und Resultaten entsteht [1].
Das Sammeln von Daten ist jedoch nicht immer einfach. Sowohl das Erstellen von Umfragen oder Datenbanken als auch das manuelle Auslesen von Daten aus unterschiedlichen Medien kann äußerst zeitaufwendig sein. Besonders aus dem Internet ist eine manuelle Datenextraktion oft mühselig, weil dort Seiten einzeln durchsucht und wichtige Informationen per Hand in die Datenbanken übertragen werden müssen. Diesen soeben beschriebenen Prozess kann das Web Scraping jedoch automatisieren [2].

Definition- *Das Hauptziel des Web Scrapings ist die Extraktion von Informationen von ein oder mehreren Webseiten und die Umwandlung dieser in einfache Strukturen wie Spreadsheets, Datenbanken oder CSV Dateien [2].*

Bei der Verwendung von Web Scraping bewegen sich Bots durch unterschiedliche Webseiten, indem sie sich von Link zu Link hangeln. Währenddessen parsen sie durch den HTML Code der Seiten, um vorgegebene gesuchte Inhalte zu finden. Die große Stärke des Prozesses ist vor allem seine Schnelligkeit. Ein treibender Faktor für diese ist die Möglichkeit der asynchronen Informationsverarbeitung, welche es erlaubt die Anfragen in einer beliebigen Reihenfolge zu bearbeiten. Des Weiteren besitzen Bots teilweise Zugriff auf Metadaten, die sich in dem HTML Code der Seite befinden, jedoch nicht im Browserfenster angezeigt werden. Im Vergleich zum Menschen sind Web Scraping Anwendungen also um ein Vielfaches schneller und in der Lage auf zusätzliche Daten zuzugreifen. Das allgemeine Schema bei der Suche nach Informationen lautet bei einem Web Scraper wie folgt [3]:

1. Abgreifen der HTML-Datei über die Webseite
2. Durch die Daten parsen und nach den gewünschten Inhalten suchen
3. Im gewünschten Format abspeichern
4. Optional: weitere URLs durchsuchen

Das Prinzip des Web Scrapings existiert annähernd so lange wie das Internet selbst und wurde bereits für die ersten Suchmaschinen benutzt [3]. In der heutigen Zeit ist das Prinzip vor allem durch die Programmiersprache Python deutlich massentauglicher geworden und findet nun Verwendung in vielen verschiedenen Sektoren [4].

1.2 Anwendungsbereiche

- In dem Gebiet der **Wirtschaft** wird das Web Scraping häufig verwendet, um automatisch aktuelle Daten zu sammeln. So können Vergleichsportale stets die besten Angebote finden, Firmen ihre Konkurrenz preislich unterbieten oder neue Trends verfolgt und analysiert werden [3]. Eine weitere Anwendung ist das Gruppieren von Usern nach ähnlichen Interessen, wodurch die Inhalte einer Webseite abhängig von den Aktivitäten anderer für den User personalisiert werden [5].
- Für das **Maschinelle Lernen** ist die Größe der Trainingssets von großer Bedeutung. Web Scraping Tools können in kürzester Zeit passende Bilder, Datenpunkte und Texte aus Webseiten extrahieren und in einem

gewünschten Dateiformat aufbereiten. Diese Inhalte können anschließend direkt zum Trainieren des Neuronalen Netzes eingesetzt werden [6].

- Eine weitere Verwendung findet das Web Scraping in dem Gebiet des **Journalismus**. Durch das Zusammentragen von Artikeln können Übersichten über weltweite Geschehnisse erstellt werden. Zudem können einige Anwendungen, wie zum Beispiel FactExtract [7], Inhalte von Seiten vergleichen und im Folgenden Fact Checking betreiben.
- Im **wissenschaftlichen Bereich** wird Web Scraping hauptsächlich benutzt, um Informationen aus vielen im Internet verstreuten Datenbanken und Büchern zusammenzutragen und in einer großen Datei zu speichern. Dadurch können unter anderem Gen- [8], Arznei- [9], Proteininformations- [10] und chemische Strukturdatenbanken [11] gebildet werden. Die respektiven Anwendungen hierfür sind Biospyder, DrugBank, Protein Information Crawler und ChemSpider.

In allen Bereichen kann durch Web Scraping auch überprüft werden, ob Inhaltsveränderungen an einer Webseite vorgenommen wurden. Somit werden gesammelte Daten kontinuierlich und automatisch auf den aktuellen Stand gebracht [12].

1.3 Web Scraping Tools

Von kostspieligen, bereits fertiggestellten Komplettlösungen für Kooperationen bis zu kostenlosen Open Source Libraries, gibt es eine Vielzahl an Tools, welche zum Scrapen von Daten verwendet werden können. Die große Auswahl führt dazu, dass abhängig von der Aufgabenstellung verschiedene Tools Vorteile gegenüber anderen haben. Zudem kann nicht jeder Scraper ein beliebiges Dateiformat ausgeben. Daher sollte man sich die Wahl des Tools vor dem Beginn seiner Arbeit gut überlegen. Allgemein kann zwischen den folgenden zwei Softwareansätzen von Scrapern unterschieden werden:

1.3.1 Komplettlösungen

Diese Tools sind fertig gestellte Programme, welche zum Scrapen von Daten nur noch das Ziel der Suche benötigen. Als mögliche Umsetzungen gibt es sowohl Softwareplattformen als auch Browser Extensions. Eine beliebte Methode der Suchzielbestimmung ist vor allem bei Letzteren point-and-click. Dies ermöglicht zwar eine einfache Bedienung, aber bei einer Umgestaltung der Seite muss jedes Mal das Ziel der Suche neu definiert werden. Während Browser Extensions in der Regel nur grundlegende Funktionen ausführen können und nicht sehr leistungsfähig, dafür aber kostenlos sind, besitzen Softwareplattformen deutlich mehr Funktionen und sind auch deutlich leistungsstärker.
Jedoch sind dabei oft nur die Grundfunktionen kostenlos zugänglich. Zusätzlich gibt es bei einigen Anwendungen eine Obergrenze an Anfragen. Mit der uneingeschränkten Nutzung solcher Dienste sind letztendlich immer Kosten verbunden. Allerdings bieten einige Softwareplattformen mit zum Beispiel Java Script Rendering und IP-Rotationen Lösungen für Probleme, welche in Eigenarbeit kompliziert zu lösen sind. Generell gilt noch zu erwähnen, dass kein Web Scraper jede beliebige Seite scrapen kann. Bei Misserfolg ist es daher empfehlenswert ein anderes Tool auszuprobieren.

Fast alle Browser Extensions beinhalten zwar keine Obergrenze für Suchanfragen, aber sie bieten auch keine kostenlose Option der Automatisierung. Deshalb sind sie nur für simple Scrapes mit geringer Frequenz wirklich empfehlenswert. Beispiele für Scraper in diesem Bereich sind in Tabelle 1 aufgeführt.

Name	Features	Dateiformate
Instant Data Scraper	AI Data Detection, Java Script Rendering	Excel, CSV
Web Scraper	Java Script Rendering	CSV, XLSX, JSON
Agenty	Change Detection	JSON, CSV, TSV

Tabelle 1: Scraper für Chrome

Das Benutzen von Google Chrome ist für den Gebrauch von Browser Extensions zu empfehlen, da dort mehrere und im Vergleich zu Firefox auch deutlich seriösere Optionen angeboten werden.
Im Bereich der Softwareplattformen gibt es viele verschiedene Anbieter, so dass es schwer ist eine repräsentative Übersicht dieser zu erstellen. Der für eine Arbeit optimale Scraper hängt auch hier wieder stark von dem Verwendungszweck ab. Wichtige Faktoren, auf welche man jedoch bei einer Wahl achten sollte, sind der Preis, die Features wie IP-Rotationen oder JavaScript Rendering, die Handhabung, die höchstmögliche Anzahl an gleichzeitigen Suchen und die Obergrenze an Suchanfragen. Bevor man aber ein Abonnement mit einem Anbieter eingeht, ist es empfehlenswert zuvor noch einige der kostenlosen Testversionen auszuprobieren.

1.3.2 Libraries

Diese Tools sind Softwarebaukästen, aus welchen man sich seine eigenen Web Scraper zusammenbauen und konfigurieren kann. In diesem Bereich gibt es Libraries für verschiedene Programmiersprachen. Am stärksten vertreten sind hier JavaScript, Java und Python.

Für JavaScript:

- Durch die Kombination von **NodeJS** und **Puppeteer** kann ein sogenannter "headless Browser" erzeugt werden, durch den eine Webseite wie im Browser dynamisch geladen und anschließend gescrapt werden kann [13].

Für Java:

- **JSoup** ist anfängerfreundlich, unterstützt jedoch kein auf XPath basierendes Parsing [14].
- Die **HTMLUnit Library** kann das Browserverhalten von normalen Usern simulieren und deswegen auch zum Testen von Webseiten benutzt werden. Des Weiteren ist es einfach zu automatisieren und unterstützt sowohl XPath basierendes Parsing als auch JavaScript [15].
- Das Vorgehen von **Jaunt** ist sehr ähnlich zu dem von Puppeteer. Das Tool arbeitet auch mit einem headless Browser und unterstützt JavaScript [16].

Für Python:

- **Beautiful Soup** ist ein HTML Parser, der nach einem einfachen Skript läuft. Er ist einfach zu erlernen, aber nicht sehr gut für das Scrapen dynamischer Inhalte geeignet [17].

- **Scrapy** ist auch ein HTML Parser, welcher mit sich von Seite zu Seite hangelnden Spidern arbeitet. Die große Stärke des Tools ist die Asynchronität der Abfrage von Websites. So können mehrere Seiten gleichzeitig gescrapt werden. Zudem gibt es hier noch die Optionen des Pipelinings, des dezentralen Scrapens und des Aufbaus eines Scraping Servers [18].
- **Selenium** ist eigentlich zum Testen von Webseiten gedacht, kann aber auch zum Scrapen verwendet werden. Es arbeitet im Kontrast zu den anderen zwei nicht mit HTML, sondern mit dem Document Object Model (DOM). Das DOM entsteht aus der Verarbeitung eines Quelltextes bei einem interfacelosen Aufrufen einer Seite. Der Vorteil hier ist, dass dynamische Inhalte besonders einfach ausgelesen werden können [19]. Zuletzt kann Selenium auch mit Beautiful Soup oder Scrapy kombiniert werden, um bei Letzteren das Scrapen von dynamischen Informationen zu vereinfachen [20].

Allgemein ist Python für das Web Scraping am besten ausgelegt, da die Tools in dieser Programmiersprache sowohl über die größten Communities und Library Resources verfügen als auch die Sprache an sich gut für das Web Scraping geeignet ist [21].

1.4 Rechtliche Lage

Allgemein muss man bei dem Thema Web Scraping auf die rechtliche Lage achten. Vermehrte Anfragen auf kleine Seiten können z.B. einer DDoS-Attacke [12] gleich kommen oder das Kopieren von Copyright geschützten Inhalten kann zu Urheberrechtsverletzungen führen. Folgende Grundregeln müssen deshalb in Deutschland für ein legales Scrapen stets befolgt werden [22]:

1. Die Verfügbarkeit und die Funktionsfähigkeit einer Webseite dürfen nicht durch Scraping Aktivitäten eingeschränkt werden.
2. Technische Maßnahmen zur Begrenzung des Web Scrapings dürfen nicht umgangen werden.
3. Das Urheberrecht des Webseitenbetreibers darf nicht verletzt werden.
4. Nur öffentlich zugängliche Daten dürfen gescrapt werden.

Des Weiteren gilt das Urheberrecht auch für Datenbanken. So gilt nach dem Datenbankherstellerrecht, dass nur unwesentliche Teile einer urheberrechtlichen Datenbank ohne Erlaubnis vervielfältigt werden dürfen [23]. Für den Bereich der Forschung wird dieses Recht jedoch durch die sogenannte SSchranke für Text und Data Miningäufgelockert [24]. Darin wird festgelegt, dass unter bestimmten Bedingungen ein voller Zugriff auf urheberrechtlich geschützte Datenbanken zulässig ist. Zusammenfassend sollte man sich immer vor dem Start eines Web Scraping Projektes mit der Rechtslage des Landes vertraut machen.

2 Web Scraping - Projekt

2.1 Scrapy

Für dieses Projekt habe ich mich dazu entschlossen mit dem Tool Scrapy [25] zu arbeiten, da es sehr performant, einsteigerfreundlich und einfach zu skalieren ist. Des Weiteren bietet Python im Vergleich zu anderen Programmiersprachen, wie bereits erwähnt, einige Vorteile.
Die hohe Performanz von Scrapy geht aus einem eventbasierten und asynchronen Arbeiten hervor. Dadurch können gleichzeitig und in beliebiger Reihenfolge mehrere Aufgaben in Form von Requests bearbeitet und anschließend erledigt werden. Die Einsteigerfreundlichkeit des Tools entsteht aus einer Vielzahl von Tutorien, einer guten Dokumentation und der bereitgestellten Shell. Durch Letztere kann Scrapy auch in einem Terminal verwendet werden. Hier können sehr einfach Codeabschnitte bzw. Abfragen getestet oder, für ein besseres Verständnis, Zeile für Zeile ausgeführt werden. Zudem genügen am Anfang nur wenige Zeilen Code, um einen sogenannten Spider zu programmieren. Dieser reicht bereits aus, um einen ersten rudimentären Web Scraper zu realisieren. Dessen Skalierung ist anschließend sehr einfach. Durch zusätzliche Spider und weitere Konzepte von Scrapy können Daten einfacher gesucht, gesammelt und verarbeitet werden. Durch das Programm Scrapyd kann zusätzlich eine Automatisierung des Scrapers erreicht werden.
Die grundlegenden Konzepte des Tools werden nun im nächsten Abschnitt genauer betrachtet.

2.2 Scrapy - Aufbau

Bei der Initialisierung eines Scrapy Projektes werden automatisch mehrere Dateien erzeugt, aus welchen bereits die grundlegende Struktur des Scrapers erkennbar ist. Für die elementaren Konzepte von Scrapy werden dabei die folgenden Files angelegt.

2.2.1 Spiders

In `spider.py` wird festgelegt, welche Webseiten auf welche Art und Weise gescrapt werden. Anfangs werden Requests an prädefinierte URLs geschickt. Die Responses der Webseite werden anschließend einzeln an die sogenannte `parse` Funktion übergeben. Aus dem HTML- bzw. XPath-Code der Response können nun mit Hilfe von Scrapy-spezifischen Selektoren bestimmte Abschnitte ausgewählt werden. Die Selektoren sind dabei in der Lage Codesegmente nach CSS Klassen oder nach dem Inhalt von Abschnitten zu filtern. Anschließend geben sie aus den ein oder mehreren gefilterten Bereichen den HTML oder XPath Code als Ganzes oder selektierte Elemente wie Textinhalte, URLs und Regular Expressions zurück. Die gefundenen ausgewählten Daten können im Anschluss gespeichert oder weiterverarbeitet werden. Zusätzlich können neu gefundene URLs optional mit weiteren `parse` Funktionen durchsucht werden. Das Scrapen von Webseiten und das Folgen von Links wird oft auch als Crawling bezeichnet.

2.2.2 Items

Die `items.py` Datei dient der Definition eines Containers, in welchem gescrapte Informationen gleicher Struktur in Feldern abgespeichert werden können. Die Container sind dabei als Dictionaries oder Klassen aufgebaut, wobei bei Letzteren noch

verschiedene Untergruppen existieren. Während des Scraping-vorgangs können so gefundene, zusammenhängende Daten direkt auf eine strukturierte Art und Weise gespeichert werden. Bei diesem Prozess helfen sogenannte ïtem loaders"die gesammelten Daten direkt aus dem Spider-Skript in ein davor initialisiertes Item zu laden. Ein großer Vorteil bei der Benutzung von Items ist, dass auf jedes Feld eines Containers individuelle Inputprocessing- und Outprocessing-Funktionen angewendet werden können. Dadurch lassen sich z.B. Texte oder Zahlen während des Inputs einfach formatieren oder Elemente während des Outputs einfach selektieren.

2.2.3 Item Pipelines

Nachdem die Informationen in ein Item geladen wurden, durchläuft dieses die Item Pipelines.
In der `pipelines.py` Datei wird jede Pipeline als Klasse definiert. In jeder Klasse gibt es eine Methode, welche auf jedes durchlaufende Item angewendet wird. Dieses Prinzip wird verwendet, um zu bestimmen, ob ein Item weiterhin gespeichert bleiben oder gelöscht werden soll. Die Reihenfolge, in welcher die Pipelines und ihre Methoden auf ein Item angewendet werden, wird in der settings.py Datei durch eine einfache Zahlenreihenfolge festgelegt. Mögliche Anwendungsbereiche sind z.B. die Validierung von Daten, das Löschen von Duplikaten und die Säuberung von HTML Daten.

2.3 Scrapy - Ablauf

Die soeben erklärten Konzepte spielen nun während dem Ablauf des Scrapens eine zentrale Rolle. Weitere wichtige Elemente für das Scrapen sind die Middleware, der Scheduler und die Engine:

- In der Middleware können die durchlaufenden Requests und Reponses abgeändert bzw. angepasst werden.
- Der Scheduler regelt die Reihenfolge, in welcher die Requests abgearbeitet werden.
- Die Engine regelt den Datenfluss zwischen allen Komponenten und löst basierend auf Aktionen weitere Events aus.

Aus der Zusammensetzung der Elemente ergibt sich die Architektur von Scrapy, wie sie in Abbildung 1 zu sehen ist.

Der Ablauf eines Scrapes verläuft dann wie folgt:

1. Die Engine bekommt den ersten Requests für URLs von den Spidern.
2. Die Engine gibt die Requests an den Scheduler weiter, der den Requests jeweils einen Platz in der Reihenfolge der Abarbeitung zuweist. Anschließend frägt die Engine an, welcher Request als nächstes gecrawlt werden soll.
3. Der Scheduler übergibt den nächsten Crawl-Request an die Engine.
4. Die Engine übergibt den Request nun an den Downloader, welcher die Webseite herunterladen soll. Der Request durchläuft dabei die Middleware des Downloaders.
5. Sobald die Webseite heruntergeladen wurde, schickt der Downloader diese in Form einer Response an die Engine. Dabei wird erneut die Middleware des Downloaders passiert.
6. Dieselbe Response wird von der Engine an die Spider-Datei weitergeleitet. Hier wird nun auch die Spider-Middleware durchlaufen.
7. Die Spider-Datei verarbeitet die Response, scrapt die wichtigen Informationen und schickt diese als Items oder Requests zurück an die Engine.
8. Zuletzt werden die Items an die Pipelines und die Requests, welche weitere zu scrapende URLs enthalten, an den Scheduler übergeben.

Diese Schritte werden so lange ausgeführt, bis sich kein Request mehr in der Warteschlange des Schedulers befindet.
Die Daten, welche aus der Pipeline herauskommen, können anschließend in einer Datei gespeichert werden. Dabei unterstützen die Grundfunktionen von Scrapy bereits JSON, JSON lines, CSV und XML. Das Format wird durch einen Kommandozeilen-Schalter bei dem Aufruf eines Spiders im Terminal festgelegt. Die Daten können auch in SQL-Datenbanken, Servern oder Clouds gespeichert werden. Dazu müssen jedoch entweder weitere Programme bzw. Pakete installiert oder bestimmte Einstellungen vorgenommen werden.

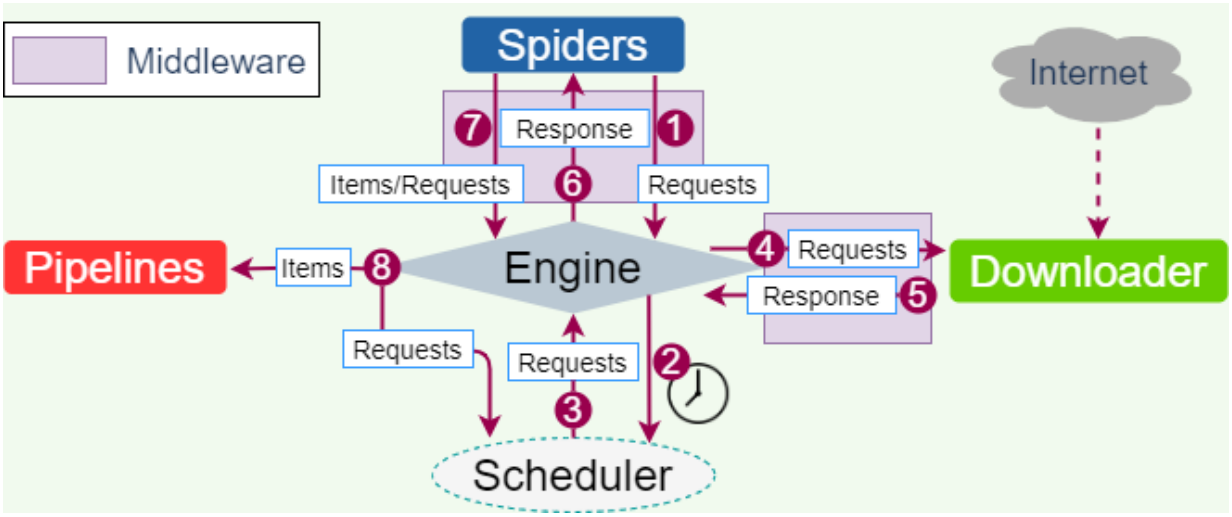

Abbildung 1: Scrapy Architektur

2.4 Der erste primitive Scraper

Nach dem Herunterladen von Scrapy bewegt man sich durch das Terminal zu dem gewünschten Speicherort des Scrapers und initialisiert dort mit dem Befehl `scrapy startproject tutorial` die Ordnerstruktur, wie sie in Abbildung 2 zu sehen ist. Tutorial ist hier ein einfacher Platzhalter für einen beliebigen Namen. Am Anfang ist es empfehlenswert mit der Webseite `quotes.toscrape.com` zu arbeiten. Diese enthält sowohl Zitate als auch eine kurze Biografie des Zitierten und wurde von einem anderen Webscrapinganbieter als Übungsbeispiel aufgesetzt. Durch das Benutzen der Seite kann man in einem kurzen Zeitraum viele Anfragen gleichzeitig schicken, ohne einen IP-Bann zu riskieren. Zudem gestaltet sich der HTML-Code der Seite eher simpel. Dadurch ist die Website vor allem für Anfänger ein guter Einstieg.

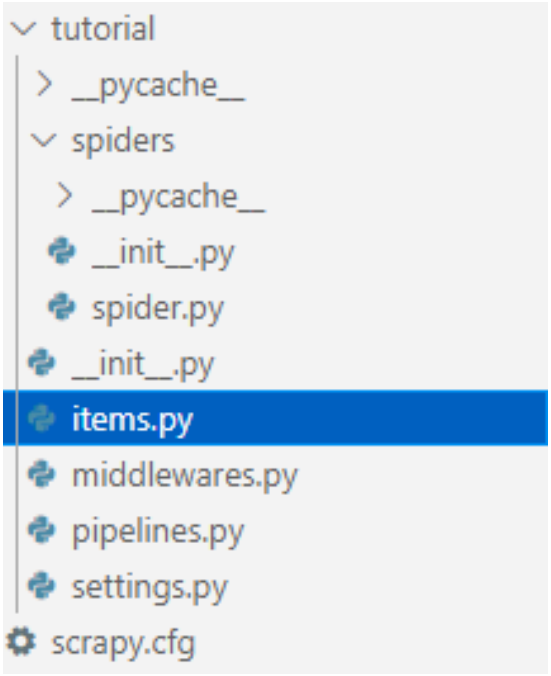

Abbildung 2: Scrapy Ordner Struktur

Für das Programmieren eines ersten primitiven Scrapers müssen nur wenige Zeilen in der `spider.py` Datei hinzugefügt werden. Diese wird im Folgenden genauer betrachtet. Anfangs muss Scrapy importiert werden und die Spider als Klasse definiert werden, welche von der `scrapy.Spider` Datei erbt. Innerhalb der Klasse wird nun der Name des Spiders festgelegt, indem der `name` Variablen ein String zugewiesen wird. Anschließend werden die Webseiten, welche gescrapt werden sollen, als Liste an die Variable `start_urls` übergeben. Im letzten Schritt definiert man nun die `parse` Funktion als Methode der Klasse und übergibt ihr neben sich selbst auch die Response der Webseite, welche in Schritt 6 an die Spiders überreicht wird, wie in Abbildung 1 zu sehen ist. In der Funktion wird festgelegt, welche Informationen aus dem HTML-Code der Seite extrahiert und wie sie anschließend abgespeichert werden. Insgesammt ergibt sich die in Abbildung 3 zu sehende Struktur.

Die Auswahl der HTML-Brackets, welche die benötigten Daten enthalten, erfolgt mit Hilfe der zuvor bereits erwähnten Selektoren.

Bei Scrapy gibt es davon zwei verschiedene Arten:

- Der XPath Selektor ist eigentlich hauptsächlich für XML Dokumente gedacht, kann aber für HTML benutzt werden. Mit ihm kann man besonders einfach nach Brackets in bestimmten Verhältnissen suchen, z.B. nach einen span-Element, das als Parent ein `div` Element einer bestimmten CSS-Klasse hat. Er ist sehr präzise, aber dessen Syntax ist, wie man in Abbildung 3 sieht, etwas kompliziert.
- Der CSS Selektor wählt, wie der Name bereits erahnen lässt, Elemente nach dem CSS Stil der Brackets aus. Wie Abbildung 3 zeigt, ist die Syntax hier eher simpel und einsteigerfreundlich. Bei einem komplizierteren HTML Code ist es aber durch die fehlende Präzision eher schwierig das richtige Objekt auszuwählen. Der Selektor dient hauptsächlich der einfacheren Benutzbarkeit und wird von Scrapy intern zu XPath umgewandelt.

```python
import scrapy

class QuoteSpider(scrapy.Spider):
    name = "quote"

    start_urls = ['http://quotes.toscrape.com']

    def parse(self, response):
        xpathvar = response.xpath("//span[@class='text']/text()").getall()
        cssvar = response.css(".text::text").getall()
        regexvar = response.css(".text::text").re(r"[^.]* genius [^.]*")
        yield{
            "quotes_xpath" : xpathvar,
            "quotes_css": cssvar,
            "sentence_regex": regexvar,
        }
```

Abbildung 3: Beispiel Scraper

In dem Beispiel aus Abbildung 3 werden die HTML Container, welche die Zitate enthalten, selektiert und der Text aus diesen extrahiert und in einem Dictionary gespeichert. Die Variablen `xpathvar` und `cssvar` sind hier zu Demonstrationszwecken komplett identisch. Beide Selektoren können theoretisch alle HTML Elemente auswählen. Am Ende ist es einem freigestellt, welchen der beiden man bevorzugt.

Den HTML Aufbau einer Webseite kann durch den Browser eingesehen werden. Bei Google Chrome wird durch das Drücken der F12-Taste eine Developerübersicht geöffnet. Soll ein bestimmtes Element ausgewählt werden, kann man entweder den HTML Code danach durchsuchen oder nach einem Rechtsclick auf das Objekt die Option Üntersuchenäuswählen, wodurch das gewünschte Elemente direkt im HTML Code angezeigt wird.

Die Aufrufe nach dem Muster `reponse.css("...")` oder `response.xpath("...")` geben einen Selektor zurück. Auf diesen Selektor können im Folgenden weitere Operationen angewendet werden:

- Durch einen anderen `.css("...")` oder `.xpath("...")` Aufruf kann die Suche weiter verfeinert werden.
- Durch die `.getall()` Funktion werden alle Inhalte extrahiert, welche die prädefinierten Kriterien erfüllen.
- Durch die `.get()` Funktion wird nur das erste Element extrahiert, welches die prädefinierten Kriterien erfüllt.

- Durch die `.re(r"...")` Funktion können alle selektierten Elemente mit Hilfe der Regex Notation durchsucht und extrahiert werden.

Die Prüfung auf Korrektheit von Selektoren kann äußerst einfach über die Scrapy Shell erfolgen. Durch den Befehl `scrapy shell url` wird die Response der Webseite in der Shell bereitgestellt. So können die Selektoren einzeln überprüft werden, was das Programmieren bzw. das Debuggen von Scrapern deutlich erleichtert.
Die einfachste Art der Datenspeicherung ist ein Dictionary, das sich innerhalb eines Yields befindet. Das Yield ist notwendig um sicherzugehen, dass die Variablen bereits definiert sind, bevor sie den Keys zugewiesen werden. Ansonsten kann es durch die Asynchronität zu Fehlern kommen.
Der Aufruf des Scrapers erfolgt anschließend über das Terminal. Im Ordner des Scrapers kann durch den Befehl `scrapy crawl tutorial` der Scraper gestartet werden. Dabei ist `tutorial` der in der Spiderklasse definierte Name. Durch den Zusatz eines Kommandozeilen-Schalters wie zum Beispiel `-o quotes.json` werden die gescrapten Ergebnisse automatisch in einem gewählten Dateiformat, hier JSON, geschrieben und in dem Ordner gespeichert, in dem der Aufruf stattgefunden hat.

2.5 Speichern von Daten in Items

Wie in Abschnitt 2.2.2 bereits erwähnt dienen Items der strukturierten Speicherung der Daten. Dabei wird der Aufbau des Datencontainers in der `items.py` Datei festgelegt. Dazu werden, wie in Abbildung 4 zu sehen ist, Scrapyfelder mit bestimmten Input- und Outputprozessoren in den Attributen der Klasse gespeichert. Die Prozessoren dienen der Bearbeitung der gespeicherten Daten.

```
class QuoteItem(scrapy.Item):
    quote_content = scrapy.Field(
        input_processor=MapCompose(remove_quotes),
        output_processor=TakeFirst()
        )
    tags = scrapy.Field()
    author_name = scrapy.Field(
        input_processor=MapCompose(str.strip),
        output_processor=TakeFirst()
        )
    author_birthday = scrapy.Field(
        input_processor=MapCompose(convert_date),
        output_processor=TakeFirst()
        )
```

Abbildung 4: Beispiel Items

Anschließend wir das Item in die Spiderdatei importiert und dort von einem Itemloader mit Daten beladen. Während der Inputprozessor die Daten abändert und dann in den Itemloader lädt, bestimmt der Outputprozessor letztendlich welche Daten in das Item geladen werden. Zu dem Beispiel in Abbildung 4:

- Durch `MapCompose` als Inputprozessor können bestimmte Operationen auf die Daten angewendet werden und so zum Beispiel unnötige Anführungszeichen entfernt werden.
- Durch `TakeFirst` als Outputprozessor wird immer das erste Element der sich im Itemloader befindlichen Daten ausgewählt.

Mit Hilfe der Prozessoren können Daten hiermit sehr einfach im großen Stil verändert und selektiert werden.
Um die Elemente aus der Spiderdatei in den Itemloader zu laden muss zuerst der Loader mit dem Item und der Response oder dem Selektor als Argument initialisiert werden. Im Anschluss kann der Loader durch Methodenaufrufe beladen werden. Hierbei gibt es drei essentielle Optionen:

1. `.add_css("Feldname", "CSS-Selektor")` als Methode entspricht `Feldname = response.css("CSS_Selektor").getall()`
2. `.add_xpath("Feldname", "XPath-Selektor")` als Methode entspricht `Feldname = response.xpath("XPath_Selektor").getall()`
3. `.add_value("Feldname", "value")` als letzte Methode ist besonders hilfreich, um Selektoren mit `.get()` und `.re(r"...")` in Variablen zwischenzuspeichern und erst dann an den Loader zu übergeben.

Sobald alle Werte den jeweiligen Feldern zugewiesen wurden, sorgt die `.load_item()` Methode dafür, dass diese an die Felder weitergegeben werden und das Item erzeugt wird.

2.6 Folgen von Links

Das Extrahieren und Folgen von Links ist für das Scraping enorm wichtig, da man dadurch nicht auf die anfängliche Webseite limitiert ist und das Scrapingverhalten somit deutlich dynamischer wird. Der Zugriff auf Daten wird erheblich vereinfacht und es ist nicht erforderlich für jede einzelne Seite einen neuen Scraper zu programmieren. Das Extrahieren des Links erfolgt wie zuvor durch die Selektoren. Auf die referenzierte URL kann einfach durch `url=response.css('a .class::attr(href)').get()` zugegriffen werden.
Je nach Anzahl der zu folgenden Links werden verschiedene Funktionen benutzt:

- Wenn einem Link einzeln gefolgt werden soll, wird `yield response.follow(url,self.parse)` benutzt. Dadurch wird die zuvor extrahierte Seite aufgerufen und mit Hilfe der `parse` Funktion durchsucht. Diese Option wird vor allem verwendet, um ergänzende Informationen für bereits initialisierte Items zu finden.

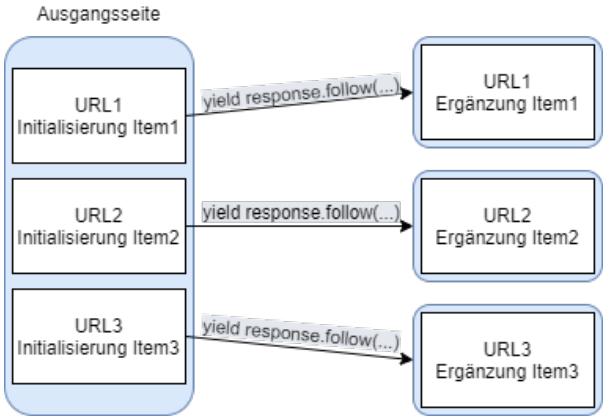

Abbildung 5: `yield response.follow(...)`

- Sollen mehrere Webseiten in beliebiger Reihenfolge besucht werden, kann bei dem obigen Selektor die get-Methode entfallen und das Innere zu `.class + a` geändert werden. Dadurch erhält man eine Liste von Selektoren. Diese Liste wird anschließend an `yield from response.follow_all(url,self.parse)` übergeben. Hierbei werden anschließend die Seiten in zufälliger Reihenfolge besucht und die Informationen extrahiert. Diese Option wird vor allem verwendet, wenn sich die zu scrapenden Daten nicht auf der ursprünglichen URL, sondern auf den verlinkten Seiten befinden und erst dort die Items initialisiert werden.

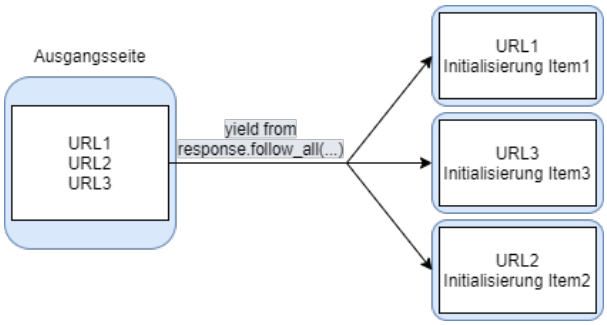

Abbildung 6: `yield from response.follow_all(...)`

Falls der Aufbau der zu folgenden Seite unterschiedlich zu den bis jetzt besuchten ist oder andere Informationen als bisher extrahiert werden sollen, muss eine neue `parse` Funktion definiert und an die `follow` bzw. `follow_all` Funktion als `self.newParse` übergeben werden.
Wenn zum Beispiel das Geburtsdatum des Autors gescrapt werden soll, muss man von der ursprünglichen Seite über den about Button zu der Biografie gelangen. Diese Biografieseite besitzt einen neuen Aufbau und es wird nach einer neuen Art von Information gesucht. Folglich ist eine neue `parse` Funktion nötig, welche auf einen derartigen HTML Aufbau angepasst ist und die gewünschten Daten abgreift. Diese Funktion wird nun `parse_autor` genannt und wird als `self.parse_autor` an eine der beiden `follow` Funktionen übergeben. Da alle Biografieseiten identisch aufgebaut sind, ist die Funktion skalierbar und kann so das Geburtsdatum aller Autoren extrahieren.

2.7 Aussortieren durch Pipelines

Jedes Item durchläuft vor seiner endgültigen Speicherung die aktivierten Pipelines. Letztere bieten die Möglichkeit, die gesammelten Items auf ihre Richtigkeit und Vollständigkeit zu überprüfen. Falls gewählte Kriterien von einem Item nicht erfüllt werden, kann es durch die Pipeline aussortiert werden. Somit wird es gelöscht und ist nicht Teil der finalen Speicherdatei.
Wie in Abbildung 7 zu sehen ist, wird eine Pipeline als Klasse definiert, welche eine `process_item` Methode besitzt. Diese Methode wird auf jedes Item angewendet unter der Voraussetzung, dass die Pipeline in der `settings.py Datei` aktiviert wurde. Am Ende der Methode muss immer entweder ein Item zurückgegeben oder die `DropItem` Exception erhoben werden.

Im Allgemeinen sind Pipelines sehr nützlich, um Daten zu validieren, Duplikate zu finden oder HTML Daten zu reinigen.

```python
from scrapy.exceptions import DropItem

class TutorialPipeline:
    def process_item(self, item, spider):
        if item["dateOfBirth"] == []:
            raise DropItem("DOB empty")
        return item
```

Abbildung 7: Aufbau einer Pipeline

Die Reihenfolge und Aktivierung von Pipelines wird in den Settings festgelegt. Wie Abbildung 8 zeigt, werden die Einstellungen in einer `dict` Struktur gespeichert. Die Festlegung der Abfolge entsteht durch die Zuweisung einer Zahl zwischen 0 und 1000. In zahlenmäßig aufsteigender Reihenfolge laufen die Items nun zukünftig durch die unterschiedlichen Pipelines.

```python
ITEM_PIPELINES = {
  'webtutorial.pipelines.DupicatePipeline': 200,
  'webtutorial.pipelines.WebtutorialPipeline': 300,
}
```

Abbildung 8: Aktivierung und Reihenfolge in den Settings

2.8 Scrapen dynamischer Webseiten

Eine große Schwäche von Scrapy ist seine Unfähigkeit dynamische Inhalte zu scrapen. Sobald der HTML Code Elemente mit reaktiven Javascript enthält, ist Scrapy nicht mehr in der Lage diese zu erkennen und zu extrahieren. Diese große Lücke kann jedoch geschlossen werden, indem Scrapy mit einem anderen Scraper namens Selenium verbunden wird. Wie in 1.3.2 bereits erwähnt, wurde Selenium als ein Webseitentestprogramm entwickelt. Deswegen kann man über dieses Tool automatisierte Verhaltensweisen definieren und mit dynamischen Inhalten interagieren.
Selenium nimmt bei der Verbindung der beiden Programme die Rolle des Downloaders ein. Das Tool öffnet die ursprüngliche Webseite, führt bestimmte Operationen aus und übergibt dann an Scrapy durch einen Selektor den neuen Inhalt der Webseite. Die neue Architektur des Scrapers ergibt sich zu der in Abbildung 9.
Selenium operiert mit der Hilfe eines sogenannten Drivers. Hier gibt es die Möglichkeit mit verschiedenen Browsern zu arbeiten. Ich habe mich einfachheitshalber für Google Chrome entschieden. Im ersten Schritt muss die ausführbare Datei `chromedriver.exe` heruntergeladen werden. Dabei ist darauf zu achten, dass die Versionen des Drivers und des eigenen Browsers übereinstimmen. Nach dem Importieren von Selenium kann man den Driver mit dem Befehl `driver = webdriver.Chrome(...)` starten. Als Argumente müssen hierbei der Speicherort der Chromedriver Datei und die vorgenommenen Einstellungen übergeben werden.
Im nächsten Schritt öffnet sich ein Browserfenster und der Driver ruft die zu scrapende Webseite auf. Während des Ladevorgangs werden keine weiteren Prozesse durchgeführt. Erst wenn die Seite fertig geladen hat können

weitere Operationen durchgeführt werden. Das hat zum Vorteil, dass es zu keinen Synchronitätsfehlern kommt. Im Anschluss können nun unter anderem Buttons gedrückt, Webseiten gescrollt und Eingaben getätigt werden. Die Auswahl welche Operation auf was angewendet wird, erfolgt genauso wie bei Scrapy durch Selektoren. Durch die Aufrufe `driver.find_element_by_xpath(..)` oder `driver.find_element_by_css(..)` können Elemente im HTML Code gefunden werden. Durch Zusätze wie `.click()` oder `.sendKeys(..)` können so reaktive Webseiten beeinflusst und verändert werden.
Sobald die zu scrapenden Informationen zugänglich sind, kann der Datenaustausch zwischen Scrapy und Selenium stattfinden. Dieser ist sehr simpel und kann durch eine Zeile realisiert werden, indem ein neuer Scrapy Selektor definiert wird. Als Argument wird dabei der HTML Code des Drivers durch die Variable `text` übergeben.
Mit diesem gespeicherten Selektor kann man nun wie zuvor vorgehen und weitere Selektoren wie `.xpath` und `.css` anwenden. Nachdem die gesuchten Daten gespeichert wurden, können nach dem gleichen Muster weitere Selenium Operationen auf den Driver durchgeführt werden. Sobald die neuen zu scrapenden Daten einsehbar sind, kann erneut ein Scrapy Selektor definiert werden. Diese Vorgehensweise wird solange durchgeführt, bis alle wichtigen Informationen extrahiert wurden. Nachdem alle gewünschten Informationen abgegriffen wurden, muss der Driver mit `driver.quit()` oder `driver.close()` geschlossen werden. Ansonsten kann eine zu große Anzahl an geöffneten Fenstern zu Problemen führen.
Eine Schwäche von Selenium ist sein Umgang mit Exceptions. Während Scrapy bei fehlerhaften Selektoren einfach einen leere Liste zurückgibt und diese im Programm zu überspringen scheint, treten bei diesen in Selenium unerwartete Verhaltensmuster auf. Deshalb ist es beim Benutzen von Selenium sehr wichtig mit try-except-Blöcken zu arbeiten. Letztere können Fehler abfangen und anschließend Alternativen vorgeben oder Defaultwerte festlegen.

Im Allgemeinen gilt zu sagen, dass das Scrapen mit Selenium im Vergleich zu Scrapy durch einen höheren Ressourcenaufwand deutlich langsamer ist. Daher ist es empfehlenswert so wenig Selenium wie möglich zu verwenden. Da jedoch immer mehr Webseiten dynamisch konzipiert werden, ist ein Einsatz des Tools selten vermeidbar.

2.9 Finaler Scraper

In diesem Teil der Arbeit soll nun das Ziel des Scrapens von Sehenswürdigkeiten in München realisiert werden. Die wichtigste Voraussetzung für einen guten Scraper ist das Finden einer geeigneten Website. Die Seite muss dafür über eine möglichst große Datenbank verfügen und das Web Scraping allgemein erlauben. Webseiten, welche diese Voraussetzung erfüllen sind `muenchen.de`, `tripadivor.de` und die Reisesparte von Google `google.com/travel`. Im Folgenden werden diese drei Seiten genauer analysiert um festzustellen, welche am Besten für den Scraper geeignet ist:

- Die Seite *muenchen.de* verfügt über eine große Anzahl an Daten hoher Qualität. Die Einträge zu Sehenswürdigkeiten sind gut herausgearbeitet und es gibt auch viele noch eher unbekannte Orte. Des Weiteren sind die gewünschten Informationen nicht reaktiv, wodurch das Arbeiten mit Selenium nicht nötig wäre. Das große Problem dieser Seite ist jedoch, dass es keine komplette Übersicht über alle Sehenswürdigkeiten gibt. Stattdessen sind die Daten über viele unterschiedliche Seiten verstreut, was einen sehr hohen Arbeitsaufwand zur Folge hat.
- Im Gegensatz zu seinem Vorgänger besitzt *Tripadvisor* eine volle Übersicht über alle Sehenswürdigkeiten in München, wodurch das Scrapen auf dieser Seite sehr einfach wäre. Die Qualität der Daten ist jedoch etwas geringer und ein paar der Seiten sind dynamisch.
- *google.com/travel* besitzt zwar sehr gute und einfach zugängliche Daten, aber die Seite selbst als auch alle verlinkten sind dynamisch konzipiert. Dadurch wäre man gezwungen sehr viel Selenium zu verwenden, was eine deutlich längere Laufdauer des Scrapers zur Folge hätte.

Letztendlich habe ich mich dazu entschieden mit Tripadivsor zu arbeiten, da die Struktur der Seite dem Web Scraping sehr entgegen kommt, die Qualität der Daten immer noch

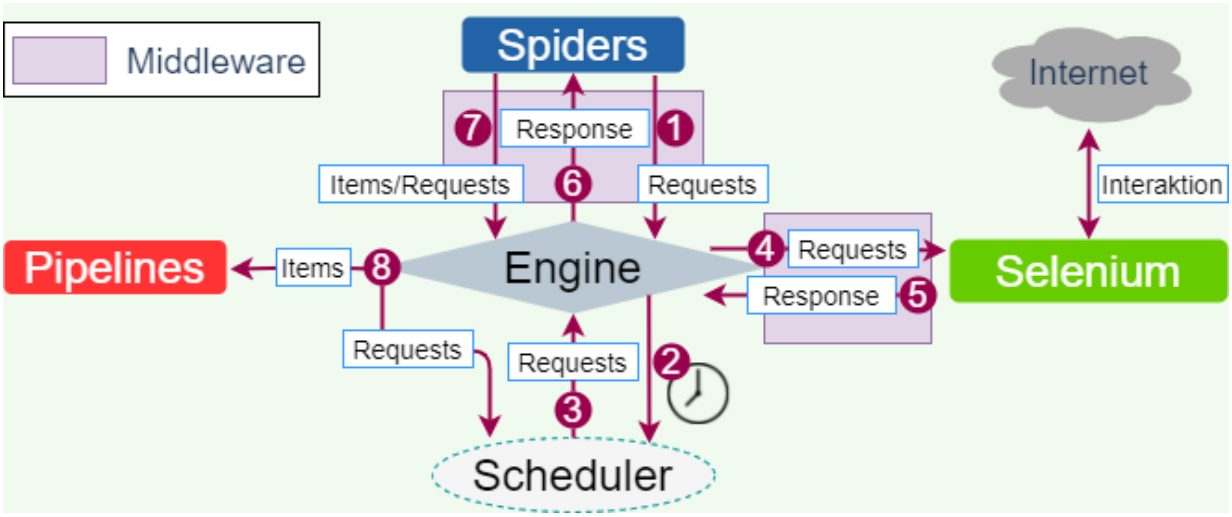

Abbildung 9: Scrapy Selenium Architektur

ausreichend gut ist und die reaktiven Webseiten den Scraping Prozess nicht allzu stark verlangsamen.

2.9.1 Welche Daten werden gescrapt?

Das Ziel des Scrapers ist es zu jeder Sehenswürdigkeit den Namen, die Kategorie, die Beschreibung, die Adresse und die vorgeschlagene Aufenthaltsdauer zu extrahieren und in einer JSON Datei zu speichern. Nicht alle Daten sind zwangsläufig für alle Sehenswürdigkeiten vorhanden, weil zum Beispiel der Englische Garten keine Adresse hat. Da Scrapy mit Exceptions, wie einen für die Seite fehlerhaften Selektor, gut umgehen kann und wie bereits erwähnt eine leere Liste retourniert, kommt es durch fehlende Daten und damit auch fehlende HTML Blöcke zu keinen Problemen. In der JSON Datei wird die leere Liste anschließend durch ein `None` ersetzt, wodurch dieser Fall allgemein keiner weiteren Bearbeitung bedarf.

2.9.2 Welchen Ablauf hat das Scraping?

Man startet mit der URL zur Gesamtübersicht aller Sehenswürdigkeiten in München. Wie in Abbildung 10 zu sehen ist, sind auf dieser Webseite alle Sehenswürdigkeiten in einem `div` Feld gespeichert. Jedes dieser Felder besitzt die gleiche Klasse.

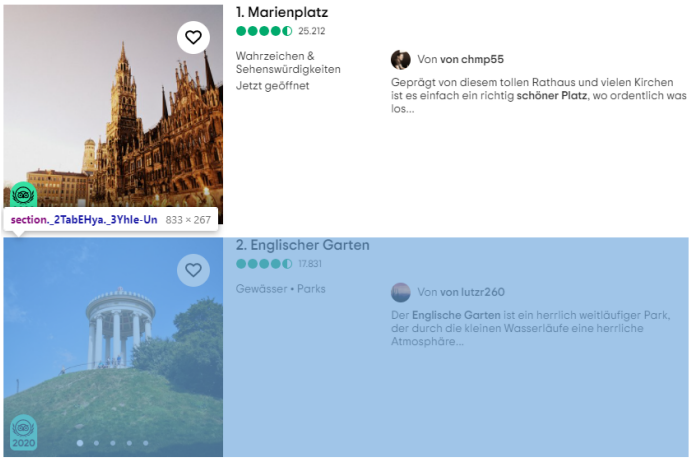

Abbildung 10: HTML Aufbau Tripadvisor

Von dieser Seite kann man bereits den Namen und die Kategorie der Sehenswürdigkeit einsehen. Daher ist ein Aufbau wie in Abbildung 11 zu sehen ist sinnvoll. Dazu muss eine Liste mit zurückgegebenen Selektoren für diese `div` Box erzeugt werden und anschließend jeder Selektor einzeln abgearbeitet werden.
Für jeden zurückgegebenen Selektor wird zuerst ein Item initialisiert. Anschließend wird aus dem retournierten Selektor der Name und die Kategorie mit Hilfe von weiteren Selektoren extrahiert und in dem Item gespeichert. Im nächsten Schritt wird die URL der verlinkten Seite extrahiert und an die `response.follow(...)` Funktion übergeben. Diese Funktion wird hier verwendet, da das Item bereits initialisiert wurde und nur einem Link gezielt gefolgt werden soll.
Auf der gescrapten URL wird nun eine andere `parse` Funktion namens `parse_sight` angewendet. Da die verlinkte Seite sowohl statisch als auch dynamisch sein kann, muss als erstes festgestellt werden, um welchen Aufbau es sich dabei handelt. Das lässt sich mit Scrapy einfach überprüfen, indem nach einem HTML Element unterschieden wird, welches in einem Seitenaufbau existiert, im anderen aber nicht.

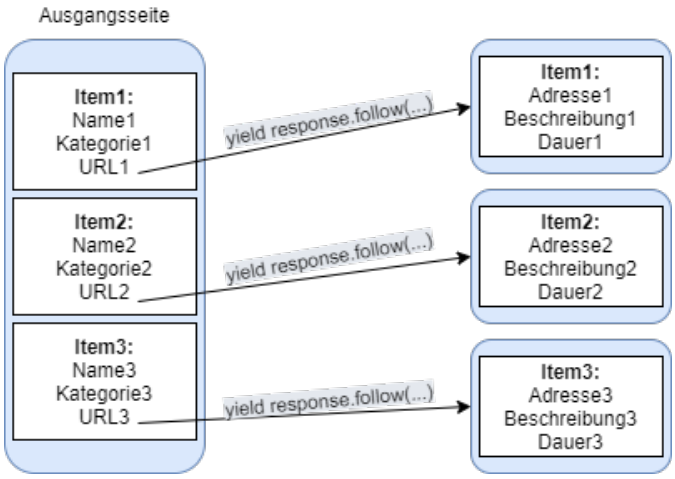

Abbildung 11: Scraping Ablauf Tripadvisor

Der Ablauf unterscheidet sich wie folgt:

- Ist die Seite reaktiv, muss Selenium verwendet werden. Der Driver wird daraufhin mit der URL initialisiert und die Cookies werden im Folgenden akzeptiert. Anschließend übergibt Selenium den Inhalt der Seite an Scrapy. Die Adresse und die vorgeschlagene Aufenthaltsdauer sind nun einsehbar und werden in den Itemloader geladen. Bei der Adresse kann man sich den typischen Aufbau nutzen und mit Hilfe von Regular Expressions nach dem Satz suchen, welcher das Wort "Bayernënthält. Für das Finden der Beschreibung muss eine weitere Selenium Operation durchgeführt werden, denn erst durch das Drücken des "mehr"Buttons ist die gesamte Beschreibung einsehbar. Daher muss mit Hilfe eines Selektors dieser "mehr"Button gefunden und geclickt werden. Anschließend werden die Informationen der veränderten Webseite von Selenium an Scrapy weitergegeben und wie zuvor in einen Itemloader geladen und gespeichert. Da der "mehr"Button jedoch nicht bei allen dynamischen Seiten existiert, muss hier mit einem try-except-Block gearbeitet werden. Falls der Knopf nicht vorhanden ist, werden stattdessen die Kommentare von Reisenden gescrapt. Nachdem alle Daten gespeichert wurden, wird der Driver beendet und die nächste `div` Box betrachtet.
- Ist die Seite statisch so vereinfacht sich der Ablauf des Scrapens beträchtlich, weil kein Selenium verwendet werden muss. Alle Informationen können durch die Selektoren abgegriffen, in Itemloaders geladen und anschließend gespeichert werden. Danach wird die nächste `div` Box betrachtet.

Unabhängig davon, ob die Seite dynamisch oder statisch ist, wird hier in der `parse_sight` Funktion mit XPath gearbeitet, weil der HTML Aufbau der verlinkten Seiten recht kompliziert ist und viele Objekte die gleiche Klasse besitzen. So ist eine gezielte Auswahl von Elementen mit Hilfe von CSS nur schwer möglich.
Nachdem die `div` Boxen aller Sehenswürdigkeiten abgearbeitet wurden, extrahiert man nun den Link zur nächsten

Seite, auf welcher erneut die `parse` Funktion angewendet wird. Der Vorgang endet, sobald keine nächste Seite existiert oder sobald ein prädefiniertes Limit erreicht wurde.

2.9.3 Konzipierung von Items und Pipelines

Die Attribute der Sehenswürdigkeitsitems sind als einfache Scrapy Felder definiert. Die Input- und Outputprozessoren sind ebenfalls recht simpel. Die Inputprozessoren entfernen ausschließlich unnötige Zeichen mit Hilfe von `string.strip()` und die Outputprozessoren benutzen alle die `TakeFirst()` Funktion.
Pipelines werden für diesen Scraper nicht wirklich benötigt. Weil alle Informationen von einer Seite kommen, sind Duplikate nicht möglich. Zudem müssen die Daten nicht validiert werden. Folglich gibt es für Pipelines in diesem Fall keinen wirklichen Verwendungszweck.

3 Zusammenfassung

Das Prinzip des Web Scrapings kann in vielen Bereichen für das automatische Extrahieren von Daten verwendet werden. Zusätzlich gibt es für die technische Umsetzung eine Vielzahl an Tools, welche jeweils ihre Vor- und Nachteile haben. Außerdem ist es vor dem Beginn des Scraping Projektes empfehlenswert sich mit der rechtlichen Lage vertraut zu machen. Das ursprüngliche Ziel des praktischen Teils der Ingenieurspraxis wurde erreicht. Das Scrapen der Sehenswürdigkeiten ist anhand von der Seite Tripadvisor erfolgt. Die Kategorisierung wird dabei bereits durch die User der Webseite vorgenommen und war so nicht weiter nötig. Die Informationen der Sehenswürdigkeiten wurden nach dem Scrapen abschließend in einer JSON Datei gespeichert. Die weitere Verwendung der gesammelten Daten ist dadurch sehr simpel.

Literatur

[1] E. G. Ularu, F. C. Puican, A. Apostu, M. Velicanu, *et al.*, "Perspectives on big data and big data analytics," *Database Systems Journal*, vol. 3, no. 4, pp. 3–14, 2012.

[2] Sirisuriya, De S and others, "A comparative study on web scraping," *Proceedings of 8th International Research Conference*, 2015.

[3] R. Mitchell, *Web scraping with Python: Collecting more data from the modern web*, pp. 4–11. "O'Reilly Media, Inc.", 2018.

[4] D. M. Thomas and S. Mathur, "Data Analysis by Web Scraping using Python," in *2019 3rd International conference on Electronics, Communication and Aerospace Technology (ICECA)*, p. 451, 2019.

[5] E. Vargiu and M. Urru, "Exploiting web scraping in a collaborative filtering-based approach to web advertising.," *Artif. Intell. Research*, vol. 2, no. 1, pp. 44–54, 2013.

[6] S. Borde, A. Rane, G. Shende, and S. Shetty, "Real estate investment advising using machine learning," *Int. Res. J. Eng. Technol*, vol. 4, no. 3, pp. 1821–1825, 2017.

[7] E. N. SARR, O. SALL, and A. DIALLO, "Factextract: Automatic collection and aggregation of articles and journalistic factual claims from online newspaper," in *2018 Fifth International Conference on Social Networks Analysis, Management and Security (SNAMS)*, pp. 336–341, 2018.

[8] S. Johnson and D. BHSAI, *Design & Implementation of a Pipeline for High-throughput Enzyme Function Prediction*. PhD thesis, George Mason University, 2006.

[9] Wishart, David S and Knox, Craig and Guo, An Chi and Shrivastava, Savita and Hassanali, Murtaza and Stothard, Paul and Chang, Zhan and Woolsey, Jennifer, "Drugbank: a comprehensive resource for in silico drug discovery and exploration," *Nucleic acids research*, vol. 34, no. suppl_1, pp. D668–D672, 2006.

[10] U. Mayer, "Protein information crawler (pic): Extensive spidering of multiple protein information resources for large protein sets," *Proteomics*, vol. 8, no. 1, pp. 42–44, 2008.

[11] Day, Nicholas E, *Automated analysis and validation of open chemical data*. PhD thesis, University of Cambridge, 2009.

[12] B. Zhao, "Web scraping," *Encyclopedia of big data*, pp. 1–3, 2017.

[13] "https://github.com/puppeteer/puppeteer."

[14] "https://jsoup.org/."

[15] "https://htmlunit.sourceforge.io/."

[16] "https://jaunt-api.com/."

[17] C. Zheng, G. He, and Z. Peng, "A study of web information extraction technology based on beautiful soup.," *JCP*, vol. 10, no. 6, pp. 382–383, 2015.

[18] D. Kouzis-Loukas, *Learning scrapy*. Packt Publishing Ltd, 2016.

[19] R. S. Chaulagain, S. Pandey, S. R. Basnet, and S. Shakya, "Cloud based web scraping for big data applications," in *2017 IEEE International Conference on Smart Cloud (SmartCloud)*, pp. 139–140, 2017.

[20] W. Hejing, L. Fang, Z. Long, S. Yabin, and C. Ran, "Application research of crawler and data analysis based on python," *Associate Editor-in-Chief*, p. 64, 2020.

[21] D. M. Thomas and S. Mathur, "Data analysis by web scraping using python," in *2019 3rd International conference on Electronics, Communication and Aerospace Technology (ICECA)*, pp. 450–454, IEEE, 2019.

[22] "OLG Frankfurt a.M., Urteil vom 05.03.2009 - 6U 221/08."

[23] "§§ 87a ff. UrhG, Schutz des Datenbankherstellers."

[24] "§60d UrhG, Text und Data Mining."

[25] "https://scrapy.org/."