universität
wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Web content mining: towards an adaptable framework for delivering structured data“

verfasst von / submitted by

Jakob
Rathmair BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2022 / Vienna, 2022

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 921
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Informatik
Betreut von / Supervisor:	Univ.-Prof. Dipl.-Ing. Dr. Wolfgang Klas

Acknowledgements

I want to thank Univ.-Prof. Dipl.-Ing. Dr. Wolfgang Klas for his useful advice and supervision throughout the entire process of this master thesis. I want to thank Dipl.-Ing. Dr. Peter Kalchgruber for his support during the conceptual elaboration of the topic of the thesis.
My time as a student was a very fulfilling and happy time that would have never been possible without the support of my family and friends. Therefore, I want to thank my parents Michaela and Albert as well as my siblings Veronika, Franz, Josef and Martin. I also want to thank my girlfriend Julia for her emotional support over the last years.

Without all of you I would never have been able to come this far.

Abstract

The World Wide Web consists of a massive amount of interconnected Web pages which provide information on a wide variety of topics. This information is usually presented in a human-readable way. However, it is challenging to extract the underlying data automatically. Often, explicit information about the context of the presented information is missing. In addition, Web pages differ greatly in design, and it is therefore hardly possible to identify general representation patterns.
This paper addresses this problem and presents a system which extracts information from a wide variety of Web pages and stores it in a uniform format. The presented framework is flexible and can be easily adapted to varying requirements because it is based on a modular microservice architecture. Furthermore, the presented system can be hosted in a modern cloud infrastructure and thus the available performance can be scaled on demand. Finally, the high flexibility of the system is demonstrated by extracting data from a series of websites, which use diverse information presentation formats and styles. In addition, the durability of the system is tested by successfully processing over 100 000 Web pages automatically over a period of one week.

Kurzfassung

Das World Wide Web setzt sich aus einer riesigen Menge von vernetzten Webseiten zusammen, welche Informationen über unterschiedlichste Themen bereitstellen. Diese Informationen werden üblicherweise so dargestellt, dass sie für Menschen übersichtlich und leicht verständlich sind. Es wird aber selten Wert daraufgelegt, dass diese Informationen von Computer Programmen automatisiert ausgelesen und interpretiert werden können. Dafür fehlen oft explizite Angaben über die Bedeutung von dargestellten Informationen. Außerdem unterscheiden sich Webseiten im Design sehr stark und es ist daher kaum möglich, allgemeine Darstellungsstrukturen zu identifizieren.
In dieser Arbeit wird diesem Problem nachgegangen und ein System vorgestellt, mit welchem Informationen von unterschiedlichsten Webseiten extrahiert und in einem einheitlichen Format abgespeichert werden können. Das vorgestellte Framework ist flexibel einsetzbar und kann durch die Verwendung einer modularen Microservice Architektur leicht auf veränderte Anforderungen angepasst werden. Des Weiteren kann das vorgestellte System in einer modernen Cloudinfrastruktur gehostet werden und somit die verfügbare Leistung bedarfsorientiert skaliert werden. Letztendlich wird die hohe Flexibilität des Systems unter Beweis gestellt, indem Daten von einer Reihe von Webseiten mit unterschiedlichen Darstellungsformen extrahiert werden. Außerdem wird die Beständigkeit des Systems getestet, indem über einen Zeitraum von einer Woche erfolgreich über 100 000 Webseiten automatisiert verarbeitet werden.

Contents

List of Tables

List of Figures

1 Introduction

1.1 Motivation

The World Wide Web consists of more than three billion Web pages, according to Maurice de Kunder [2]. The massive amount of information available on those sites is primarily provided based on the human-oriented presentation format, called the Hypertext Markup Language (HTML). In the early days of the World Wide Web, pages were primarily statically stored on Web servers as Files. With the introduction of new technologies for dynamic Web content creation, the spread of information via the internet has accelerated rapidly. Nowadays, the biggest content providers on the internet create Web pages dynamically. These are commonly based on general design structures called templates, containing the specific information requested by the user. The structured data, which is usually stored in decentralized database systems for fast and consistent data access, is often not accessible to the public. For many applications, however, this structured data is of interest. By structured data we mean descriptions of objects retrieved from those underlying databases that are displayed on Web pages. Access to this structured data is a necessity for many interesting application areas. For example, it can be used to compare information provided by different content providers, or it can be used as an input for data models in a variety of scientific domains.
Viewing Web documents like HTML files using a modern Web browser is great for everyday internet users to gain new knowledge, but a major challenge for content mining applications to identify the underlying information in the mass of navigational elements, design elements and general information. Due to great heterogeneity and the lack of structure on Web pages, the identification and the extraction of relevant data from semi-structured or completely unstructured information collections is a crucial research topic in modern computer science. A related field is information retrieval, which concerns the identification of relevant Web documents. There are already advanced approaches which are well-developed and documented in the academic and commercial field. Web content mining or information extraction is however still in an early research state.

Web content mining addresses the process of extracting needful informative data from websites, as outlined in more detail by Mughal [3]. This content can include audio, video, text, or structured data. The different types of information representations can be distinguished as unstructured data, structured data, semi-structured data, and multimedia data.

As mentioned above, the most common documents for representing information on the Web are HTML Web pages. HTML documents contain the published information

within HTML tags and can be assisted by technologies such as CSS and JavaScript to improve readability and adding immersive features to Web pages. These HTML elements add structural semantics to information on the Web. Some HTML structures like lists and tables may allow greater interpretation of the structure of the underlying data while other elements are only used for creating a particular appearance.

By analysing the structure of HTML documents, certain patterns can indicate structural information about the underlying data which gives us the possibility to identify properties and values related to referenced data entities. For some Web resources which are only insufficiently or not encoded at all, other specialized techniques like semantic text analysis or multimedia analysis techniques are necessary to recognize the structure of the underlying information.

1.2 Goal

The goal of this master thesis is to develop an approach to extract structured data from Web pages provided by dynamic Web applications and store it in a consistent data representation format. By structured data we mean descriptions of objects retrieved from underlying databases that are presented on Web pages. However, we do not mean high level structures that are generated for the appearance and layout of these pages. Thereby, a variety of different information representations on the Web shall be taken into account in the scope of this paper. As a result, the developed prototype should be applicable with many different content providers. To achieve this, the chosen approach must be flexible and adaptable. In order to consider the specific information representations, the developed prototype should be highly configurable. Furthermore, the system should be able to automatically find and process relevant Web pages (crawling). In addition, the system should be built in such a way that it is easily extendable and the range of functions can be conveniently increased without great integration effort.
Part of the master thesis is also the analysis of the approach. In doing so, we want to look closely at the flexibility of the chosen approach, as well as its expandability and durability.

1.3 Outline

In this thesis we discuss in Chapter 2 which data representations are used on the Web and which approaches for automated data extraction already exist. In Chapter 3, we discuss the goal of this project in detail and list the requirements needed to achieve it. Simultaneously, we also describe how we will evaluate the resulting prototype to analyze the system with respect to the defined goals. Chapter 4 explains important terms and technical concepts that are important for the implementation of the project. Chapter 5 then explains in detail how the prototype was developed, which components interact with each other and the system's general process flow. In Chapter 6, we then describe an illustrative use case to show the application of the system. In Chapter 7, we evaluate the

system against the criteria we have defined earlier. And finally, Chapter 8 concludes this thesis' findings and the further outlook.

2 Related Work

2.1 Data on the Web

According to Tim Berners-Lee [4], the inventor of the World Wide Web (WWW): "the WWW was developed to be a pool of human knowledge, which would allow collaborators in remote sites to share their ideas and all aspects of a common project". The Web is compiled by documents which are exchanged by use of the HTTP protocol. Those documents or Web pages are identified by a *Universal Resource Identifier* (URI) [5]. A *Uniform Resource Locator* (URL), is a form of URI which expresses an address which maps onto an access algorithm using network protocols. These Web documents are usually encoded in the *Hyper Text Markup Language* (HTML) [6]. Web users can render those pages by the use of an *internet browser*. Web pages primarily present information in a human readable way, since the target audience is mostly human users rather than machines. Users can consume an overwhelming amount of information on the Web, but it is challenging for *Web Content Mining* applications to understand the structure of the presented data and to assess the relevance of the presented information. Berners-Lee believed as early as 1994 [4] that objects on the Web should no longer just be human-readable documents but should also contain machine-readable semantic information.

This idea to "weave a Web that not only links documents to each other but also to recognize the meaning of the information in those documents" [7] is called the *Semantic Web*. Adding semantics promotes automatic understanding, processing and exchange of data in Web documents. The ultimate goal of the Semantic Web is to transform the internet from millions of independent database islands into a single gigantic database Pangea. To achieve this goal, the content on Web pages has to be enriched with descriptive tags or metadata using well-defined standards.

One of the most widely adopted standards is the *Resource Description Framework* (RDF) [8]. RDF is a model for data interchange on the Web. In this approach, each statement consists of the three entities: subject, predicate, and object; where a resource is described in more detail as a subject with another resource or value (literal). To implement RDF data in a machine-readable format in HTML documents, multiple serialization standards have been developed. The most popular RDF serialization formats are: RDF in Attributes (RDFa) [9], Turtle [10] and JSON-LD [11].

The Semantic Web field has made significant progress since the existence of the idea [12], best understood by means of the wide adoption of supporting applications like

Schema.org, industrial knowledge graphs, Wikidata and ontology modelling applications. Nevertheless, in many areas these standards are not or only partially used and therefore make it difficult to interpret a large part of the information on the Web. Often, these machine-readable annotations are only included when there is a direct benefit to the content provider. Since the large search engines can read some of these standards and thus the search results are displayed better and possibly earlier, semantic annotations are included on Web pages mainly for this reason.

2.2 Web Mining Definition

In the following, the term Web mining describes the extraction of implicit, non-trivial and useful knowledge from Web resources.

Etzioni [13] generalizes Web mining into three different subtasks:

- *Resource discovery*: the task of locating new documents with relevant data on the Web.
- *Information extraction*: the task of automatic extraction of specific data from newly discovered Web resources.
- *Generalization*: the task of uncovering general patterns across different Web sites in order to make information comparable.

2.3 Resource Discovery/ Web Crawling

The most prominent field of application of resource discovery from Web Crawlers is the creation of searchable indices of Web documents. Those indices are created by search engines like Google, Yahoo or Bing to locate relevant documents according to specific search queries. Yuan et al. [14] estimate that Web Crawlers cause currently about 40% of all network traffic, which demonstrates the widespread use and importance of this topic. HTML documents consist of a structure of nested HTML elements, which are called HTML tags. HTML documents usually contain hyperlinks to other documents, identified by the tag "a". Web Crawlers traverse from document to document by using those hyperlinks to visit many Web pages. During this process, the crawling applications keep track of previously visited pages and pages still to visit. This technique enables an automatized approach for discovering Web documents across the World Wide Web. There are different techniques to perform this task. For instance a traditional approach is an *Incremental Crawler* which accesses one document at a time and updates the corresponding data. Those standard Incremental Crawlers do not assess the pages they discover, the goal is most of all to locate all pages within a defined domain. This process can take a long time, because of the sheer size and number of Web documents available. In contrast, more advanced approaches like *Focused Crawler*, which rank pages to discover according to the assumed relevance, are more efficient according to Xhumari [15]. Focused Crawlers do not discover new Web documents by random, but choose pages by some pre-defined

topic of interest. Irrelevant regions of the Web are ignored by Focused Crawlers in order to save resources.

Chakrabarti et al. [16] describe an approach of a Focused Crawler which selectively seeks out pages according to some pre-defined set of topics. The selected topics for this experiment represent a relatively narrow segment of the Web. Therefore, the hardware and network resources needed to achieve respectable coverage is relatively low. In a relatively short time, this approach is able to recommend relevant Web pages with regard to an initial set of pages by exploring the Web in great depth rather than breadth.

Diligenti et al. [17] examine a Focused Crawling approach called *Context Focused Crawler*, which builds a model for the context in which topically relevant pages occur on the Web. For this purpose, the links and the content of the documents that are closely linked to the target pages are modelled, in order to find content that belongs to a certain category.

Chakrabarti et al. [18] employ a general enhancement for Focused Crawlers, which helps assigning improved priorities to unvisited pages. For this purpose, DOM [19] features of Web documents are analyzed to calculate precise predictions about the relevance of a link.

2.4 Information Extraction (IE)

We define information or data extraction as the task of extracting relevant information from Web documents. Information extraction in the context of Web pages is also often called *Web Scraping*. An increasing number of websites use markup standards like Microformats [20], RDFa [9] or Microdata [21], to enrich HTML documents with semantics of the underlying data. The extraction of information provided with these standards is further referred to as information extraction from structured data.

2.4.1 IE from Structured Data

Structured data refers to data that has defined format with a high degree of organization. This means that the data is organized in an identifiable structure to allow it to respond to queries in order to retrieve information. An increasing number of websites have started to annotate structured data within their HTML pages. Especially markup standards like Microdata, RDFa and Microformats are widely used as these formats of meta-data is crawled by search engines such as Google, Yahoo! and Bing which use the data to enrich search results and show users relevant descriptions.

The *Web Data Commons* project [1] has extracted all Microformat, Microdata and RDFa data from the Common Crawl [22] Web corpora from 2009/2010 as well as February 2012 and provides this extracted data in the form of RDF-graphs. As a result, about 4.5 billion Web pages, mostly HTML documents, have been processed.

	2009/2010	2012
Crawl Dates	09/09 – 09/11	02/12
Total URLs	2.8B	1.7B
HTML Pages	2.5B	1.5B
Pages with Data	148M	189M

Figure 2.1: Comparison of the Common Crawl Corpora [1]

Figure 2.1 shows that the relative fraction of Web pages that contain structured data has increased from 6% in 2010 to 12% in 2012. The analysis of the data representation formats shows that especially RDFa and Microdata are used more often, while the use of Microformats remains more or less constant. The deployed entity types as well as the deployed formats seem to closely correlate to the announced support of the big Web companies for specific types and formats, meaning that Google, Microsoft, Yahoo! almost exclusively determine adoption.

Based on the work of the Web Data Commons project, Meusel et al. [23] have extracted structured data from another Web corpora provided by the Common Crawl foundation from 2013 to further analyse the deployment of annotation standards for structured data. They showed that the use of markup standards more than quadrupled from the 2012

dataset to the 2013 dataset. However, a closer look at the data also showed that most entities were characterized by only very few properties, which in turn were rather generic.

When semantic data annotation standards such as Microformat, Microdata and RDFa are used, the underlying data can be easily extracted. There are libraries for these standards for many different programming languages to serialize and de-serialize data in and from the respective representation format. Thus, it can be said that the use of these standards greatly supports the process of data extraction and has already been adopted for longer by commercial search engines from Google, Yahoo! or Microsoft.

2.4.2 IE from Semi-structured Data

A huge amount of information on the Web is represented in a semi-structured form. "Semi-structured data is irregular data that may be incomplete and have a structure that changes rapidly or unpredictably but does not conform to a fixed or explicit schema" according to Eberendu et al. [24]. Common features include widely used structures such as tables, lists, and containers, which are also supported in HTML documents and are often mixed together. A major challenge in the extraction of semi-structured Web pages is that the layout structures differ very much across the Web and therefore it is very difficult to develop a single extraction approach for all Web pages. In research, there are already several different approaches to extract structured data from very diverse semi-structured documents. Most of these approaches require some form of human labeled data or manual configuration for each unique Web layout to extract relevant data. However, there are also some interesting approaches that are completely automated.

Since Web pages from various domains are very different in structure, *Wrappers* are often used in the IE research field [25, 26]. A wrapper is a procedure for extracting structured data from resources based on specific layouts. When working with HTML documents, a specific data-item of interest can be identified and subsequently extracted by defining a path from the root node of the underlying DOM tree to the target node which contains the relevant information, by successively extracting each node on the path. The difficult part of such an approach is the automatic creation of this path or wrapper. In the following, three interesting approaches for extracting semi-structured data are looked at in more detail.

WHISK [27] is an IE machine learning system which learns IE rules for the extraction of semi-structured or unstructured data automatically. These rules are represented in the form of regular expressions that can extract single or multiple properties. WHISK requires a certain amount of human labeled training data and based on that information, implements a supervised learning approach.

Another interesting approach called IEPAD [28], which stands for Information Extraction based on Pattern Discovery, is a system that discovers extraction patterns from Web pages without user-labeled examples. IEPAD only requires users to select record pattern

and information slots. In addition, IEPAD performs equally well in terms of extraction accuracy compared to previous work, but requires much less human intervention to produce an extractor.
Another approach for information extraction has been proposed by Carlson and Schafer [29]. This system demonstrates an approach for IE on a large number of semi-structured documents, by exploiting a tiny, fixed amount of human effort per domain of interest. The central idea is that the proposed system can bootstrap from a few training sites by building a model of data values and contexts for each schema column in the target domain. Subsequently, this model is applied to many more sites with no further human effort.

These approaches all have advantages and disadvantages. However, they have all been developed for a specific area of application and must be reconfigured or retrained for changed conditions. None of them offers a solution for extracting structured data without requiring configuration, as the structures containing the information can be very different.

2.4.3 IE from Unstructured Data

Unstructured data is information that does not have any particular format to indicate the underlying data structure. Most research in the context of information extraction on unstructured data deals with free text. Nevertheless, text does have a structure to some extent, but it is usually not explicit. The task of information extraction is to recognize this implicit semantic structure and thus to identify entities and relationships within them. IE is usually the result of several analysis tasks. Eberendu et al. [24] generalize IE from unstructured data into 5 different components: Named Entity recognition, syntactic analysis, coreference resolution, semantic analysis and cross-document coreference resolution. In the following, a brief overview of these are presented.

- Named Entity Recognition

  The task of named entity recognition is to recognize and classify entities within unstructured text. Some common names can be generally recognized (such as "Austria" or "Google"). It is more difficult to identify or classify common names that can refer to many different types of entities (people, places, organizations, etc.). It is often hard to make categorization decisions because the classification can depend on the term and its context. The entity type can usually be defined with the help of local features. For this purpose, the text surrounding the specific term can be analyzed as context together with the term itself.

- Syntactic Analysis

  Syntactic analysis is concerned with the general linguistic relationship between entities in each sentence. Therefore, a sentence is broken down into a sequence of chunks which can be divided into noun groups and verb groups. By applying syntactic regularization, relationships can be generalized and sentences with the

same meaning can be mapped together. For example, passive sentences are transformed into active ones, normalization is applied to verb forms, and different verbal compliments with the same meaning are classified together. By reducing the number of possible inputs, the training of semantic analysis can be greatly accelerated.

- Coreference Resolution

  Coreference resolution aims to generalize and standardize all occurrences of the same entities. Thus, for example, if a person appears more than once in a document but is referred to differently (e.g. "Alexander van der Bellen", "Van der Bellen", "Alexander", "He", "The president"), it should be linked to the same entity each time. For this, each variation is replaced with the most complete designation.

- Semantic Analysis

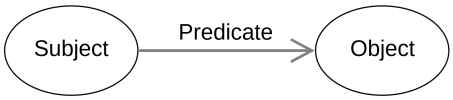

Figure 2.2: The Basic Semantic Triple Model[1].

  Semantic analysis creates statements based on all the previous processes. These statements are often encoded using RDF triples (see Figure 2.2). These RDF triples consist of a subject, a predicate, and an object. Entities can appear in different contexts as subject or object respectively. The sentence "Alexander van der Bellen won the 2016 Austrian presidential election" is thus transformed into the triple:

  - Subject: Person (Alexander van der Bellen)
  - Predicate: won
  - Object: Election (2016 Austrian presidential election)

  This process creates knowledge that can be stored consistently in a database, which in turn can be queried and verified.

- Cross-Document Coreference Resolution

  To perform IE on a large collection of documents with unstructured data, it takes more than just looking at each document individually. Single entities can be labeled differently by different authors. At the same time, different entities may have the same term in separate documents. Therefore, an IE system needs to resolve these cross-document coreferences in order to create a coherent knowledge base.

[1]Source: `https://commons.wikimedia.org/wiki/File:Basic_RDF_Graph.svg`

These five steps are carried out by almost all information extraction approaches for unstructured data in one form or another. In the scientific literature, there are already various promising approaches towards information extraction from unstructured data or free text. Two interesting systems are presented as a next step.

SystemT

The IBM Almaden Research Center developed a system called SystemT [30] for the extraction of unstructured text. They have built a declarative system that represented a paradigm shift for rule-based information extraction systems. To define the rules for the extraction process, they have developed a declarative language called *AQL* (Annotation Query Language). After an automated optimization step, a processing plan is created on the basis of this declarative language, which facilitates efficient information extraction. This allows SystemT to efficiently perform complex extraction tasks on large document collections.

BERT [31]

Bidirectional Encoder Representations from Transformers (BERT) is a machine-learning technique to convert natural language (unstructured text) into queryable knowledge. This system was developed by Google. The language models developed with BERT capture relationships and contextual connections of words very well. The algorithm is available as open source and can be used for information extraction from text. BERT is based on transformer language models with a certain number of encoder and decoder layers. As usual for transformer models, BERT processes text input bidirectionally rather than sequentially from left to right. The bidirectionality is mapped by processing all words of an input sequence simultaneously. For this purpose, BERT uses artificial neural networks. An entire ecosystem of NLP algorithms and applications has evolved around BERT in recent years. Google itself has implemented BERT in Google Search. For example, Zhang et al. [32] tackle the challenge of IE from business documents with content in the form of natural language. This is achieved by formulating a sequence labeling task which is solved with a fine-tuned BERT system. By using simple models, this project demonstrates the achievement of reasonable accuracy as well as the power and flexibility of BERT.

2.5 Generalization

To make data from different content providers comparable, the entity related data must be available in a similar structure or in a common schema. A schema describes how the data and its structure can be interpreted. A schema can also provide to some extent aspects about the content and meaning of the data. There are many different schema languages for different application areas. Examples include:

- Structured Query Language (SQL) for relational schemas

- XML Schema Definition (XSD) and Document Type Definition (DTD) for XML document schemas
- Web Ontology Language (OWL) for ontologies

Identifying related properties between data entities using different schemas requires a detailed analysis of the provided information, which makes it very difficult to automatise. To complicate the matter further, schemas on the Web are often extremely purpose-oriented and can therefore vary widely from domain to domain. Similar property names do not necessarily indicate the same property. Moreover, property names can also be encoded or abbreviated in a special way. Often the entity properties only make sense in the context of the originating Web page. In order to match two or more datasets that use different schemas, a schema matching process must be applied. Therefore, either one schema must be transformed into the other (transformation) or the two schemas must be merged into a common new schema (integration). This process is generally referred to as schema matching. Do Hong Hai [33] examines different semi-automated schema matching approaches and identifies the main criteria for determining their effectiveness. The author demonstrates a generic modifiable schema matching system called *COMA++*, which is a highly flexible system for a wide range of application areas. Furthermore, a mapping-based approach called *GENMAPPER* is presented with the goal to integrate heterogeneous Web data sources. Therefore, similarities are identified and mapped together. The approach is evaluated in the field of the bioinformatics domain.
In the next section, a universal schema definition is presented which aims to find application in as many different domains as possible.

2.5.1 Schema.org

The most widely adopted general purpose schema definition on the Web is *Schema.org* [34]. Its goal is to provide a single schema across a wide range of topics. The driving factor in the design of Schema.org is to making it easier for webmasters to publish their data in a machine-friendly way [35]. The Schema.org initiative is a community project that has been steadily expanding since 2011. It was originally founded by three of the largest search engines in the world Google, Bing and Yahoo! and is sponsored by Google, Microsoft, Yahoo and Yandex.

Schema.org uses URLs as the identifier of entities and properties. Those URLs refer to Web pages from Schema.org which contain the entity or property description. Furthermore, associated properties, sub-entities and entities implementing the referenced property are listed. Additionally, examples of how to use those schema items are described. Schema.org defines 10 major entity types such as Action, BioChemEntity, CreativeWork, Event, Intangible, medicalEntity, Organization, Person, Place, Product and Taxon. The description provides an even more detailed categorization by defining numerous sub-types. The properties of these entities are associated with specific data types like Boolean, Date, DateTime, Number, Text and Time.

Although Schema.org already covers a very large set of topics in significant depth, there is still a number of specific information entities and associated properties that have not yet been defined [[36], [37], [38]]. However, it is probably by far the largest project for a universal schema definition that exists.

3 Conceptual Foundation

In this chapter, the objective of this thesis is discussed in more detail. Firstly, the project topic is precisely delimited in order to identify the individual tasks which have to be performed. Subsequently, the particular requirements for the implementation of the prototype are listed in detail. We also discuss, how the system is being evaluated at the end of the thesis to see if the goals we have set ourselves have been achieved, and to assess the quality of the prototype.

3.1 Problem Definition

As mentioned above, the World Wide Web presents information primarily in human-readable form. Although there have been initiatives in recent years to increase the use of machine-readable annotations on Web pages, they are often implemented insufficiently or not at all. Furthermore, there are several standards for this purpose which are used for different application areas. As a consequence, extracting information in a structured form from a multitude of different Web pages becomes impossible, when only using a single information extracting approach, due to the heterogeneous representations of information on the Web. In order to process a large number of different Web documents on the internet, a flexible system using different methods is needed to extract structured information. In addition, this system must be highly configurable in order to best respond to different information representations on the Web. The WWW is developing very fast and is highly dynamic. Therefore, it should be possible to easily extend the application area and integrate further processing techniques to the system. It is exactly these challenges which are approached within the scope of this project.

3.1.1 Context of this Project

This research project focuses on Web resources containing information that can be directly associated with a single entity object. Let us assume, for example, a HTML Web page accessible via the URL `http://example.com/flubber` contains information about the real-world entity, say the movie Flubber. The application developed for this research project should process this Web resource so it can derive a data object with the title Flubber classified as entity-type Movie, including properties that can be directly associated with this entity such as director, actors, publishing date, etc. Kalchgruber et al. [39] provide a basis for this project to the extent that we deal with similar Web resources and have a primary objective in common, which is to make similar data on the Web comparable.

3.1.2 Relevant Content Representations of Web Resources

There is a great number of different resources available on the Web, such as HTML, XML, PDF-, or TXT files. Every file type uses different encodings and representations for the provided content. Therefore, customized processing and analyzing techniques are required because their representation differs too much for a generic approach. Due to this circumstance, it would go beyond the scope of this research project to implement a Content Mining solution for the numerous different types of documents available on the Web. This research project focuses on HTML Web pages since the vast majority of the dynamic Web is comprised of them.
A HTML Web page can consist of many different elements, which encode information using various representations. Subsequently, several data formats which are important to be distinguished to identify and extract information from HTML documents are discussed in this section.

Plain Text

One of the most central data formats is *Plain Text* which we define as a specific part or area on a Web page that does not include any additional structure than the raw text. However, due to the structure of HTML documents, every Web page naturally inherits some structure. The term Plain Text stands for a certain text area on Web documents with no further structure like tables or lists. Plain Text can include presentational tags which change the weight, font, or size of the text. One way to extract structured data from Plain Text is by using Natural Language Processing (NLP) tools for entity recognition and entity linkage, which maps mentions of proper nouns to the corresponding entities as discussed in 2.4.3.

Data Encoded in HTML Structures

The crucial concern of the project is the extraction of structured data from HTML Web resources. Therefore, the underlying HTML DOM [19] tree can be traversed, with the purpose of finding specific patterns to further process resources into structured data records. As discussed in 2.4.2, we refer to data encoded in HTML structures as semi-structured data. The structures of HTML encoded data vary strongly across the internet as there is no commonly used standard for representing such information. Different content providers use different structures according to a broad range of aspects like visual appearance, functionality, technology, underlying frameworks or stylesheets. Information can be encoded in HTML lists or tables as well as in any other structural HTML elements or any compound of different elements. To identify and process those differing representations, selecting adaptive approaches according to the used structure is key.

Multimedia Data

Due to the decreasing costs of digital storage and the improvements in the telecommunication network in the last decade, a growing amount of multimedia data like images, videos, and audio files are published on Web pages. Multimedia resources can provide a lot of information which is interesting for Web content mining. Although there are already advanced applications in the area of multimedia classification and feature extraction based on deep learning algorithms, research is still in an early state.

RDFa and Microdata

In recent years, a rising number of websites embed structured data into their HTML pages using markup standards such as RDFa or Microdata (see 2.4.1). These markup formats enrich HTML Web pages with annotations which support Web applications to understand the content of Web pages. For this purpose, specific attributes are added to HTML elements that describe the published information.

JSON-LD

Another possibility to collect structured data about specific Web resources is to make use of the JSON Linked Data (JSON-LD). This is a lightweight data format for providing machine-readable information about web resources. JSON-LD is, similarly to RDFa, a serialization format of RDF. The adoption of JSON-LD is a way to create a network of standards-based, machine-readable data across websites. Some websites add this data to their HTML web pages to enable third-party applications to facilitate automatic content processing. The processing of this type of data is very simple due to the usage of a uniform schema. Some content providers like `moviepilot.de` offer the complete information structure of entity records on their website, formatted in the JSON-LD schema allowing third-party applications to easily access and process underlying data.

All the previously discussed data representations contain potentially relevant information for our information extraction purposes. The implementation of custom-fit extraction methods for every defined data representation would go beyond the scope of this thesis. Therefore, we will subsequently only consider the extraction of data encoded in specific HTML structures, as well as the data representation formats RDFa, Microdata, and JSON-LD. We are not going to further develop a solution for multimedia data or Plain Text. Nevertheless, the design of the resulting framework should allow future adaptations to add this functionality to the system.

3.2 Requirements

In this section, we list all functional and non-functional requirements the developed system has to fulfill. In addition, general requirements that are important for modern application development are addressed.

3.2.1 Functional Requirements

1. **Fetching of Web Resources**

   The system which will be presented in the course of this paper should be able to fetch Web resources, in particular HTML Web pages, when given an URL of the concerning document. It should be noted that many websites prohibit or restrict automated crawling. This circumstance should be taken into account accordingly by providing the option to configure such individual requirements.

2. **Process Documents to Obtain Structured Data**

   The fetched documents should be processed by using one or several different processing steps to extract information about properties that are directly related to the Web Entity. The application should at least be able to process the following data formats:

   - **JSON-LD Records**
     The system should be able to identify and extract JSON-LD data provided in HTML documents. The information should be included in the resulting data object if the published properties are relevant to the concerned entity record.

   - **Property-Value Tables**
     The system should be capable of identifying and processing two-dimensional HTML tables that reveal information about entity related properties. The application identifies the exact location of the property names and associated values and extracts them appropriately.

   - **HTML Listings**
     The application should be able to process certain HTML listings that show values for entity-related properties. The lists can be created using the HTML tag "li" but also by using other repetitive HTML elements, as long as they have a consistent structure.

   - **RDFa and Microdata**
     The application should be able to extract structured data from HTML Web

pages where markup standards like RDFa or Microdata are used. The embedded annotations should indicate to which properties the published data can be mapped.

3. **Property Mapping**

   The application should provide the possibility to automatically map extracted properties to specific concepts to improve the comparability of the data for further applications.

4. **Resource Crawling**

   The application should be able to independently identify relevant HTML pages on relevant domains according to some configuration and independently initiate the extraction process for these resources.

5. **Explicit Extraction Request**

   In addition to the automatic identification and extraction of Web pages, it should also be possible to explicitly extract specific pages. Thus, it should be possible to answer individual processing queries in addition to automatic crawling.

6. **Data Management**

   The application should persistently store the acquired data in a consistent format and automatically update the data if it is considered outdated.

7. **Frontend - Dashboard**

   The application should contain a user-friendly dashboard which can be used to configure customized process workflows for different resources. The dashboard should also provide the possibility to configure the procedure of crawling resources from specified domains.

8. **Configurable Process Flow**

   Concerning the processing of a specific Web resource, users can configure the selection of involved modules and the order of their execution in regard to the source domain and the data type without a higher understanding of the underlying system.

9. **Error logging**

   The system should display errors or bugs occurring during the extraction process on the front end dashboard in order to comprehend missing or incorrect data.

3.2.2 Nonfunctional Requirements

In this section, non-functional requirements that are essential for a modern Web application are discussed.

1. **Usability**
    - It is possible to access the configuration dashboard via every modern browser.
    - The system can be easily configured in order to handle new Web resources
    - Special setting options in the resource process configuration are annotated with additional explanations
    - Interactive elements follow standard design guidelines commonly used in other Web applications.

2. **Virtualization**

    State of the art Web application development needs to consider an adequate hosting environment. Therefore, it should be possible that the implemented system can be deployed to the Microsoft Azure cloud to ensure the usage of an appropriate underlying hardware and improve the management and scalability of the application.

3. **Programming Language**

    Python is one of the fastest growing major programming languages [40], with a large open-source developer community. One reason for Pythons popularity is its excellent readability. This makes it easier for developers to produce understandable code which is important for readability and further documentation. Due to its highly extensible modular structure and the great availability of special interest functionality, it is the perfect programming language for major parts of this project.

4. **Web Framework**

    Software frameworks help developers to build applications efficiently by providing a set of standardized libraries, tools for programming, and support a clear organisation of resources. Therefore, it is reasonable to use an appropriate Web framework for the configuration and for configuration and visualization of the application workflow.

3.3 Evaluation Design

The system should achieve the previously defined goals and requirements as well as possible. Therefore, the system's flexibility is tested using freely available data representations as input. Furthermore, its durability is assessed by processing a large number of resources automatically over a longer period of time.

3.3.1 Flexibility Evaluation

The main goal of this work is to develop a framework that can extract information from various Web pages and store it in a uniform structure. The system should not be designed for a specific domain of Web resources, but should be able to process a wide range of resources using different presentation standards and designs. The system should also be able to handle multiple different presentation formats used in a single Web resource to obtain a maximum of available data.

To evaluate this, we will extract data from a variety of information providers, which in turn use different data formats to present their information. Furthermore, we want to extract data sets from different subject areas.
In particular, we want to test whether our system has high flexibility by means of different resource inputs. Therefore, the system should be able to handle the following situations:

- The system will be tested with Web pages from at least five different content providers who provide their websites on five different domains.
- These Web resources cover datasets of at least three different entity types.
- The system will be exposed to annotation standards like RDFa and Microdata or the RDF serialization standard JSON-LD.
- The system will have to extract data from several different semi-structural HTML pages, which do not implement any machine-readable annotation standards.

Afterwards we want to inspect the extracted data and its quality. For this purpose we will analyze the records based on the following questions:

- Which information can we extract from the Web resources?
- What is the quality of the data in terms of completeness and comparability with similar data entities?
- Which data could not be extracted or was extracted incorrectly? Why is this happening and how could it be improved?

We also want to know to what extent the system design creates a flexible and modular framework, how far the system can be adapted to other application areas and whether the tool set can be extended without making major changes to the core system.

3.3.2 Durability Evaluation

We want to test the durability, reliability and independent workflow of the system by not only processing individual Web resources, but also by processing a larger number of resources automatically. This is crucial because the system must be able to autonomously process websites which publish gigantic amounts of information spread out over many individual Web pages. The system must therefore be able to solve the following sub-activities automatically:

- Identification of relevant Web pages on a particular domain
- Extraction of data from these Web pages
- Transforming the data into a uniform format
- Permanent storage of data in a persistent database

The entire process is tested by processing tens of thousands of Web pages. The result is then analyzed in detail to answer the following questions regarding performance and quality criteria:

- How many Web pages can be processed in a specified period of time?
- How high is the utilization of individual resources in the process with regard to storage capacities and the execution environment?
- What are the resource bottlenecks and how can performance be improved in terms of the resources and technologies used?

4 Technical Foundation

In this chapter, technical concepts and technologies are explained which are crucial for the implementation of the project. This also serves as a fundamental basis for further understanding this thesis.

4.1 Cloud Computing

In the last 10 to 20 years, cloud computing has experienced a real hype. Many software company giants such as Google, Amazon and Microsoft offer numerous Cloud Computing services for companies and private users. With a growing range of products and simplified resource management, Cloud Computing providers are driving the demise of many companies' self-managed on-premises server facilities. But what exactly do we mean by Cloud Computing?
The *National Institute of Standards and Technology* (NIST) defines Cloud Computing as

> "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models" [41].

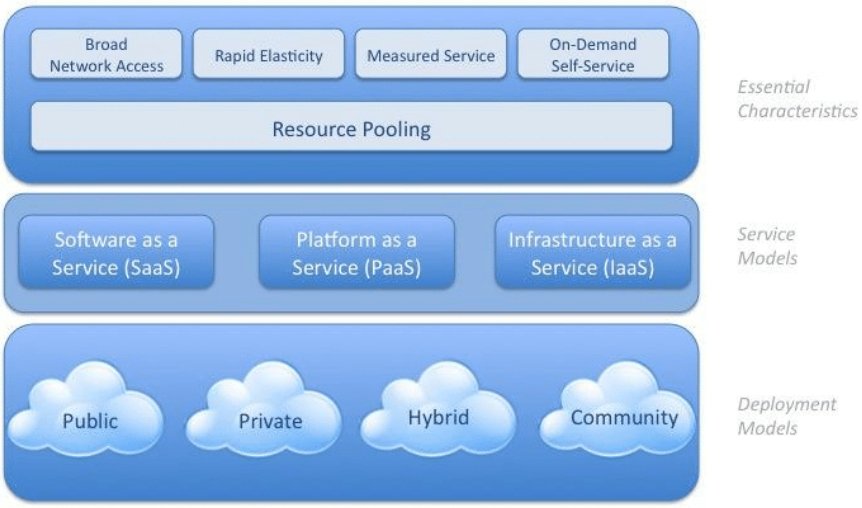

Figure 4.1: NIST Visual Model of Cloud Computing as Service Models [1]

The NIST defines these five essential characteristics as following:

1. *On-demand self-service*
   Cloud computing customers can manage their own computing capacities without the intervention of a service provider employee.

2. *Broad network access*
   Customers can interact with the offered capabilities via the network by means of a standardized common mechanism which promote use by heterogeneous complex or simple client platforms.

3. *Resource pooling*
   The service provider combines its capacities into pools and uses these to meet customers' needs for virtual resources. Allocation is generally independent of location.

4. *Rapid elasticity*
   Resources can be used and released quickly and automatically. For the customer, the resources are infinitely available and can be purchased at any time in any quantity.

5. *Measured service*
   Cloud systems automatically control and optimize resource usage by measuring parameters that depend on the service in question such as data storage, computing capacity, bandwidth or number of active users. Resource usage can thus be monitored, controlled, and transparently viewed both by the service provider and the customer.

[1] Source: `https://www.researchgate.net/figure/NIST-Visual-Model-of-Cloud-Computing-a-Service-Models-Cloud-computing-can-be-classified_fig2_241195178`

Depending on the provided IT resources, a distinction is made between different service models:

1. *Software as a Service (SaaS)*
   Software as a Service refers to the provision of software via the internet. The user does not have to install, operate or maintain the software. Access can be independent of location and time. Additionally, various end devices can be used, such as a browser or a program interface. The providers of SaaS solutions not only provide the software itself but also guarantee its availability and the security of the data and applications.

2. *Platform as a Service (PaaS)*
   Customers can use the cloud infrastructure to deploy their applications, but can only use the tools provided by the service provider. The customers have no possibility to control the used cloud infrastructure (server, operating system, data storage, etc.), with the exception of certain configuration options provided by the service provider. Therefore, customers have control over their deployed applications and can make adjustments as they see fit.

3. *Infrastructure as a Service (IaaS)*
   The provider makes computing capacity, data storage, network and other basic infrastructure resources available, which can be used by the customer at will. Customers have no ability to influence the given cloud infrastructure, but have full control over operating systems, tools and deployed applications.

The cloud deployment model determines the specific nature of the cloud environment based on cloud ownership, scope, and access. The NIST differentiates between four different deployment models.

1. *Private cloud*
   The cloud infrastructure is provisioned for a single organisation. No other customer has access to the these resources.

2. *Community cloud*
   The cloud infrastructure is provisioned for a group of customers who share similar requirements, in terms of a common mission, security requirement, similar policy, or compliance considerations.

3. *Public cloud*
   The cloud infrastructure is provided for open use for the general public by the service provider.

4. *Hybrid cloud*
   The cloud infrastructure is a composition of two or more different cloud infrastructures (private, community or public), which remain independent entities but are interconnected by standardized or proprietary technology. This allows data and applications to be transferred between different clouds.

4.2 Web Services

The World Wide Web Consortium (W3C) defines a Web service as "a software system designed to support interoperable machine-to-machine interaction over a network. [...] Systems interact with the Web service in a manner prescribed by its description using SOAP [42] messages, typically conveyed using HTTP [43] with an XML [44] serialization in conjunction with other Web-related standards" [45]. A Web service is an abstract notion that must be implemented by a concrete agent. The agent is the application that sends and receives messages, while the service is the resource characterized by the abstract set of provided functions. Web services can be accessed via a unique URI. The client does not need to know virtually any implementation details of the Web service in order to use it. In this way, a distributed architecture can be created.

By using SOAP, a client can call methods of a server-side object using relatively complex XML messages. The combination of XML and HTTP made the whole protocol independent from programming languages and platforms. However, SOAP also has some key weaknesses according to Helmich [46]: the complicated XML messages have a rather large overhead containing a lot of metadata and the assembly and disassembly of the XML messages is intense in computational effort. In addition, there are now numerous SOAP sub-standards, which also makes the application much more complicated. A simple alternative for communication between services is the *REST architecture* [47], where interactions between components take the form of dynamically sized messages. Thus, small or medium sized messages are used for control semantics, whereas larger messages are only needed when accessing or sending resources.
Both SOAP and REST require interfaces called Web APIs to access and manipulate resources. These interfaces can also be used to exchange messages or data between distributed services which represent a common process. Thus, a workflow of logical processing steps performed by services can also be created by using these Web APIs.

4.3 Microservices

The name *Microservice Architecture* refers to a particular way of designing software applications. The main feature is that individual system components can be published independently of each other, in contrast to monolithic applications. Although there is no clear definition of microservice architecture, common characteristics of microservice applications can be identified, as Lewis and Fowler [48] have done.
A software application system that follows microservice architecture consists of individual components. These components are ideally kept very small and have a very specific task to fulfill. Each of them can be used independently of the others. Such independent components which fulfill a special task are refered to as services. Applications built from microservices aim to be as decoupled and as cohesive as possible. Therefore, the services communicate with each other, usually with mechanisms such as Web service requests or remote procedure calls, unlike in-memory function calls which are used in monolithic

systems. One consequence of dividing the system process into microservices is the need for precisely defined interfaces. This makes the process flow more transparent and improves joint work on a project. A disadvantage of the loose coupling of components and the corresponding communication mechanisms is that they take longer than conventional function calls.

The microservice approach divides an application into services organized by business capabilities. This makes it easier to distribute areas of responsibility in software development. Adjustments can be made more easily, as not the entire development team has to get involved. If we split the components of a monolith into services, we have more options when creating each service. We can freely decide which programming language, Web framework, database technology or libraries we want to use regardless of the technologies used in the other services. Furthermore, the individual components are easily interchangeable and updates can be made later with less effort.

4.4 Web framework

A software framework is a set of standardized libraries and tools for a programming language, to efficiently build a well-organised software application. A Web framework is used to simplify the development of dynamic websites, Web applications or services. The reuse of code and the clear organisation of resources is thereby promoted. Most Web frameworks offer database access, usually via an object-relational mapping. Often, an object-relational mapper simplifies the data management by taking care of the access and storage of the data. Many frameworks follow the *Model View Controller's* (MVC) architectural pattern to clearly divide the system into the data model, the business rules, and the user interface. This is generally considered good practice because it modularizes the code and promotes code reuse.
Two of the most famous Web frameworks for Python are *Django* and *Flask*. Ghimire [49] compares those two frameworks, and concludes that both offer minor advantages and disadvantages, but none of the systems is fundamentally better than the other. Therefore, both are very well-suited to promote the development of a well-structured Web application.

4.4.1 Django

Django is a high-level Python Web framework. It provides a set of tools that encourages rapid development and clean, pragmatic design. Django provides functions that solve many common problems related to Web development, such as security functions, database access, sessions, template processing or URL routing. Django is based on a MVC architecture. The typical workflow of a Web request can be seen in Figure 4.2.

As illustrated in Figure 4.2, users can request resources which usually are Web pages via the browser, for instance. A request to a Django Web application is initiated by calling a specific URL. This URL is matched against the defined URL paths in the urls.py

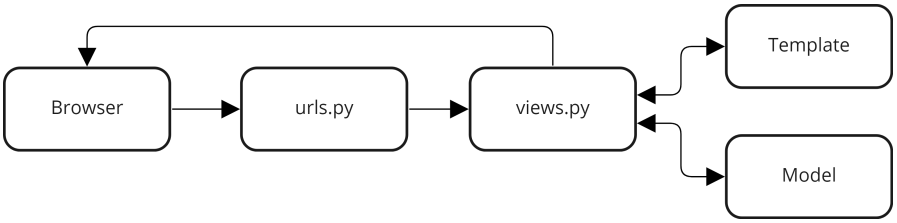

Figure 4.2: Typical Django Process Workflow

file. This process is called URL mapping.

Django calls a *View* associated with the requested URL. This View is defined in the views.py file. The triggered view can use data models relevant to the requested content, before returning a response to the user. The requested information can be represented in different ways. Typically, templates are used to create a reader-friendly presentation. These templates can consist of HTML code and static files such as CSS and Javascript files or resources like images and videos.

4.5 NoSQL Databases

NoSQL stands for "not only SQL (Structured Query Language)" and refers to databases that follow a non-relational approach and thus break with the long history of relational databases. In contrast to SQL databases, where data is stored in rows and columns, NoSQL databases can be based on completely different data structures, which have different advantages and disadvantages depending on the area of application. Generally, in addition to the classic relational databases, we can distinguish four common types of NoSQL databases. The following classification is based on the discussion of Strauch et al. [50] and the NoSQL documentation from Microsoft [51] and MongoDB [52].

- *Key-value stores*
  A Key-value store is a simple database that uses an associative array as the fundamental data model. Each individual element is stored in the database as an attribute name, also referred to as key, along with its value. Key-value databases are best used when the key is known and the value associated with the key is unknown.

- *Graph databases*
  Graph databases use a model based on nodes and edges that represent connected data, similar to the relationships between people in a social network. They allow easy storage and navigation of highly interconnected relationships. One of the best known applications of graph databases is the *Resource Description Framework* (RDF) [8]

- *Column stores*
  Column-oriented databases store data tables by columns and not by rows like relational databases. They are optimized for searching large data sets and store all

values together in one column. This makes data access much more efficient, but they are usually less efficient at inserting new data.

- *Document stores*
  Document databases combine each key with a complex data structure called a document. Each document can contain many different key-value pairs and even embedded documents. This allows complex data structures to be stored in the database and queried based on specific properties.

4.5.1 Advantages of NoSQL

NoSQL databases help developers meet new challenges due to increasingly diverse data types and models. They are also extremely effective at working with unpredictable data. The data access is thereby often extremely fast. Unlike the relational model, its data model is designed to address the following challenges:

- Large volumes of rapidly changing structured, semi-structured and unstructured data
- Agile development and frequent schema changes
- Flexible object-oriented programming
- Geographically distributed and scaleable architecture instead of monolithic architectures

One of the most important features of NoSQL databases is the use of dynamic data schemas. In relational databases, schemas must be precisely defined upfront and subsequent changes often require significant effort. However, in agile development, it is often necessary to change the schema of the database afterwards. Moreover, in a relational database, it is not possible to process completely unstructured data or data with a data type that is not known in advance. In contrast to this, NoSQL databases allow the insertion of data without a predefined schema. As a result, applications can be easily changed in real time without interrupting operations. This speeds up development, simplifies the integration of new software and reduces the time spent on database management.

4.6 Microsoft Azure

Azure is a cloud computing platform from Microsoft, that provides support for public, private, and hybrid clouds. Microsoft offers the full range of cloud computing service models. IaaS with full control over servers, storage and network elements. PaaS which additionally offers middleware, development tools, business intelligence services, database management systems and more. As well as SaaS such as *Microsoft Office 365*.
Azure leverages large-scale virtualization in many Microsoft data centers around the world, with plans to increase capacity in the foreseeable future by building 50 to 100 new data centers per year [53]. Azure offers more than 600 different services [54]. We will take

a closer look at some of these services, which are relevant for the implementation of this project.

4.6.1 Azure Service Bus

The *Azure Service Bus* [55] is a fully managed message broker for cloud services. The service bus is used to decouple applications and services from each other. Thereby, data is exchanged between service applications in the form of messages. The message content can be either plain text or encoded in a common data format such as JSON or XML. The Message Bus offers two different concepts to send and receive messages:

1. Queues
   With service bus queues (see Figure 4.3), sent messages are stored in queues until

Figure 4.3: Azure Service Bus Queue Model

   they are received by a corresponding process. The *first-in-first-out* (FIFO) principle applies. Only one message consumer receives a particular message in the queue. Therefore, consumers compete with each other. A major advantage of message queues is load balancing, which allows producers and consumers to send and receive messages at different rates. The intermediation of messages between producers and consumers through a queue results in the consuming application only having to handle the average load and not the peak load. Since the producer and consumer are unaware of each other, a consumer can be changed or replaced without affecting the producer.

2. Topics and Subscriptions

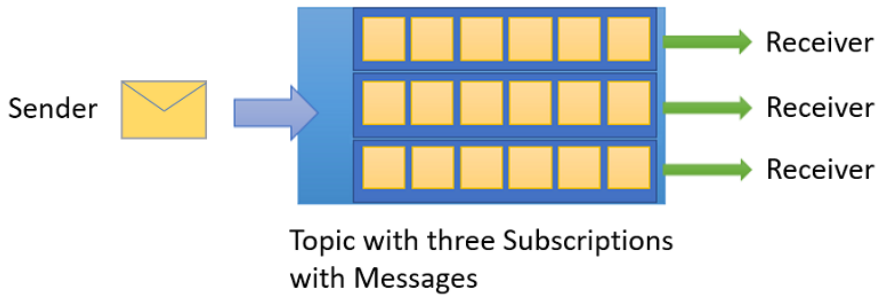

Figure 4.4: Azure Service Bus Topics Model

As an alternative to queues, topics (see Figure 4.4) can be used to send and receive messages. Topics implement a publish/subscribe scenario. They can have multiple, independent subscriptions attached to each channel. Subscriptions can also be configured so that messages expire after a certain time. Apart from that, they work similarly to queues from the receiving side.

An important configuration option of Service Bus Messages are sessions. Any producer can create a session when sending messages to a topic or queue by defining a unique identifier for the session ID property of a message. This can guarantee a FIFO principle in the service bus. Furthermore, a Request-Reply pattern (see 4.5), which makes a two-way conversation possible, can be created using the messaging system.

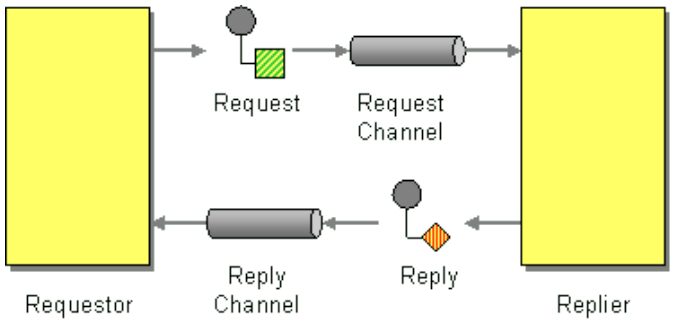

Figure 4.5: Request-Reply Messaging Pattern [2]

4.6.2 Azure Cosmos DB

Azure Cosmos DB [56] is a fully managed NoSQL database for modern app development. It has the ability to automatically and instantly scale the storage capacity and the data throughput of the database. Therefore, the data-store can be adjusted to changing requirements at any time. A Cosmos database consists of multiple containers in which items can be stored. Items within a container can use different schemas and also be of different entity types. The records are required only to have a single id property value for each logical element. They can be uniquely identified with this id. The data is stored on one or more servers called partitions. In order to increase throughput or storage space, additional partitions can be added. When creating a database container, we have the possibility to configure the data throughput in two different ways:

- Dedicated throughput
  The throughput provisioned for a container is reserved exclusively for that container.
- Shared throughput
  The throughput is set at the database level and then shared with up to 25 containers within the database.

[2] Source: `https://www.enterpriseintegrationpatterns.com/patterns/messaging/RequestReply.html`

Azure Cosmos DB provides several database APIs, including the Core (SQL) API, API for MongoDB, Cassandra API, Gremlin API, and Table API. These APIs can be used to model real-world data using documents, key-value, graph, and column data models. This allows us to treat the Azure Cosmos DB as if it is based on one of those database technologies. These APIs can be used to benefit from the tools and capabilities that each data model brings to the table.

API for MongoDB

With the Azure Cosmos DB MongoDB [57] API, a Cosmos DB can be accessed as if it would be a pure Mongo database. The API implements the Wire protocol [58] for MongoDB which is a simple socket-based, request-response style protocol. Clients communicate with the database server through a regular TCP/IP socket. Cosmos DB stores data in a document structure based on the BSON format. BSON is a binary representation of JSON documents, though it contains more data types than JSON. Figure 4.6 shows an example of a typical BSON document. The Azure Cosmos DB API for MongoDB implementation enables transparent compatibility with native MongoDB client SDKs, drivers and tools. Thus, it is possible to connect to the database with any MongoDB client driver that is compatible with the API version used.

Figure 4.6: Exemplary Structure of a BSON Document `https://www.mongodb.com/docs/manual/core/data-modeling-introduction/`

4.6.3 Azure Functions

Azure Functions [59] is a serverless compute service that enables users to run event-triggered code without having to provision or manage infrastructure. With Azure Functions, the logic of the system can be implemented in easily readable blocks of code, which are called "functions". Functions can be executed at any time, if explicitly requested or independently in response to certain events. Azure Functions can be used to achieve decoupling, high throughput, and code reusability. This makes it suitable for creating a microservice architecture. When a function's demand increases, multiple instances

with associated resources can be dynamically created to handle the higher demand. When the load decreases again, all additional resources and application instances are automatically switched off. Consequently, only as much resources as necessary are used. On-demand provisioning of compute resources is at the core of serverless computing in Azure Functions.

Design Principles for Azure Functions

In order to improve the performance and reliability of Azure functions, various best practices can be applied.

- *Avoid long running functions*
  The larger an Azure function is, the higher the risk of unexpected timeouts. Furthermore, only the necessary dependencies should be included per function, as this can often increase the loading time considerably. Large functions should be subdivided into smaller collaborating functions whenever possible.

- *Write functions to be stateless*
  Functions should be stateless if possible. It makes sense to combine the required data with the associated state information.

- *Write defensive functions*
  Azure functions should be programmed defensively. Thereby, it is not necessarily about the requirement that no error may occur in a function, but rather about that, if an error occurs, it is handled in a particular specified way. As with all kinds of defensive programming, avoiding bugs is a primary objective; however, the motivation is not as much to reduce the likelihood of failure in normal operation, but to reduce the attack surface. Programmers must assume that the software might be misused actively to reveal bugs, and that bugs could be exploited maliciously.

- *Use multiple worker processes*
  By default, one worker process or also called thread is created for a function. To improve performance, the number of threads can be increased. Azure Functions attempts to distribute the concurrent function calls evenly across the selected number of threads.

Triggers and Bindings

Triggers cause a function to execute. Each function can be called by exactly one type of trigger. These triggers are associated with data which is often provided as a payload to the function. A binding to a function is a way to declaratively associate other resources with functions. Azure offers a variety of different ways to trigger a function. The trigger of a function can be an event such as an insertion, modification or deletion within an Azure database. Functions can also be triggered by events delivered to an Azure Event Hub [60] or a subscription in an Azure Event Grid [61]. There are many more ways to

Interval format	example (every 5 minutes)	description
NCrontab	0 */5 * * * *	{sec} {min} {hour} {day} {month} {weekday}
Timespan	00:05:00	hh:mm:ss

Table 4.1: NCrontab and Timespan Expression Example

define triggers and bindings for Azure functions; however, we will take a closer look at three different mechanisms.

- Timer Trigger [62]
  The Timer trigger for Azure Functions makes it possible to execute a function automatically at regular time intervals. Therefore the time between the function calls is defined in the function settings. The duration can be set by using a NCrontab [63] expression, which is similar to a CRON expression except that it includes an additional sixth field at the beginning which is used for defining the time precision in seconds. Alternatively, a Timespan expression can also be used. An exemplary definition of such a time interval expression can be seen on the Table 4.1. For functions triggered by a timer, only one instance is executed at a time. They will not trigger again if a pending invocation is still running.

- HTTP Triggers [64]
  HTTP triggered functions can be used to quickly create custom serverless APIs. The input type of an Azure Function can be defined as an HttpRequest. Thereby, the entire request object can be accessed, including the JSON formatted request body, in order to receive custom data at the function interface. In addition to the request payload, parameters declared in a function's endpoint route can also be accessed. Functions always end with the return of an HttpResponse object, the default response contains the status "HTTP 204 No Content" and an empty request body. Azure offers several ways to restrict access to HTTP triggered functions by the introduction of access keys, which facilitate three different authorization levels:
  - *anonymous*: No access key is required; everyone can call the function.
  - *function*: A function-specific API key is required to call the function.
  - *admin*: The admin-level host key is required to call the function.

  In addition to these basic access restrictions, it is of course also possible to use extended authentication mechanisms. Azure functions support the ability to use third-party identity providers to determine the identity of the requestor without much implementation effort for instance. In addition, it is also possible to implement a self-developed authentication process.

- Service Bus Trigger [65]
  A service bus trigger will execute an Azure function after receiving a message from a service bus queue or topic. The service bus message is passed to the function either as a String or as a JSON object. To connect a service bus queue or topic to

an Azure function, the connection String of the service bus must be specified in the function definition. When using a queue, the queue name must also be declared. If a topic is used instead, the subscription name must be declared in addition to the topic name. All the management around the message delivery is handled by the Azure services, which enables programmers to easily send and receive messages between loosely coupled services.

Function App

Each Azure function is embedded in an execution context called a function app. All functions in a common execution context must be written in the same programming language. The functions share common settings in this environment and all have access to the same environment variables. Individual functions in a function app are deployed and scaled together.

4.6.4 Azure SQL Database

The Azure SQL Database [66] is based on a Microsoft SQL Server database engine and is fully managed by Microsoft (PaaS). This means that developers do not have to deal with upgrades, patches or backups. Data stores can be allocated for a wide variety of applications, using the Azure SQL Database service. The provided resources, such as storage capacity and data throughput, can be scaled at any time to meet changing requirements.
Azure provides two different deployment models for SQL databases. The first option is a fully managed single database. This is the most straightforward option to use when you need a single database for your cloud services. The second option is the use of a so-called *elastic pool* which enables you to host several databases, which share dedicated resources such as performance and data storage capacities, on a single virtual server. This option is advantageous if you have unpredictable usage requirements. In addition, the price performance for a group of databases within a given budget can sometimes be better maintained with elastic pools, as overprovisioning of resources based on usage peaks can thus be better balanced. Elastic pools can ensure that databases get the performance resources they need at the right time. They provide a simple mechanism for resource allocation within a predictable budget.

4.6.5 Azure App Service

Azure App Service [67] is an HTTP-based service for hosting Web applications. Azure App Service provides support for applications and Web frameworks programmed in ASP.NET, ASP.NET Core, Java, Ruby, Node.js, PHP, or Python. A Variety of features to automatically scale Web applications, set up continuous integration and deployment, patch and maintain the OS and language frameworks, are provided by Azure for developers. Azure App Service can be hosted in a Microsoft or Linux server environment. Since applications with Python 3.7 and higher are supported, it is possible to quickly migrate

or redeploy Web applications such as applications which are based on the previously discussed Web framework Django.

5 Design and Implementation

In this chapter, a prototypical implementation of a general purpose framework which fulfills the requirements described in Chapter 3, is presented. To achieve this goal, the technologies explained in Chapter 4 are used.
First, the back-end, which performs the main task of information extraction, is discussed. We introduce the used technologies and the different components which interact with each other. Afterwards, the front-end, which is responsible for the configuration of the workflow of the back-end, is explained.

5.1 Back-End

The back-end is the heart of the information extraction system. It takes care of the whole process of content extraction from Web pages and offers the possibility to start this process via public APIs. It also contains a Web crawling component that allows us to automatically find Web pages and initiate the extraction process for them. All system parts are based on Azure cloud services to obtain an adequate hosting environment with the possibility to dynamically allocate additional resources for varying computational demand.
The system is designed according to a microservice architecture. Therefore, it consists of many small services, each performing a core task. The communication between the services is done either via well-defined interfaces or via a fully managed messaging system. By choosing this architecture design, individual services can be modified, replaced or added without the need to necessarily adjust other related components. Figure 5.1 shows the implemented components and the communication flows between them. The upper part of the figure shows the ExtractionAPI, URLCrawlerAPI, URLCrawler, DataAvailabilityChecker and the ExtractionProcessController. These services implement the core functionality of the system. In addition, the messaging system, called Service Bus, is immediately recognisable. With the help of two service bus topics, messages are exchanged between services. At the right edge of the illustration you can see the databases in which the final data objects and settings for the data processing are stored. Finally, in the lower part of the figure we see the resource extraction and modification services. These functions are called for the specific extraction of the different data representations.

5.1.1 Cloud Services Used

A total of 4 different cloud service technologies are used for the back-end system.

1. *Azure Functions*:

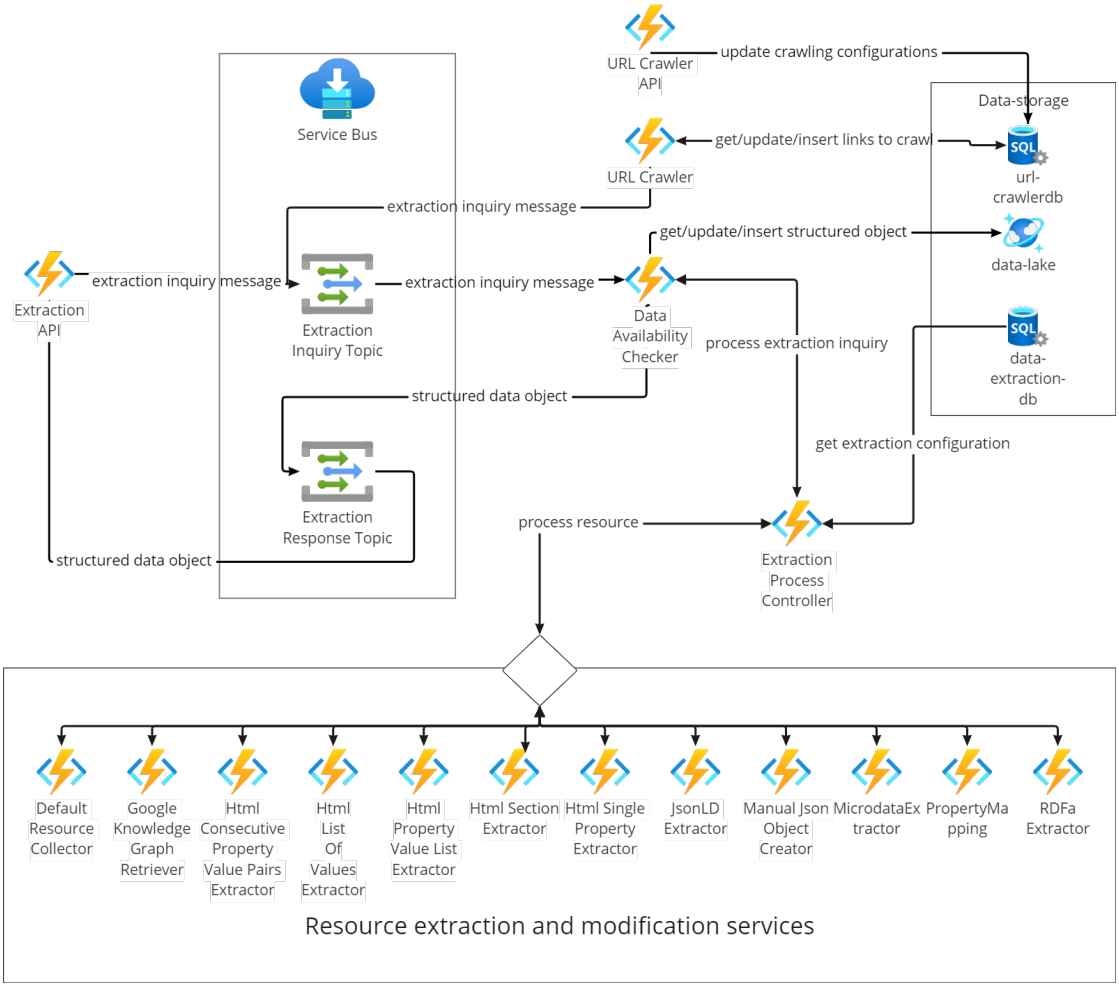

Figure 5.1: Component Diagram of the Back-end

All logical processes are implemented using Azure Functions. They are all executed in a common function app, i.e. a common execution environment and are therefore deployed together. The functions run in a Linux environment and are developed in Python. We use a total of three different trigger mechanisms for the implemented services:

- *Timer trigger*: The UrlCrawler function is triggered by a timer trigger which starts the service at regular intervals, usually several times per minute.
- *Service Bus trigger*: The DataAvailabilityChecker service is triggered by messages sent to the extraction inquiry topic on the Azure service bus. The ExtractionAPI also uses this Service Bus, however, it is not triggered by the Extraction Response Topic, but the service waits during runtime for a specific

message from this service bus topic.

- *HTTP trigger*: All other services, including the ExtractionAPI, are triggered by an HTTP trigger and return an HTTP response to the requestor. The function endpoint is composed of the base URL of the common function app environment and the name of the respective function. Thus, for example, the ExtractionApi can be addressed via the endpoint: `https://jr-functionapp.azurewebsites.net/api/HttpTrigger_ExtractionApi`. All HTTP triggered functions expect certain JSON serialized data in the request payload to respond to a wide variety of scenarios.

An important point regarding the efficiency of Azure Functions is that after about 20 minutes of inactivity of a function, all associated resources are deallocated. After that, a so-called cold start is performed, which significantly increases the execution time. If a function is executed more regularly, for example every few minutes, static resources do not have to be reloaded. Therefore, we have developed the services to use static parameters wherever possible, for example when using database or service bus connectors.

2. *Azure Service Bus*
   The Azure Service Bus is used to send messages between services. This message broker is essential for the decoupling of the UrlCrawler service and the general extraction process worklflow. The UrlCrawler sends the identified Web page URLs to the Extraction Inquiry topic on the Service Bus and does not have to wait for a response. This completely detaches the crawling process from the actual information extraction process. However, to get the correct data object from an individual request via the ExtractionAPI, an additional topic, namely the Extraction Response topic, is needed to send the extracted data back to the correct requester. The association of the message to the correct requester can be done by the definition of a session parameter on the message. Thereby a Request-Reply messaging pattern can be implemented.
   To restrict the use of the functions, a Function app authorization key is used for all Azure Functions which have been implemented in this project. This prevents unauthorized access to the services. However, this does not replace a complete security concept, but rather provides an appropriate access restriction for a prototypical implementation.

3. *Azure Cosmos DB*
   An Azure Cosmos DB is used to persistently store extracted data. We access the database in our services by using the MongoDB API (see Section 4.6.2). In the collection called data-lake, the data related to a specific Web page, identified by a unique URL, is stored as JSON serialized document which is compatible with MongoDB's BSON documents. Since the structure of the data can be unpredictable and complex, this document store is a perfect fit. If we were using a relational database, the structure of our records would need to be predictable from the outset, which is why we do not use this common technology for such data. In addition,

the Cosmos DB is very well-suited for very large data volumes and can handle this data very well in terms of performance.

4. *Azure SQL Server*
   Multiple services access an Azure SQL database to solve different tasks. Two different databases were created for this purpose. In the *url-crawler-db*, information about already visited and not yet visited websites is stored by the URLCrawler service. Additionally, the behavioural settings of the Crawler is stored. In the *data-extraction-db* it is specified which extraction and modification methods will be used for each individually defined extraction process. The database also stores configurations for the property mapping and log messages which are created during the extraction process.
   Both databases are deployed in an shared elastic pool in order to limit the prices for hosting multiple databases. It would also have been possible to put all tables into a common database, but we wanted to couple the URLCrawler and the rest of the system as little as possible and therefore decided for two seperate databases. The chosen design makes it possible to upgrade the system by introducing new databases without having to adjust the required budget.

5.1.2 Core-Services Description

The back-end consists of 17 different microservices implemented as Azure Functions. In this chapter the individual services and the communication between them will be explained in detail.

URL Crawler API (full name: HttpTrigger_UrlCrawler)

The URLCrawlerAPI service is responsible for adding new configurations to the url-crawlerdb database, as well as for initiating the crawling process on new Web domains. It is also possible to configure the crawling process via the front-end dashboard or by directly manipulating the database entries; this service serves as an alternative to that. The function responds to an HTTP request and accordingly manipulates data from two tables in the url-crawlerdb database.

links_to_crawl		
PK	id	INT
Unique	link	VARCHAR
	http_response_code	VARCHAR
	http_error_msg	VARCHAR
	last_time_crawled	DATETIME

Table 5.1: links_to_crawl Database Table

Database Description The *links_to_crawl* database table (see Table 5.1) contains all URLs or links that still need to be visited by the URLCrawler service as well as all those that have already been visited. A not yet visited link can be recognized by having only the link property defined, the http_response_code, http_error_msg and last_time_crawled are NULL values in this case. As soon as a link has been processed by visiting the respective Web page, the HTTP response code and if an error occurs, the corresponding HTTP error message is added to the database for the respective entry. The time of processing is also recorded.

crawling_configuration		
PK	id	INT
Not Null	netloc	VARCHAR
	extraction_pattern	VARCHAR
	link_discovery_pattern	VARCHAR

Table 5.2: crawling_configuration Database Table

The *crawling_configuration* table (see Table 5.2) contains processing instructions for the URLCrawler. The netloc property contains the network location of a URL which includes the respective domain and if available also the subdomain. The extraction_pattern property contains a regular expression which is used to check which URLs are relevant to initiate an information extraction process. The link_discovery_pattern property also contains a regular expression which is used to decide which forwarding URLs on a website are significant for the search of further relevant Web pages.

Title	Create Crawling Configuration		
URL	api/HttpTrigger_UrlCrawler		
Method	POST		
Parameters	urls	Array of strings	Optional
	netloc	String	Optional
	link_discovery_pattern	String	Optional
	extraction_pattern	String	Optional
Success	200 Success Message		
Error	400 link_discovery_pattern parameter is no valid regular expression		
	400 extraction_pattern parameter is no valid regular expression		

Table 5.3: Description of the URLCrawler API Endpoint.

Service Description The URLCrawlerAPI accepts four different parameters (see Table 5.3). The urls parameter is used to specify a list of links that will be added to the links_to_crawl table which will then be visited by the URLCrawler. To create a new crawling_configuration the netloc parameter must be specified. However, you can also submit the entire URL of a relevant Web page, the service then extracts the network

location from the URL by itself. The extraction_pattern and the link_discovery_pattern can also be specified when creating a new configuration. These parameters have to be correct regular expressions according to the Python library *re* [1]. If one of these two parameters is not specified, the default value "/*" is used instead, which considers all possible links as valid.
The service then adds all the newly specified URLs to the *links_to_crawl* database table and creates a new entry in the *crawling_configuration* database table using the parameters netloc, link_discovery_pattern and extraction_pattern.

URL Crawler (full name: TimerTrigger_UrlCrawler)

The URLCrawler is responsible for finding relevant Web pages and to initiate the information extraction process automatically for the identified resources. The URL Crawler implements the functionality of a so-called Incremental Crawler that visits only specifc Web pages based on some configuration options. We did not develop a Focused Crawler because for our application we cannot distinguish between more relevant and less relevant Web pages. A Web page is either relevant or not, so a simpler Incremental Crawler is used. Figure 5.2 shows the sequence diagram of the URLCrawler service. The process is described in detail in the next paragraph.

Service Description The URLCrawler is a timer triggered service and is executed in regular intervals. The interval can be set in the functions.json file of the service. An interval of 10 seconds was selected when carrying out the experiments for this thesis. When deciding on the duration of the interval, the runtime of the URLCrawler itself was the main decisive factor. The subsequent processes can dynamically handle higher workloads. However, since the URLCrawler is only a single service instance, this process cannot be parallelized automatically and we have to be satisfied with a certain runtime. When the URLCrawler is called, the following process steps are executed in sequence:

1. The URLCrawler fetches a random batch of URLs from the links_to_crawl table in the url-crawler-db database (see Table 5.1) which have not been visited yet or where the last_time_crawled parameter is outdated. Both the batch size and the expiration time is defined by two environment variables. Therefore, these configurations can be adjusted in the function app environment without having to change the program code.
   We have decided to process a batch of links at once as this gives better performance. Especially if the links_to_crawl table is very large and only limited database access resources are available, it is advantageous to query a batch of entries at once instead of processing only one entry at a time.

2. After that, the following operations are performed for each of the links fetched in step one.

[1] `https://docs.python.org/3/library/re.html`

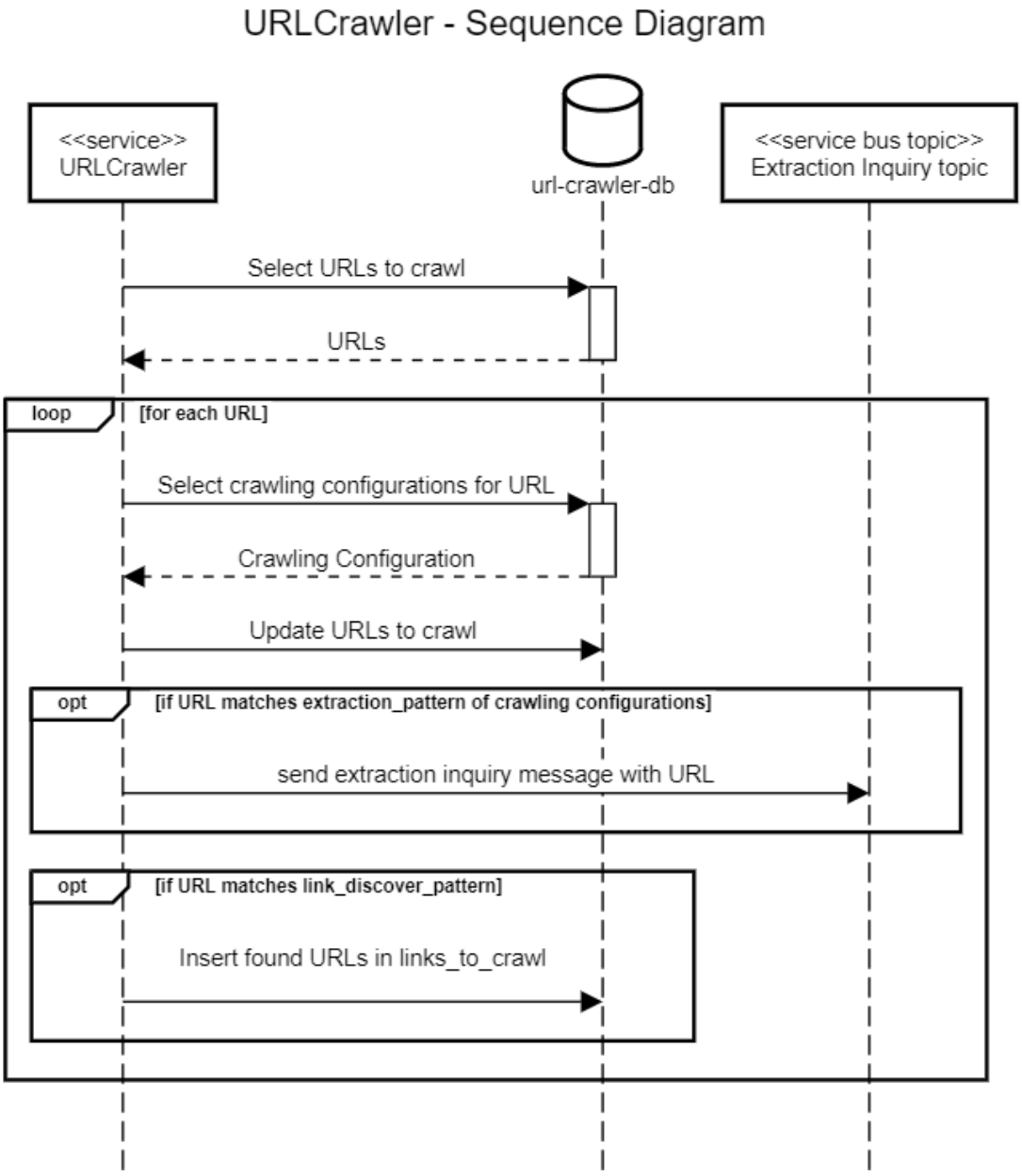

Figure 5.2: Sequence Diagram of the URLCrawler Service

2.1 At first the corresponding crawling configuration for the URL will be requested from the crawling_configurations table. The corresponding configuration can be recognized by a matching netloc parameter. If multiple configurations are defined for a specific netloc parameter, all of those configurations will be

considered.

2.2 The Web page is then retrieved using the python requests library [68]. Thereby a simple HTTP GET request to the desired resource is performed.

2.3 After that, the entry in the links_to_crawl table is updated to indicate that this link has been visited. This is done by adding the HTTP response code and the current timestamp to the respective link.

2.4 A new message will be sent to the extraction inquiry topic on the Azure service bus if the URL matches one of the configurations extraction pattern. The payload of such a message contains the respective URL.

2.5 If the URL matches the link discovery pattern of one of the corresponding configuration entries, all internal links which are forwarding links to Web pages on the same domain will be identified. This is done with the help of the Python Beautifulsoup library [69]. All href attributes of HTML link elements are therefore discovered. If these links match the extraction pattern or the link discovery pattern and the link is not already in the links_to_crawl database table, it will be added to it.

Extraction API (full name: HttpTrigger_ExtractionAPI)

The ExtractionAPI is used to initiate the information extraction process for individual Web pages. Unlike the URLCrawler, the extracted structured data of the requested resources is returned to the requester.

Title	Extract Data from Web page		
URL	api/HttpTrigger_ExtractionAPI		
Method	POST		
Parameters	url	String	Mandatory
	save_entry	Boolean	Optional
	override	Boolean	Optional
Success	200	JSON object	
Error	400	no JSON payload given	
	400	no value 'url' defined in JSON payload	
	400	the optional override parameter has to be a boolean (true, false)	
	400	the optional save_entry parameter has to be a boolean (true, false)	

Table 5.4: Description of the Extraction API Endpoint.

Service Description The ExtractionAPI accepts three different parameters as shown in Table 5.4. The parameter url is mandatory and contains the target page where the information extraction should be applied to. The save_entry parameter defines if the extracted data should be saved to the database (CosmosDB data-lake see Figure 5.1).

The default value is true. The save_entry parameter can be defined as false if the request is just for testing purposes and the extracted data should not be automatically added to the database. The override parameter determines whether the data must be extracted from the resource again under any circumstances. The default parameter is false, and therefore the data is only extracted if there is no entry or just an outdated entry in the database corresponding to the given URL. The handling of this configuration is done by the DataAvailabilityChecker and not by the ExtractionAPI. These parameters will be sent to the Extraction Inquiry topic additionally to the URL in the message payload.
Furthermore, another parameter is defined in the message payload for the Extraction Inquiry topic, namely a custom-generated session id. After the message is beeing sent to the service bus, the ExtractionAPI service waits for an incoming message from the Extraction Response Topic with the previously defined id set as session id. This implements a request-reply messaging pattern with which the requested data can be returned to the correct requestor.

Data Availibility Checker (full name: SBTTrigger_DataAvailabilityChecker)

The DataAvailabilityChecker is used to check whether a corresponding document already exists in the database for a requested URL or if the information extraction process has to be initiated first in order to get the data.

Service Description The general process flow of the DataAvailabilityChecker can be seen in the sequence diagram in Figure 5.3. The DataAvailabilityChecker is triggered by the service bus as soon as a new message is sent to the extraction inquiry topic. The message payload contains the URL of the target Web page. Furthermore, additional parameters can be included which have already been explained in the description of the ExtractionAPI. The DataAvailabilityChecker checks if there is already a document in the CosmosDB data-lake database which refers to the same URL. For each document, a timestamp of the time of extraction is stored. The service checks the existence of the entry and if it is still valid. The validity period is determined by the CosmosDBObjectExpirationTime environment variable. If the document is available, not outdated and the optional override parameter is not defined or false, no extraction process will be initiated. However, if a session id is defined in the message payload (i.e. the message comes from the ExtractionAPI), a message with the structured data from the database is sent to the Extraction Response topic and the given session id is set as session in the message.
On the other hand, if no document or only an outdated one is found, the extraction process is started. To do this, the ExtractionProcessController is called using an HTTP POST request that passes the URL in the payload. The ExtractionProcessController returns the structured data in the response body which is then inserted or overwritten in the database with a current timestamp in the data-lake database (Cosmos DB). If the parameter save_entry is specified as false, then the data will not be saved. Finally, if a session id is specified, a message with the extracted data-object as payload will be sent to the Extraction Response topic.

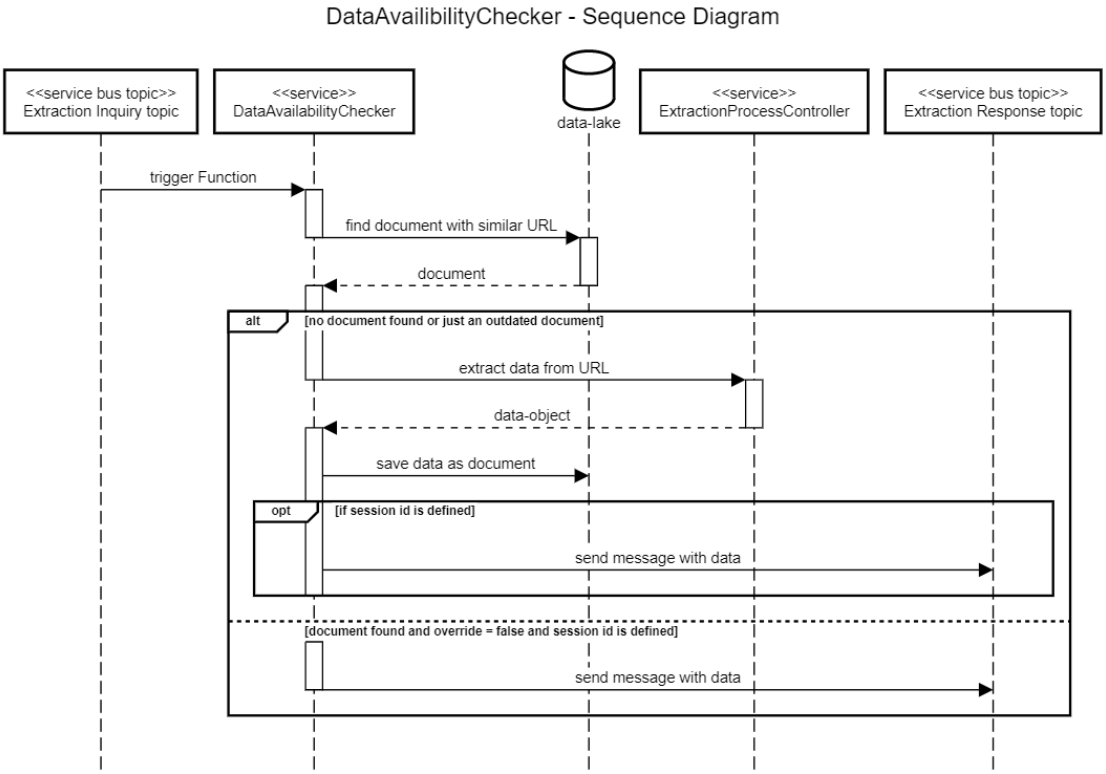

Figure 5.3: Sequence Diagram of the DataAvailabilityChecker Service

Extraction Process Controller (full name: HttpTrigger_ExtractionProcessController)

The ExtractionProcessController takes care of the coordination and execution of the information extraction process. Depending on the Web page from which the information is to be extracted, different process steps are carried out. These processes with the respective process steps are stored in the database *data-extraction-db*. Based on the configuration defined in this database, the ExtractionProcessController calls a sequence of resource extraction and modification services in order to extract certain data from the target Web page. The process workflow can be created and modified directly in the database or by using the front-end dashboard via an easy-to-use form.

Database Description The control of the process workflow is based on four database tables denoted with the prefix 'pc' (process controller) in the data-extraction-db. By abstracting the process flow settings into a dynamic database, we facilitate a highly configurable and flexible workflow. This configurable environment makes it very easy to create new extraction processes or modify existing ones. Furthermore, we have created a system that can be used to introduce completely new extraction processes and add new extraction and modification services without any intervention in the program code,

because the decision which services to call is determined by the database configurations. The ExtractionProcessController accesses a total of five relational database tables, only four of which are critical for the process flow. The database design and the purpose of the individual components are described in detail in this section.

pc_process		
PK	id	INT
	name	VARCHAR

Table 5.5: pc_process Database Table

pc_pattern		
PK	id	INT
Unique	pattern	VARCHAR
FK	process_id	INT

Table 5.6: pc_pattern Database Table

The pc_process database table (see Table 5.5) contains an entry for each process workflow. Such an defined extraction process can be applied to many similar structured Web pages. The database table pc_pattern (see Table 5.6) defines for which Web pages (identified by their URLs), which process should be applied. The field pattern contains a regular expression which should match with a range of URLs. It is possible that the same process is used for several different patterns. Such a pattern could be for example:

```
1 (http:\/\/|https:\/\/)(www\.|)(imdb\.com\/title\/tt)\d*\/?$
```

This example will match with every Web page on the internet movie database website (imdb.com) which presents information about a specific movie or TV show. The table

pc_function		
PK	id	INT
Unique	function_endpoint	VARCHAR

Table 5.7: pc_function Database Table

pc_function (see Table 5.7) contains all resource extraction and modification services. The microservices are identified by their individual function endpoints. The Extraction-ProcessController uses this information to know how to call the corresponding service. When a new extraction or modification service is added to the framework, an entry must also be added to this database table. The database table pc_process_step (see Table 5.8) contains the individual process steps that the ExtractionProcessController performs in sequence for each Web page on which the information extraction is performed. The following parameters must be defined for each process step:

pc_process_step		
PK	id	INT
Not Null	step	INT
FK	function_id	INT
FK	process_id	INT
	input_step	INT
	input_params	VARCHAR
	result_type	VARCHAR

Table 5.8: pc_process_step Database Table

- The parameter step is an integer value from 1 to n. It indicates the execution sequence of the individual process step. An extraction process consisting of five steps would then have five associated entries in the process_step table. For these entries the values of the field step would be between 1 and 5 according to the order of execution.
- The parameter function_id refers to an extraction or modification service defined in the pc_function table. This service is called during the process step.
- Ther parameter process_id refers to an extraction process defined in the pc_process table.
- The input_step parameter is used to define a part of the payload that will be passed to the extraction or modification service to be called. The value refers to a previous process step. The result of this previously defined process step is used as input data for the step to be executed. There are two special cases when using the input_step parameter. If the value is -1 the structured data object composed to that point is used as input data. If the value is 0 the URL of the Web page is used as input data.
- The parameter input_params contains a JSON object with custom parameters which are also included in the payload when requesting the respective extraction or modification service.
- The parameter result_type denotes if the result of the execution of the defined function should be included in the overall resulting data-object ("final") or if the result is just used as input data for subsequent processing steps ("interim").

The table extraction_process_log (see Table 5.9) is used to record malfunctions or unexpected errors during the extraction process. The logs include a custom error message, an error level which indicates the seriousness of the error, a timestamp, the URL of the Web page on which the malfunction occours, the corresponding extraction process, as well as the respective extraction step. The information from this process logs is presented in the front-end dashboard to simplify the identification and correction of incorrectly defined process flows.

extraction_process_log		
PK	id	INT
	message	VARCHAR
Not Null	error_level	INT
	timestamp	DATETIME
Not Null	url	VARCHAR
FK	extraction_process	INT
	extraction_step	INT

Table 5.9: extraction_process_log Database Table

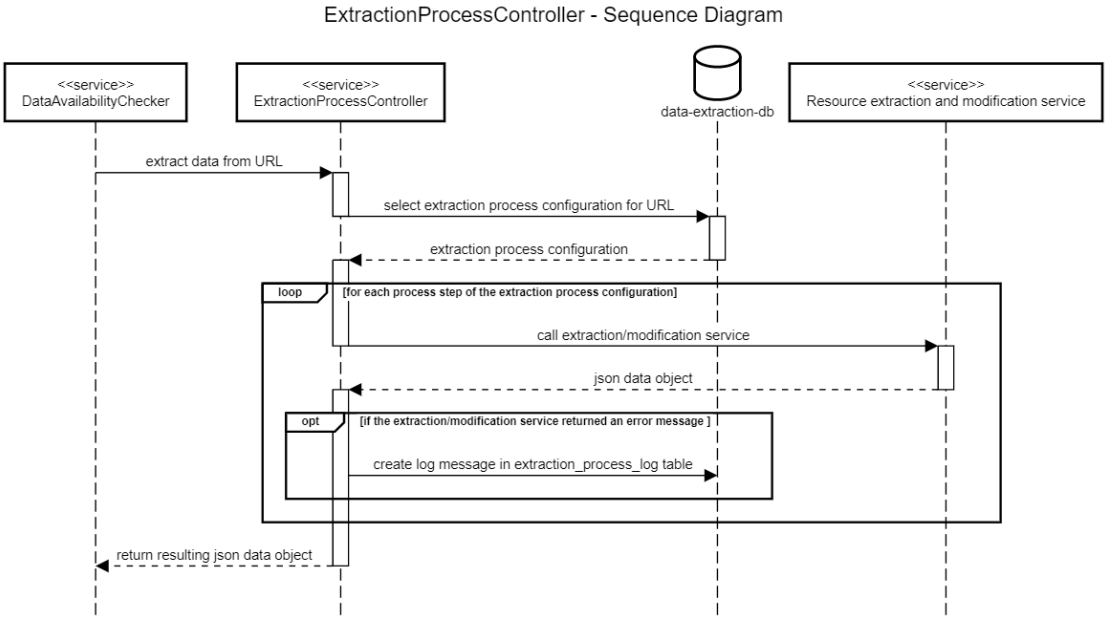

Figure 5.4: Sequence Diagram of the ExtractionProcessController Service (Note: The swim lane "Resource extraction and modification service" is a generalization, that refers to the selection of a specific Azure Function out of this group of services.)

Service Description The ExtractionProcessController directs the information extraction process. The procedure can be seen in the sequence diagram in Figure 5.4. The following process steps are executed in sequence for the information extraction of a specific Web page.

1. The service is called by DataAvailabilityChecker, which provides the URL of the target Web page in the request payload.

2. The ExtractionProcessController queries the specified URL patterns in the database

and checks which process should be executed for the given URL, by matching the pattern expression with the URL.

3. Next, all associated process steps are retrieved from the database for the selected process.

4. For each individual process step, ordered by the value of the field step, the following actions are performed.

   4.1 The Azure function of the currently executed process step is called by the ExtractionProcessController. Thereby two parameters are passed in the payload, namely "data" and "config". If the value of input_step is 0, the URL of the target Web page is assigned to the "data" parameter. With the value is between 1 and the <current step number> -1 the result of the this previously executed process step is assigned to the "data" parameter. If the value is -1 the structured data object composed to that point is used as input data.
   The parameter "config" is assigned the JSON object defined within the field input_params. For each extraction modification service certain attributes are allowed to be passed. The declaration of these attributes for each process step can be made in the front-end dashboard during the extraction process creation. These configuration parameters will then be put together into a single JSON object which gets stored in a serialized form in the database. Which configuration parameters are allowed depends on the used extraction or modification services. The definition of these parameters is explained in more detail in the front-end description.

   4.2 If the called service returns an error response, this is recorded in the extraction_process_log table. Otherwise, the result is first stored in an array in order to be able to pass the result as input_step in upcoming process steps. Furthermore, if the result_type of the process_step is specified as final, the resulting data is added to the data object which contains all the extracted data from the current extraction process.

5. Finally, if the process was executed successfully, the final structured data object is returned to the DataAvailabilityChecker. If an error occurred, for example if no data was found or the Web page, then an appropriate error message is returned instead.

5.1.3 Resource Extraction and Modification Services

The resource extraction and modification services provide the functionality to extract structured data from Web pages. Each service can perform a specific task, and depending on the extraction process, the services can be configured differently. The services are called sequentially via an HTTP POST request from the ExtractionProcessController. Two arguments are defined in the payload. The first parameter is called "data", it contains the result of a previously executed process step or the URL of the target Web page. The

value of the parameter is defined in the database table pc_process_step (see Table 5.8) in the field input_data. The second parameter is called "config", a JSON object containing one or more attributes that influences the services in a certain way, so that the behavior remains flexible and can be tailored to individual needs.
The setting of the config parameters can be done very easily via the front-end. The front-end dashboard accesses a database table in which all possible configuration parameters and their options are defined for each extraction and modification service. This section describes the possible configurations for each service, while Chapter 5.2 presents the front-end dashboard where these parameters can be set.
Each of the resource extraction and modification services returns either a JSON object containing structured data extracted from the Web page, or an interim result which is used for subsequent process steps. In this Chapter, the individual extraction and modification services are explained in detail.

DefaultResourceCollector

Title	Retrieve target Web page			
URL	api/HttpTrigger_DefaultResourceCollector			
Method	POST			
Parameters	data	String	Mandatory	URL of target page
	config.headers	JSON object	Optional	Custom headers parameter
Success	200 HTML content serialized as String			
Error	400 no JSON payload given			
	400 no data parameter given			
	!200 requests.get result message			

Table 5.10: Description of the DefaultResourceCollector Endpoint.

The DefaultResourceCollector (see Table 5.10) requests the target Web page, using an HTTP GET request, and returns its entire HTML content encoded as a String. For retrieving the Web page, the service uses the Python library requests [68]. Thereby it is possible to define certain HTTP header parameters. This is sometimes necessary because certain websites set policies to restrict the access for Web Crawlers. To circumvent these policies particular defined headers can be used to not be blocked by the site. As default headers the following User-Agent is used: "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.169 Safari/537.36". However, the process creator can also specify his own custom headers which will be used instead.
The content of the HTML Web page is then read and serialized using the Python library Beautifulsoup [69] and sent back, serialized as String, to the ExtractionProcessController.

Title	Returns a specific section/element of the target HTML page			
URL	api/HttpTrigger_HtmlSectionExtractor			
Method	POST			
Parameters	data	String	Mandatory	HTML Web page or part of it
	config.occurrence	Integer	Optional	Occourence of the target element
	config.tag	String	Mandatory	Target HTML tag
	config.search_params	JSON object/list	Optional	Additional element search parameters
Success	200	HTML sub-section serialized as String		
Error	400	no JSON payload given		
	400	no data parameter given		
	400	no config parameter set		
	400	no tag to search for defined		

Table 5.11: Description of the HtmlSectionExtractor Endpoint.

HtmlSectionExtractor

The HtmlSectionExtractor (see Table 5.11) identifies a certain part of an HTML page and sends it back to the ExtractionProcessController. This function is used when the relevant data is located in a certain segment of a page and is mostly used as an intermediate step of the information extraction process.
To identify the desired section, the Python library Beautifulsoup is used again. By specifying the parameter tag, a particular HTML element will be identified on the Web page (for example: "h1", "div", "section", ...). The search can be classified more closely by using the parameter search_param. This parameter can contain a single argument or a list of arguments consisting of Strings or dictionaries containing key-value pairs. A simple argument encoded as String always refers to the HTML attribute "class" of an HTML element. With the definition of a key-value pair object the search option can also be applied to other attributes. For example, the argument "'id':'filmography'" would constrain the search to HTML elements that have the attribute "id" with the value "filmography" defined. With the configuration parameter occurrence it is possible to select an element when multiple suitable elements are available. The default value is 1 which results in the selection of the first element that fits the search configuration. By defining the occurrence parameter, it is also possible to select the second, third, etc. occurrence of the matching elements.

Title	Extracts the JSON-LD encoded data			
URL	api/HttpTrigger_JsonLDExtractor			
Method	POST			
Parameters	data	String	Mandatory	HTML Web page or part of it
Success	200	Serialized JSON data-object		
Error	400	no JSON payload given		
	400	no data parameter given		
	400	no JSON-LD section found on webpage		

Table 5.12: Description of the JsonLDExtractor Endpoint.

JsonLDExtractor

The JsonLDExtractor (see Table 5.12) identifies the structured data encoded in JSON-LD, a serialization format of RDF, and returns this data to the ExtractionProcessController. Many websites now publish JSON-LD serialized data, as this is also recognized by popular search engine Crawlers. Typically, this data is provided within a HTML "script" tag with the attribute "type" having the value "application/ld+json". This information is extracted using the Python library Beautifulsoup. The content of the identified HTML element is then returned to the ExtractionProcessController.

RDFaExtractor

Title	Extracts RDFa encoded data			
URL	api/HttpTrigger_RDFaExtractor			
Method	POST			
Parameters	data	String	Mandatory	HTML Web page or part of it
Success	200	Serialized JSON data-object		
Error	400	no JSON payload given		
	400	no data parameter given		
	400	no JSON-LD section found on webpage		

Table 5.13: Description of the RDFaExtractor Endpoint.

The RDFaExtractor (see Table 5.13) identifies and extracts data from RDFa annotated elements on HTML Web pages. RDFa uses certain attributes that are added to HTML elements in order to make the presented information machine-readable. This provides a way to extract the inner structure and meaning of the presented data.
To read the RDFa annotated data, the Python library pyRdfa [70] is used. The extracted information is thereby stored in a RDF graph. However, since we process and store the extracted data generally in a JSON format, it has to be transformed from a RDF graph

to a JSON object. For this purpose, the Python library xmltodict [71] is used. This is done by first serializing the graph to XML, using a parsing function from pyRDFa, and then using xmltodict to create a JSON data object from it.

MicrodataExtractor

Title	Extracts with Microdata annotiated information
URL	api/HttpTrigger_MicrodataExtractor
Method	POST
Parameters	data String Mandatory HTML Web page or part of it
Success	200 Serialized JSON data-object
Error	400 no JSON payload given 400 no data parameter given 400 No microdata found

Table 5.14: Description of the MicrodataExtractor Endpoint.

Microdata is used similarly to RDFa to annotate metadata to HTML elements to provide additional semantic information. The MicrodataExtractor (see Table 5.14) extracts this data using the Python library microdata [72]. However, since the structure of the extracted data does not correspond to the generally applied data schema of our system, the data must be transformed again afterwards. The microdata library uses additional 'properties' objects for nested information, which we do not utilise in our target schema. For this we use the adapted function "json_dict" which transforms the information into the desired schema.

HtmlConsecutivePropertyValuePairsExtractor

The HtmlConsecutivePropertyValuePairsExtractor (see Table 5.15) extracts key-value pairs that occur on HTML pages. Thereby the given parameters define how the property name and the value expression are encoded. For the parameter property_tag and value_tag the tag name of the enclosing HTML element is defined. With the parameters property_search_params and value_search_params additional search parameters can be defined to identify the target element more precisely.
If the property and value definitions are the same, i.e. both are in similar tags with similar attributes, the service assumes the first, third, fifth, etc occurrence of this element contains the property name and the second, fourth, sixth, etc occurrence contains the value.
If the property and value element definitions differ, the first occurrence of the property definition is mapped to the first occurrence of the value definition, and so on.

Title	Extracts key-value pairs from HTML listings			
URL	api/HttpTrigger_HtmlConsecutivePropertyValuePairsExtractor			
Method	POST			
Param.	data	String	Mandatory	HTML Web page or part of it
	config.property_tag	String	Optional	HTML tag of property definition
	config.property_search_params	JSON object/list	Optional	Additional element search parameters
	config.value_tag	String	Optional	HTML tag of value definition
	config.value_search_params	JSON object/list	Optional	Additional search parameters
Success	200	Serialized JSON data-object		
Error	400	no JSON payload given		
	400	no data parameter given		
	400	TypeError: ... on instantiation of BeautifulSoup object.		

Table 5.15: Description of the HtmlConsecutivePropertyValuePairsExtractor Endpoint.

HtmlListOfValuesExtractor

The HtmlListOfValuesExtractor (see Table 5.16) extracts a list of values that can be assigned to a specific property from an HTML page. The property name is either specified by an HTML element definition using the parameters property_tag and property_search_params, or the property name is specified directly using the parameter property_name. The list elements containing the individual values are specified via the parameters value_section_tag and value_section_search_params. The element that contains the exact value is specified with the parameters value_tag and value_search_params. For example, if we have a list consisting of several "div" elements and the property value is in an "a" element within the list elements, we can describe the existing structure accordingly with the following specification:

```
{
    "value_section_tag2": "div",
    "value_tag": "a"
}
```

Title	Extracts a list of values for a single property definition			
URL	api/HttpTrigger_HtmlListOfValuesExtractor			
Method	POST			
Param.	data	String	Mand.	HTML Web page or part of it
	config.property_name	String	Opt.	Property name
	config.property_tag	String	Opt.	HTML tag of property definition
	config.property_search_params	JSON obj/list	Opt.	Additional property element search parameters
	config.value_section_tag	String	Opt.	Superordinated element of values definition
	config.value_section_search_params	JSON obj/list	Opt.	Superordinated search parameter for values definition
	config.value_tag	String	Opt.	HTML tag of value definition
	config.value_search_params	JSON obj/list	Opt.	Additional search parameters
Success	200 Serialized JSON data-object			
Error	400 no JSON payload given			
	400 no data parameter given			
	400 TypeError: ... on instantiation of BeautifulSoup object.			
	400 no property definition found. (also no property name explicitly defined)			

Table 5.16: Description of the HtmlListOfValuesExtractor Endpoint.

HtmlPropertyValueListExtractor

The HtmlPropertyValueListExtractor (see Table 5.17) extracts a listing of property-value pairs from an HTML page. It first searches for the sections in which a property-value pair is located using the section_tag and section_search_params parameters. In each of these found sections the property and the value is identified. If there are several similar elements in one of these sections and only one of them is relevant, the parameters property_occourrence and value_occourrence can be used to define which one is used. The default value in each case is 1, which means that the system selects the first element found that matches the configuration.
For example, if the HtmlPropertyValueListExtractor is used to extract a 2 dimensional HTML table that has the property names defined in the first column and the corresponding

Title	Extracts a listing of property-value pairs			
URL	api/HttpTrigger_HtmlPropertyValueListExtractor			
Method	POST			
Parameters	data	String	Mand.	HTML Web page or part of it
	config.section_tag	String	Opt.	HTML tag of section (list element)
	config.section_search_params	JSON object/list	Opt.	Additional section (list element) search parameters
	config.property_tag	String	Opt.	HTML tag of property definition
	config.property_search_params	JSON object/list	Opt.	Additional property search parameters
	config.property_occourrence	Integer	Opt.	Element to use if multiple values with the same tag and properties exist
	config.value_tag	String	Opt.	HTML tag of value definition
	config.value_search_params	JSON object/list	Opt.	Additional search parameters
	config.value_occourrence	Integer	Opt.	Element to use if multiple values with the same tag and properties exist
Success	200	Serialized JSON data-object		
Error	400	no JSON payload given		
	400	no data parameter given		
	400	TypeError: ... on instantiation of BeautifulSoup object.		

Table 5.17: Description of the HtmlPropertyValueListExtractor Endpoint.

values in the second column, the following parameters could be defined:

```
{
    "section_tag":"tr",
    "property_tag":"td",
    "property_occourrence": 1,
```

```
5     "value_tag":"td",
6     "value_occourrence": 2
7 }
```

.

HtmlSinglePropertyExtractor

Title	Extracts a single property-value pair			
URL	api/HttpTrigger_HtmlSinglePropertyExtractor			
Method	POST			
Parameters	data	String	Mandatory	HTML Web page or part of it
	config.property_name	String	Mandatory	Name of the property
	config.value_tag	String	Mandatory	HTML tag of value definition
	config.value_search_params	JSON object/list	Optional	Additional search parameters
	config.value_attribute	String	Optional	Attribute of the HTML element which contains the target value
Success	200	Serialized JSON data-object		
Error	400	no JSON payload given		
	400	no data parameter given		
	400	mandatory parameter not set ('property_name' and/or 'value_tag')		

Table 5.18: Description of the HtmlSinglePropertyExtractor Endpoint.

The HtmlSinglePropertyExtractor (see Table 5.18) extracts a single value from an HTML page and assigns it to a defined property. In contrast to the previously discussed extraction services, it is possible that the target value is defined as an attribute value of an HTML element. This can be communicated to the service by defining the parameter value_attribute. If the parameter value_attribute is not defined, the wrapped inner text of the described HTML element is automatically taken as the value.

ManualJsonObjectCreator

The ManualJsonObjectCreator (see Table 5.19) adds custom created attributes to the structured data object. Thereby a data object serialized as String is submitted to the service, using the configuration parameter json_object. This service can be used if it is clear in advance that in an extraction process the assigned group of Web pages can be

Title	Adds the defined JSON object to the result			
URL	api/HttpTrigger_ManualJsonObjectCreator			
Method	POST			
Parameters	config.json_object	String	Mandatory	JSON object containing property-value pairs
Success	200	Serialized JSON data-object		
Error	400	no JSON payload given		
	400	parameter json_object is not a valid JSON object		
	400	no json_object parameter defined in config		
	400	no config parameter in request body		

Table 5.19: Description of the ManualJsonObjectCreator Endpoint.

assigned certain attributes which are the same for all of them. This can be, for example, the simple addition of the source domain or the entity type (such as "Movie", "Person", ...), which can be applied to all resources on which a particular extraction process is performed. Unlike most other extraction and modification functions, the ManualJsonObjectCreator does not use an input step but only uses the defined json_object, defined in the config parameter, since no data extraction takes place.

PropertyMapping

The PropertyMapping (see Table 5.20) service is used to apply a schema generalization on some structured data. When information is extracted from different content providers, different labels are often used for conceptually the same things. To make such properties more comparable a schema mapping can be applied using this service. In this process, property names are renamed to specific custom defined names. For this process two database tables[2] (see Tables 5.22 and 5.21) have been created to store the property label transformations. These configurations can be created and modified in the front-end dashboard or directly in the database. Each property transformation is associated with a specific property_mapping_setting, which in turn has a unique name. If the corresponding id is passed to the PropertyMapping service using the property_mapping_setting parameter, those configurations will be applied. The additional_property_mappings parameter can be used to define additional transformations for a specific extraction process. For this a JSON object is specified where each key-value pair corresponds to a mapping ("'Source name': 'Target name'"). Nested object keys can be denoted by using the term "->" e.g. "'key1->key2->key3':'new name of key3'".

Title	Renames properties in order to apply a common schema			
URL	api/HttpTrigger_PropertyMapping			
Method	POST			
Parameters	data	String	Mand.	Serialized JSON object
	config.property_mapping_setting	Integer	Optional	Id of the property mapping setting that should be applied
	config.additional_property_mappings	String	Optional	Serialized JSON object which encodes a property mapping
Success	200 Serialized JSON data-object			
Error	400 no JSON payload given 400 parameter json_object is not a valid JSON object 400 JSON decoding has failed for parameter 'data' in HttpTrigger_PropertyMapping 400 JSON decoding has failed for parameter 'additional_property_mappings' in HttpTrigger_PropertyMapping			

Table 5.20: Description of the PropertyMapping Endpoint.

property_mapping_setting		
PK	id	INT
Unique	name	VARCHAR

Table 5.21: property_mapping_setting Database Table

property_mapping_configuration		
PK	id	INT
Not Null	source_name	VARCHAR
Not Null	target_name	VARCHAR
FK	setting_id	INT

Table 5.22: property_mapping_configuration Database Table

Title	Adds properties from the Google Knowledge Graph Search			
URL	api/HttpTrigger_GoogleKnowledgeGraphRetrieval			
Method	POST			
Parameters	data	String	Mand.	Serialized JSON object
	config.query	String	Mandatory	Attribute key of the input step data which should be used as query argument
	config.type	String	Optional	Attribute key of the input step data which should be used as type parameter
	config.return_values	list of Strings	Mandatory	Defines which properties of the google knowledge graph search should be included in the resulting response
Success	200 Serialized JSON data-object			
Error	400 no JSON payload given 400 parameter json_object is not a valid JSON object 400 JSON decoding has failed for parameter 'data' in HttpTrigger_GoogleKnowledgeGraphRetrieval 400 Parameter query has to be an Integer 400 Key of 'query' parameter not found in input step data 400 Key of 'type' parameter not found in input step data 400 no return values defined !200 Bad Request using kgsearch 400 Google Knowledge Graph could not find a corresponding Entity			

Table 5.23: Description of the GoogleKnowledgeGraphRetrieval Endpoint.

GoogleKnowledgeGraphRetriever

The GoogleKnowledgeGraphRetrieval service (see Table 5.23) retrieves a specific entity from the Google Knowledge Graph[3] API to add additional attributes to the structured

[2]Both database tables used for the property mapping, are created in the data-extractiondb
[3]https://developers.google.com/knowledge-graph

data object. We use this service mainly to add certain properties to our structured data object which increases the comparability of similar entities. The goal is not to simply include the data that Google has stored about a particular entity, as this would not be consistent with the goal of this project.
Google stores a unique id for each entity. If we add this id to the structured data objects of all our extracted Web pages, we can use it to detect if two records refer to the same entity. This is much more efficient than comparing properties such as titles or names, which are often different anyway.
There are three configuration parameters that can be used to modify the behaviour of the GoogleKnowledgeGraphRetrieval service. The parameter query corresponds to the key of the property in the input data object that best describes the resource (e.g. "Alexander van der Bellen", "The Shawshank Redemption", "The Adventures of Tom Sawyer"). This parameter is used to query the Google Knowledge Graph API for this specific entity. The parameter type defines, in which attribute of the input data the entity type of the resource is defined (e.g. movie, book, person). The parameter return_values contains the property fields of the retrieved Google Knowledge Graph entity which should be returned by the service to the ExtractionProcessController. The following fields are supported:

- description: Description of the resource entity.
- name: Name of the entity. Probabily similar to the query value.
- @type: Entity type of the resource (Schema.org entity type).
- url: Some reference URL of the entity.
- @id: Google Knowledge Graph unique id of the entity. This is probably the most important property.
- detailedDescription: More detailed description of the resource entity.

To access the Knowledge Graph Search API, an API key must be used. This API key is stored in the environment variable "GoogleKnowledgeGraphAPIKey" and can be changed at any time without having to change the program code.

5.2 Front-End

The data extraction process is configured based on the database entries described in the previous chapter. Therefore, it can be used with the provided API service (ExtractionAPI and URLCrawlerAPI) and by directly accessing the database. A front-end dashboard is not necessarily needed to achieve the main goal of this project. However, a user-friendly dashboard simplifies the configuration of the system. Instead of manually setting up the process configurations in the database, a dashboard was developed which simplifies this process significantly. This makes it possible to create and test new extraction processes with just a few clicks.

5.2.1 Technical Background

The front-end dashboard was developed based on a Django Web framework (see Section 4.4.1). A great advantage of using this Web framework is a very good organization of the program code which improves readability and code reuse. The consistent adoption of the Model View Controller principle separates the data access, the logical process flow and the presentation. Extensive module libraries are available and help to solve problems quickly and clean.
For each part of the management of the system, a separate app has been developed, but all apps belong to the same Django application environment. The code is well-organized this way. A total of five apps have been developed. Each app constructs one page, that handles a specific task area of the composite website.
The Web application is hosted using the Azure App service (see Section 4.6.5). This service provides scalable and adaptable runtime environment to meet the demands of modern Web development.
To simplify the creation of attractive Web pages, the front-end css framework Bootstrap [73] is used. This allows a wide range of style templates to be used, which facilitates the creation of a modern and user-friendly interface.

5.2.2 Web Pages

To manage the data extraction system, a number of views have been developed to create and edit workflows, and to configure, test and analyze the processes. The front-end dashboard consists of a general navigation, used to switch between the individual Web pages. Responses to form submissions are displayed to the user directly below the navigation. If the input is successful, a success message with a green background is displayed. If the input is incorrect, an error message with a red background is displayed. In Figure 5.5 you can see the five different pages which will be discussed in detail in this chapter.

Extraction Process Configuration

The Extraction Process configuration consists of two parts. A form for the creation of the extraction process workflow and the pattern-process assignment. The two sections are

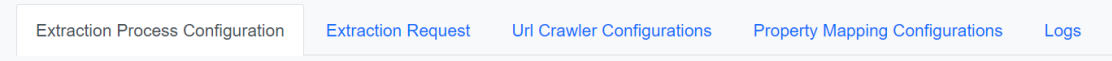

Figure 5.5: Navigation of the Front-end Dashboard

displayed either side by side or one below the other, depending on the display size. The purpose and functionality of these two forms are explained in the following paragraphs.

Extraction Process Configuration
In the extraction process configuration, new extraction process flows are defined and existing ones can be viewed and edited. Each extraction process has a name and consists of several process steps. In Figure 5.6 you can see the initial form to create a new process.

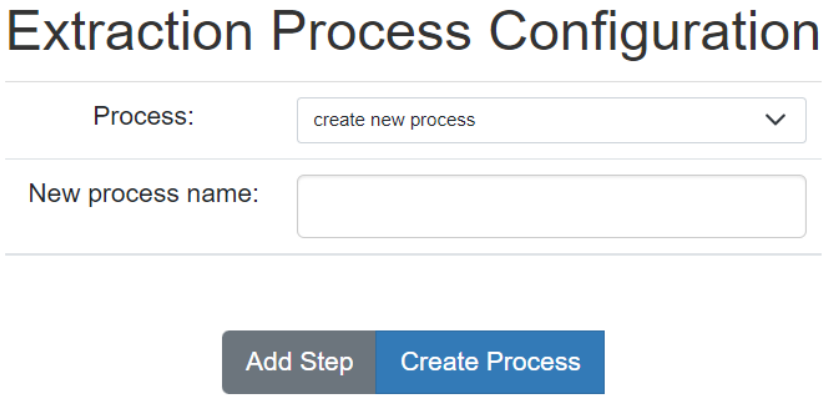

Figure 5.6: New Process Creation View

If you select an already defined process in the selection box instead of creating a new process, the text field "New process name" disappears and the individual process steps with the saved settings are displayed below. Each extraction step configuration consists of four parts as can be seen exemplarily in Figure 5.7.

1. *Function*: First of all, you have to select which extraction or modification service should be executed for this step. In the select field all functions, which are defined in the database table pc_function, are displayed.

2. *Input Step*: Secondly, you specify what should be used as input step for the selected function. There are basically four different options to choose from:
   - No input step
   - URL - The URL of the target resource to extract
   - A previous process step (1 to current step -1)

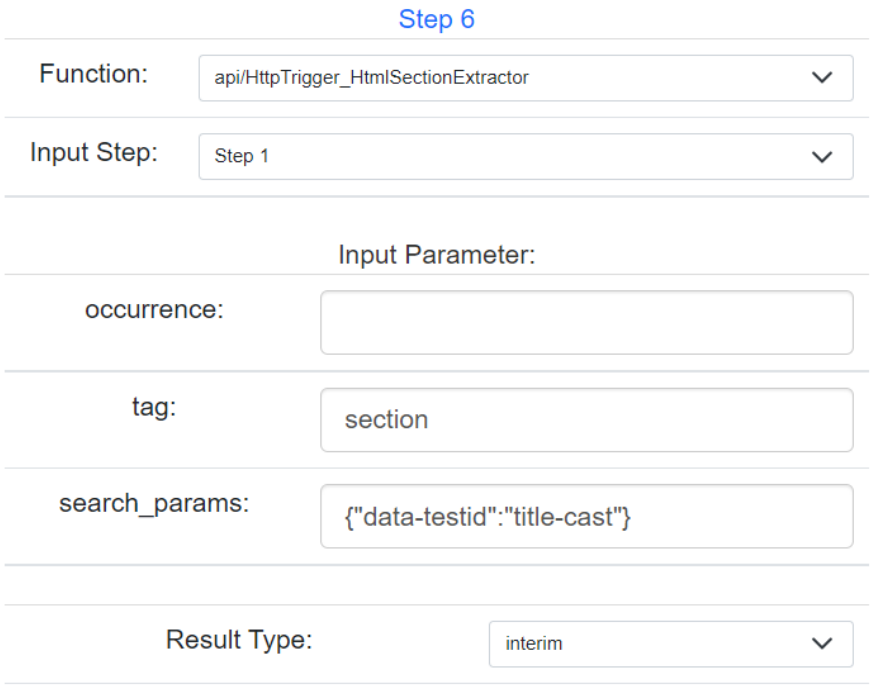

Figure 5.7: Exemplary Process Step Definition

- The latest final data object. This corresponds to the data object that is created up to that point when this process step is executed.

3. *Input Parameters*: The input parameters, or previously refered to as config parameters, section is loaded dynamically when a function is selected. For each possible input parameter that the function accepts, a corresponding input field is displayed. Which input fields are available for a function and which input values are allowed for it is defined in the two database tables pc_input_parameter (see Table 5.24) and pc_input_parameter_value (see Table 5.25). When creating a new extraction service, the corresponding entries must be created in these two tables.

pc_input_parameter		
PK	id	INT
FK	function_id	INT
Not Null	name	VARCHAR
Not Null	parameter_type	VARCHAR
Not Null	mandatory	BIT
	description	VARCHAR
Not Null	multiple_values	BIT

Table 5.24: pc_input_parameter Database Table

pc_input_parameter_value		
PK	id	INT
FK	input_parameter_id	INT
Not Null	val	VARCHAR
	description	VARCHAR

Table 5.25: pc_input_parameter_value Database Table

In the table pc_input_parameter an entry is created for each configuration parameter. To specify a configuration parameter, the function to which the parameter belongs, the name of the parameter, the parameter type, if the specification is mandatory, an optional parameter description and if the specification of multiple values is allowed have to be inserted in the database table. The name of the configuration will be displayed as label in the front-end form and is the key of the config parameter that will be passed to the function. The description will be displayed when you go over the label with the mouse on the form. The field parameter_type specifies which type of input field is displayed and therefore which values are allowed. The following five options are possible to use:

- text: Creates a text input field.
- number: Creates a number input field .
- json: Creates a text input field which checks if the entered string is a valid JSON object.
- select: Creates a select box to choose from different options.
- checkbox: Creates one or multiple checkboxes.

If the input parameter is of type text, number or json a single entry in the table pc_input_parameter is sufficient. However, if the parameter is of type select or checkbox, options associated with this parameter must be created in the table pc_input_parameter_value. In this table, the field val corresponds to the value of the option which is passed to the extraction function if the option is selected and the field description is the label which is displayed to the user for this option.

4. *Result Type*: The result type specifies whether the result of the process step is only an intermediate result that is used for the following steps or whether the result contains structured data that shall be included in the resulting data object.

To add process steps you have to click on the button "Add Step". As soon as at least one step has been added, a second button "Remove Last Step" appears, which can be used to remove the last step. In addition, if you edit an already existing process, another button "Delete Process" is displayed.

Pattern - Process Assignment

In the Figure 5.8 you can see the pattern - process assignment form. This form is used

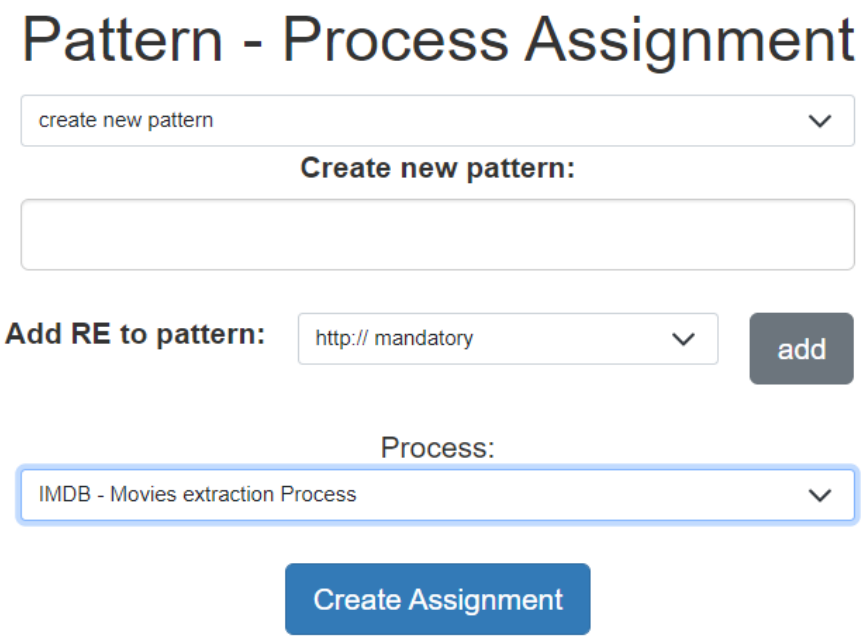

Figure 5.8: Pattern - Process Assignment Form

to assign Web pages, which match a certain pattern, defined by a regular expression, to an extraction process. For the Web pages that match this pattern, the assigned process is then executed.
The pattern - process assignment consists of the following parts:

1. *Pattern Selection*: With the first select box it is either possible to create a new assignment if you choose "create new pattern", or to edit an existing assignment, by selecting the corresponding pattern definition.

2. *Pattern Text Field*: The regular expression for the assignment is entered in the pattern text field. When an existing assignment is selected, the corresponding regular expression is loaded into this field.

3. *Expression Creation Helper*: To simplify the creation of the regular expression, there is next a drop down box with suggestions for the individual parts that often occur for the creation of a pattern for a URL. By selecting one of these suggestions and then clicking on add, the corresponding expression will be added to the pattern text field.

4. *Process Selection*: Finally, the process to be executed for this pattern must be selected.

By the way, if you don't create a new pattern but edit an existing one, next to the button

"Create Assignment" a second button "Delete Assignment" is displayed to delete existing assignments.

Extraction Request

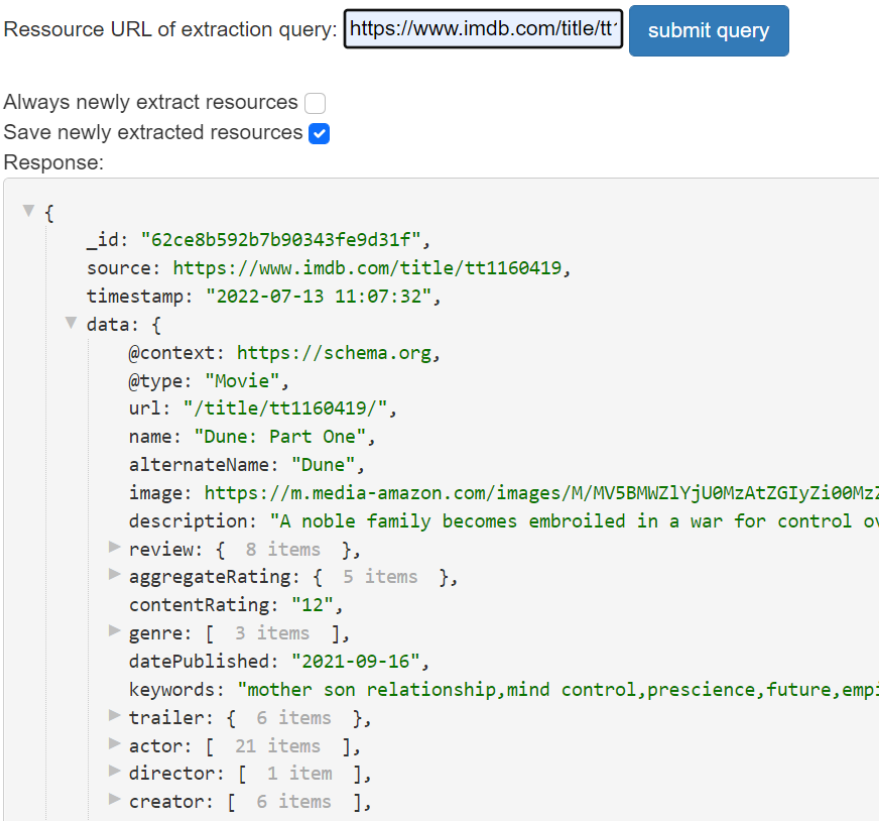

Figure 5.9: Snippet of the Extraction Request View

In Figure 5.9 you can see the extraction request Web page. Here you can specify a URL for which the data extraction process should be executed. If a suitable extraction process has been defined for this URL, the resulting data object will subsequently be displayed in an interactive text-box below. To better analyze the JSON object the properties with nested objects and lists can be expanded and collapsed. This interactive visualisation is based on the jQuery json-viewer[4].
Two checkboxes can be used to specify whether the target resource should also be stored permanently in the database and whether the extraction process should be executed again if a valid data object already exists in the database.

[4]https://github.com/abodelot/jquery.json-viewer

URL Crawler Configurations

The URL Crawler Configurations View is used to configure the execution commands of the URLCrawler. The view shows two parts which are displayed next to or below each other depending on the display size.
In the Figure 5.10 you can see the creation form for a new URL Crawler configuration. The

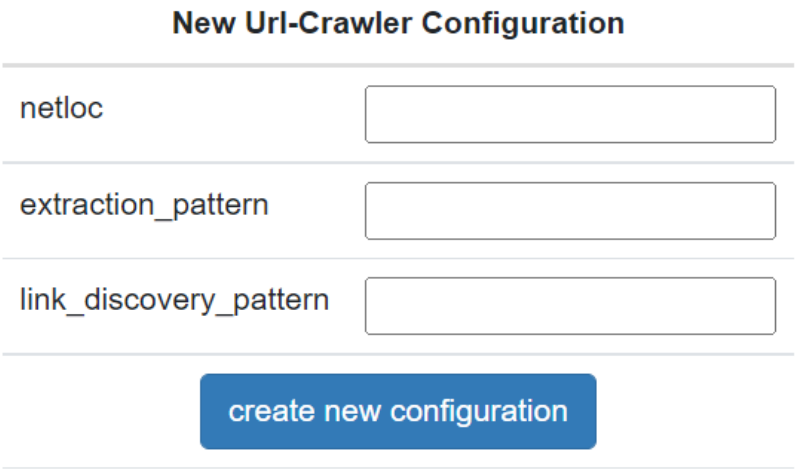

Figure 5.10: URL Crawler Configuration - Create Form

user must specify the parameters netloc, extraction_pattern and link_discovery_pattern to create a new configuration record, for which there are three text fields. While the netloc field corresponds to a Web domain, possibly including a subdomain definition, the two pattern parameters must be valid regular expressions matching a range of URLs. With this information a new entry is created in the database table crawling_configuration (see Table 5.2).

id	netloc	extraction_pattern	link_discovery_pattern		
1	de.wikipedia.org	(http:\/\/|https:\/\/)(www\.|)(de\.|en\.)(wikipedia\.org\/wiki\/)\w*\/?$	(http:\/\/|https:\/\/)(www\.|)(de\.|en\.)(wikipedia\.org\/wiki\/)\w*\/?$	change	delete
2	en.wikipedia.org	(http:\/\/|https:\/\/)(www\.|)(de\.|en\.)(wikipedia\.org\/wiki\/)\w*\/?$	(http:\/\/|https:\/\/)(www\.|)(de\.|en\.)(wikipedia\.org\/wiki\/)\w*\/?$	change	delete
3	www.imdb.com	(https:\/\/)(www\.|)(imdb\.com\/name\/nm)\w*\/?$	(https:\/\/)(www\.|)(imdb\.com\/name\/nm)\w*\/?$	change	delete
7	www.imdb.com	(https:\/\/)(www\.|)(imdb\.com\/title\/tt)\d*\/?$	(https:\/\/)(www\.|)(imdb\.com\/title\/tt)\d*\/?$	change	delete
8	www.moviepilot.de	(http:\/\/|https:\/\/)(www\.|)(moviepilot\.de\/movies\/)(\w|-)*\/?$	(http:\/\/|https:\/\/)(www\.|)(moviepilot\.de\/movies\/)(\w|-)*\/?$	change	delete
9	rottentomatoes.com	(http:\/\/|https:\/\/)(www\.|)(rottentomatoes\.com\/m\/)\w*\/?$	(http:\/\/|https:\/\/)(www\.|)(rottentomatoes\.com\/m\/)\w*\/?$	change	delete
10	rottentomatoes.com	(http:\/\/|https:\/\/)(www\.|)(rottentomatoes\.com\/tv\/)\w*\/?$	(http:\/\/|https:\/\/)(www\.|)(rottentomatoes\.com\/tv\/)\w*\/?$	change	delete

Figure 5.11: URL Crawler Configurations - Modification Form

In the second part of the URL Crawler configuration view (see Figure 5.11) we can see all existing entries in the crawling_configuration database table. With a click on the button "change" we can edit the corresponding fields in the respective line. The button then displays the word "save" instead of "change" and clicking again updates the corresponding entry in the database. In addition, the delete button can also be used to delete entries completely.

Property Mapping Configurations

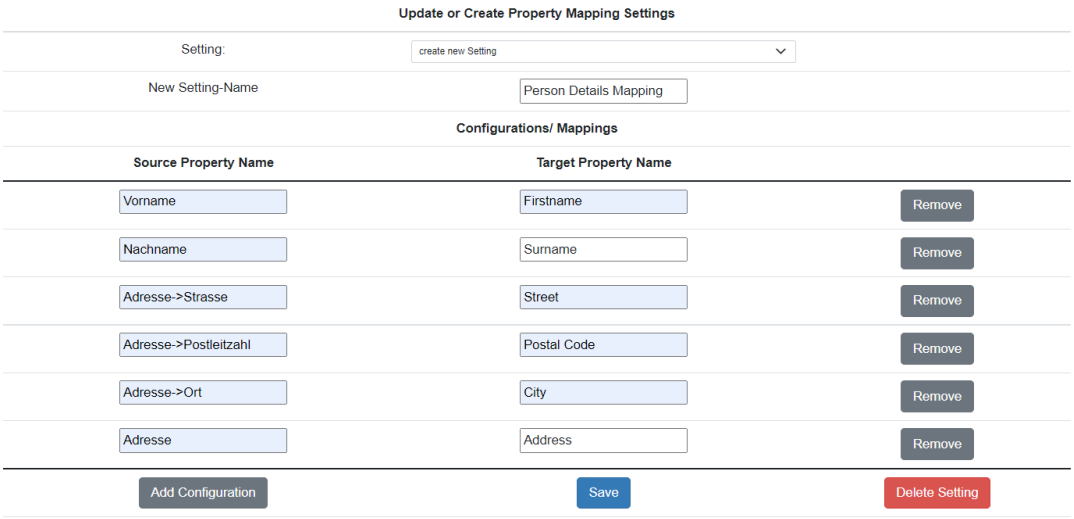

Figure 5.12: Property Mapping Configurations Form

On the Property Mapping Configurations page, new property name mappings can be created which may then be used during the extraction process creation when using the PropertyMapping service. For example, a mapping from a German naming scheme to an English one could look like the one shown in Figure 5.12. When a new mapping is created, an entry is also created in the database table pc_input_parameter_value. Therefore you can select the setting afterwards in the extraction process creation. In the settings selection at the beginning of the form you can choose to create a new setting or select an existing one. This configuration can then be edited or deleted.

Logs

The Logs tab displays exceptions and errors that occurred during the extraction process. The purpose of this analyzing overview is to make it easier to identify faulty or incomprehensible behavior during a process. Up to 10 entries from the extraction_process_log database table are displayed per page. Depending on the error_level, the entries are displayed in different colors. The goal of displaying the fields Message, Timestamp, Url,


Previous Page Page 1 Next Page

Message	Timestamp	Url	Extraction Process	Extraction Step
api/HttpTrigger_GoogleKnowledgeGraphRetrieval responded with status_code: 404 b'Google Knowledge Graph could not find a corresponding Entity'	Aug. 2, 2022, 1:13 p.m.	https://www.goodreads.com/book/show/226646.Advances_in_Behavioral_Finance	Goodreads - Movie Extraction Process	3
Extraction Process could not find any data.	July 15, 2022, 1:13 p.m.	https://footystats.org/players/serbia/aleksandar-mitrovic	footystats - players extraction process	None
api/HttpTrigger_HtmlPropertyValueListExtractor responded with status_code: 400 b'Exception: TypeError: expected string or bytes-like object on instantiation of BeautifulSoup object. Input data = []'	July 15, 2022, 1:13 p.m.	https://footystats.org/players/serbia/aleksandar-mitrovic	footystats - players extraction process	3

Figure 5.13: Snippet of the Extraction Process Logs Page

Extraction Process and Extraction Step is to inform the administrator as precisely as possible where an error has occurred. The Figure 5.13 shows a snippet from the Logs page where the latest 3 log entries are displayed.

6 Illustrative Use Case

In this chapter a complete application example which automatically extracts structured information from a group of Web pages, is explained. We discuss how the extraction process for the target pages is configured, the URL Crawler is set and the result looks in the end.

6.1 Target Web Pages

In this illustrative example we look at a part of the pages published by the internet Movie Database (IMDB). Specifically, we will extract data from the Web pages that show collected information about a particular movie, TV series or TV episode. These pages are accessible via unique URLs. Moreover, these URLs have a similar structure and we can therefore easily distinguish them from other Web pages on the domain imdb.com. All URLs start with the following string: `https://www.imdb.com/title/tt`. Directly after this string comes the internal id of the respective entry which is simply an incremental integer number.
The three figures 9.1, 9.2 and 9.3 in the appendix show an exemplary representation of such a page. In this example, information about the film "The Shawshank Redemption" is shown. The information which is directly related to the movie is displayed very differently on the packed Web page. In addition, much of the information is not related to this film but is used for the navigation on the website or for the promotion of other articles. All pages on imdb.com about movies, TV series and episodes are structured similarly to this example. However, there are also pages where some information is not available and therefore certain sections are not displayed.
We will use the implemented extraction and modification services to create a process that extracts data that is already in a machine-readable form and information that is contained in different semi-structured HTML element arrangements.

6.2 Extraction Process Configuration

Here the extraction process is explained step by step as it is created in the front-end dashboard. The process is intended for the group of Web pages defined in Chapter 6.1. A total of nine steps are performed to extract a wealth of information directly related to the entity, whether it is a movie, a TV series or episode, and create structurally consistent data objects.
The following tables show for each subsequent process step which resource extraction or modification service is called. Additionally the configured input step, input parameters

and result type is specified. Furthermore, a short description describes what task the defined process step fulfils.

Step 1	
Function	DefaultRessourceCollector
Input step	0 - URL
Input parameter	
Result type	interim
Description	The Web page and its contents are requested.

Table 6.1: Illustrative Example: Step 1

Step 2	
Function	JsonLDExtractor
Input step	Step 1
Input parameter	
Result type	final
Description	Extracts JSON-LD serialized data object

Table 6.2: Illustrative Example: Step 2

Step 3	
Function	HtmlSectionExtractor
Input step	Step 1
Input parameter	tag section search_params "data-testid":"Details"
Result type	interim
Description	Extracts HTML "Details" section

Table 6.3: Illustrative Example: Step 3

Step 4	
Function	HtmlPropertyValueListExtractor
Input step	Step 3
Input parameter	section_tag: li section_search_params: ipc-metadata-list__item property_search_params: ipc-metadata-list-item__label value_search_params: ipc-inline-list__item
Result type	final
Description	Extracts the information from the "Details" section

Table 6.4: Illustrative Example: Step 4

Step 5	
Function	GoogleKnowledgeGraphRetriever
Input step	Step 2
Input parameter	query: name type: @type return_values: ["@id"]
Result type	final
Description	Adds the Google Knowledge Graph id to the data object

Table 6.5: Illustrative Example: Step 5

Step 6	
Function	HtmlSectionExtractor
Input step	Step 1
Input parameter	tag: section search_params: "data-testid":"title-cast"
Result type	interim
Description	Extracts HTML "Top cast" section

Table 6.6: Illustrative Example: Step 6

Step 7		
Function	HtmlListOfValuesExtractor	
Input step	Step 6	
Input parameter	property_name value_section_tag value_section_search_params value_tag value_search_params	actor div "data-testid":"title-cast-item" a "data-testid":"title-cast-item__actor"
Result type	final	
Description	Extracts the information from the "Top cast" section	

Table 6.7: Illustrative Example: Step 7

Step 8		
Function	HtmlSectionExtractor	
Input step	Step 1	
Input parameter	tag search_params	section "data-testid":"TechSpecs"
Result type	interim	
Description	Extracts HTML "Technical specs" section	

Table 6.8: Illustrative Example: Step 8

Step 9		
Function	HtmlPropertyValueListExtractor	
Input step	Step 8	
Input parameter	section_tag section_search_params property_tag property_search_params value_tag value_search_params	li ipc-metadata-list__item span ipc-metadata-list-item__label div ipc-metadata-list-item__content-container
Result type	final	
Description	Extracts the information from the "Technical spec" section	

Table 6.9: Illustrative Example: Step 9

6.3 Pattern - Process Assignment

We have already explained in the Section 6.1 how the URLs, of the group of resources which are relevant for us in this example, are structured. The corresponding regular expression for the pattern - process assignment is defined as this:

```
1 (http:\/\/|https:\/\/)(www\.|)(imdb\.com\/title\/tt)\d*\/?$
```

The regular expression matches with URLs that use the HTTP or the HTTPS protocol. Followed by the protocol comes the optional string "www.". Then comes the string "imdb.com/title/tt" which contains the domain name and the custom route for the target Web pages. At the end of the character string, any number of single digits are allowed. This is defined by the token "d*" which only allows numbers. The last dollar sign signifies the end of the String.

6.4 URL Crawler Configuration

Since the target Web pages always link to other movies and TV series, we can use the same regular expression for the URL Crawler configuration as for the pattern - process assignment. This way the URL Crawler can find all relevant resources autonomously. Therefore, we set for the extraction_pattern as well as for the link_discovery_pattern also the following regular expression:

```
1 (http:\/\/|https:\/\/)(www\.|)(imdb\.com\/title\/tt)\d*\/?$
```

6.5 Extracted Data

The extracted information results in a composite data object, which gets stored in the CosmosDB document store as a JSON serialized document. Several attributes are attached additionally to the resulting document. The attribute "source", which contains the URL of the target Web page. The attribute "timestamp", which contains the time when the extraction process was performed. The attribute "data", which contains all extracted information. The attribute "_id", which is automatically added by the CosmosDB and is unique for the respective document.

In Figure 6.1 you can see the result of the extraction process from the Web page on imdb.com of the movie "The Shawshank Redemption". A large part of the data is from the information provided by the JSON-LD encoded object, such as the attributes "@context", "@type", "url", "name", "image" and "description". Properties like "Release date", "Country of Origin", "Official Sites"and "Language" are extracted from the "Details" section. Under "actors" persons are listed which are mentioned in the JSON-LD object on the one hand and those which were extracted from the "Top cast" area on the other hand. "Runtime", "Color", "Sound Mix" and "Aspect ratio" were extracted from the "Technical specs" section. The attribute "@id" is a unique id from the Google Knowledge Graph which refers to the same entity, namely the movie "The Shawshank Redemption".

```
{
    source: https://www.imdb.com/title/tt0111161,
    timestamp: "2022-09-12 16:56:14",
    data: {
        @context: https://schema.org,
        @type: "Movie",
        url: "/title/tt0111161/",
        name: "The Shawshank Redemption",
        image: https://m.media-amazon.com/images/M/MV5BMDFkYTc0MGEtZmNhMC00ZDIzLWFmNTEtODM1ZmRlYWMwM
        description: "Two imprisoned men bond over a number of years, finding solace and eventual re
      ▶ review: {  8 items  },
      ▶ aggregateRating: {  5 items  },
        contentRating: "12",
      ▼ genre: [
            "Drama"
        ],
        datePublished: "1995-03-02",
        keywords: "prison,based on the works of stephen king,escape from prison,voice over narratio
      ▶ trailer: {  9 items  },
      ▶ actor: [  21 items  ],
      ▶ director: [  1 item  ],
      ▶ creator: [  3 items  ],
        duration: "PT2H22M",
        Release date: "March 2, 1995 (Netherlands)",
        Country of origin: "United States",
      ▶ Official sites: [  2 items  ],
        Language: "English",
        Also known as: "De ontsnapping",
        Filming locations: "127A Smithfield Road, Frederiksted, Virgin Islands",
        Production company: "Castle Rock Entertainment",
        @id: "kg:/m/07jnt",
        Runtime: "2 hours 22 minutes",
        Color: "Color",
        Sound mix: "Dolby Digital",
        Aspect ratio: "1.85 : 1"
    },
    _id: "631f48ac90a1d61a631907f5"
}
```


Figure 6.1: Extraction result of the resource `https://www.imdb.com/title/tt0111161`

7 Evaluation

After explaining the system and its functionality in detail, we evaluate the system in terms of flexibility and durability. In this context, we address the questions that were defined in the evaluation design in Chapter 3.3. In the evaluation design, we have asked three questions each about the flexibility and the durability of the system. We want to answer these questions on the basis of the conducted experiments, which are described in the following chapters.

7.1 Flexibility Evaluation

To illustrate the flexibility of our system, we have created 11 different extraction processes, each made for a specific group of resources. Table 7.1 lists all the extraction and modification services which were used for creating these extraction processes. Table 7.2 lists

Number	Service Name	total times used
1	DefaultResourceCollector	11
2	HtmlSectionExtractor	13
3	JsonLDExtractor	3
4	RDFaExtractor	1
5	MicrodataExtractor	4
6	HtmlConsecutivePropertyValuePairsExtractor	1
7	HtmlListOfValuesExtractor	7
8	HtmlPropertyValueListExtractor	4
9	HtmlSinglePropertyExtractor	4
10	ManualJsonObjectCreator	1
11	GoogleKnowledgeGraphRetriever	9
12	PropertyMapping	2

Table 7.1: Enumeration of Extraction and Modification Services as a Supplement to Table 7.2

the 11 extraction processes which were created for this experiment. To categorize them, the respective domains and entity types are shown. The processes extract information about movies, books, authors, actors, athletes and events. The table also shows which extraction and modification services are used for this purpose. The first extraction process, for example, refers to Web pages about movies on the domain *rottentomatoes.com*. During this data extraction, the *DefaultResourceCollector*, *HtmlListOfValuesExtractor*,

		Extraction and modification services											
Target domain	Target entity	1	2	3	4	5	6	7	8	9	10	11	12
rottentomatoes.com	movie	1	2	0	0	0	0	1	1	0	0	1	2
moviepilot.de	movie	1	0	0	0	1	0	0	0	0	0	1	0
moviepilot.de	person	1	2	0	0	1	0	2	0	0	0	1	0
imdb.com	movie	1	3	1	0	0	0	1	2	0	0	1	0
imdb.com	person	1	2	1	0	0	0	2	0	0	0	1	0
goodreads.com	person	1	1	0	0	1	0	1	0	0	0	1	0
goodreads.com	book	1	0	0	0	1	0	0	0	0	0	1	0
bellabooks.com	book	1	2	0	0	0	0	0	1	2	1	1	0
fbref.com	person	1	0	1	0	0	0	0	0	0	0	1	0
soccerment.com	person	1	1	0	0	0	1	0	0	2	0	0	0
events.at	event	1	0	0	1	0	0	0	0	0	0	0	0

Table 7.2: Listing of Extraction Processes and their use of Extraction and Modification Services

HtmlPropertyValueListExtractor and *GoogleKnowledgeGraphRetriever* are each called once. In addition, the *HtmlSectionExtractor* and the *PropertyMapping* service are used twice each. For better readability, the extraction and modification services are represented with a number from 1 to 12.

Next, we want to answer the questions regarding the flexibility of the system with the experiences we have made during the creation and execution of the presented processes.

1. *Which information can we extract from the Web resources?*
   We can theoretically extract all kinds of information from the Web. With the processes we examined, we can already show that a lot of differently structured data can be collected by extracting a wide variety of data representations. However, our application area is limited to Web pages where the information can be assigned to a specific entity. Theoretically and practically we could extract data from pages that cannot be assigned to a specific entity using the existing toolset, but this would no longer fit conceptually with other datasets.
   In essence, extracted data can come from machine-readable data sources like JSON-LD, HTML elements tagged with markup standards like Microdata or RDFa, or from very diverse HTML structures, so-called semi-structural data sources. Due to the fact that a large amount of different extraction methods are available and that they are customizable to the respective use case, we can create tailor-made processes for different websites without having to write a single line of code.

2. *What is the quality of the data in terms of completeness and comparability with similar data entities?*
   The completeness of the extracted data with regard to the available information

depends on the extraction methods which the system offers and how the extraction process has been configured.
The internal structure of the data can be identified especially well if the content publisher already provides the source data by using markup formats or by providing the data directly in a structured format. This provides a perfect starting point for obtaining good data quality. The extraction services are designed to be quite generic so that they can be used for as many extraction processes as possible. However, if the information is provided in a very specific and unique way, it is difficult to extract the data with its accurate structure by using such generic approaches.
In order to compare data sets, it must be possible to identify which entries actually describe the same entity. This can be done, for example, if certain attributes are present in two entries and these have the same or similar values. To improve the entity association, we have created a way to add an entity ID to the data object, which we get from the Google Knowledge Graph. This simplifies the process of identifying similar entities. In addition, we have created a way to customize property names with the property mapping service in order to maintain a consistent attribute naming schema. As a result, names of certain attributes on different Web pages can be changed to a specifically defined name, which in turn improves the data's comparability considerably.

3. *Which data could not be extracted or was extracted incorrectly? Why is this happening and how could it be improved?*
   During the creation of the extraction processes, we repeatedly encountered sources of information that were difficult to process. This is mainly due to the fact that Web pages are structured quite differently and therefore the information is displayed in the most diverse ways. With the existing extraction services, a large amount of resources can be processed, but this still only amounts to a fraction of the freely accessible Web. There are more powerful methods to extract semi-structural data. However, these approaches often need to be trained in advance or their configuration is more complex. Since we wanted to keep the process creation fairly simple, we decided to create services which are easy to configure. Thanks to the modular and flexible structure of the extraction framework, the range of functions can be expanded through the introduction of new extraction methods at any time.

In summary, the presented system can extract information from many different data sources and store them in a consistent data format. It is clear that this version cannot be a universal tool for all application areas. However, due to the flexible system architecture and the way it works, it is very easy to extend the set of functions in such a way that the range of applications is increased. The functional scope can be extended by creating additional extraction or modification services without having to change anything at all in any of the core services. Important process steps and settings can be configured by manipulating database entries or environment variables, which creates an extremely flexible and modifiable system. The system can thus be used as a foundational framework for more advanced applications.

7.2 Durability Evaluation

To test the robustness of the system, the URL Crawler has been configured to automatically search for a specific group of Web pages and subsequently extract structured data from them. We concentrated on the group of Web pages we referred to in the illustrative use case in Chapter 6 and configured the extraction process, the pattern-process assignment as well as the URL Crawler accordingly.
We have given the system one week from July 6, 2022, 14:00 to July 13, 2022, 14:00 to autonomously identify and process the relevant target Web pages. In this time, the URL Crawler has found over 1.5 million Web pages. Of these pages, about 105 000 resources were processed, the data extracted and stored in the document store. With the knowledge gained in this process we now want to answer the questions we asked in the evaluation design regarding the durability of the system.

1. *How many Web pages can be processed in a specified period of time?*
   The URL Crawler was configured so that it is called every 10 seconds. With a batch size of six, it processes up to six Web pages per function call. For every function call, relevant Web pages which are linked on the page are identified and registered; this initiates the extraction process for up to six Web pages. With optimal processing, over 360 000 Web pages should be processed within a week. However, in our experiment, we were only able to process about 105 000 pages due to a resource bottleneck in the url-crawler-db database.

2. *How high is the utilization of individual resources in the process with regard to storage capacities and the execution environment?*
   The cloud service model we use for the SQL databases limits the data access capacity and storage space to 50 Database Transaction Units (DTUs). These DTUs are a unit of capacity invented by Amazon. This means we are using a capacity set that is intended for application development but not for a professional enterprise production environment. Nevertheless, the memory size was sufficient for our purposes. We also had no problems accessing the dataextractiondb (see Figure 7.1). However, we quickly reached the processing capacity limit for the urlcrawlerdb (see Figure 7.2). Because the table links_to_crawl filled up very quickly and contained over 1.5 million entries in the end, each query was very time-consuming as it took a considerable long time to process. Due to this delay, the entire data extraction process was subsequently delayed, resulting in the large discrepancy between the theoretically possible number of processed resources and the practically realized number in this trial.

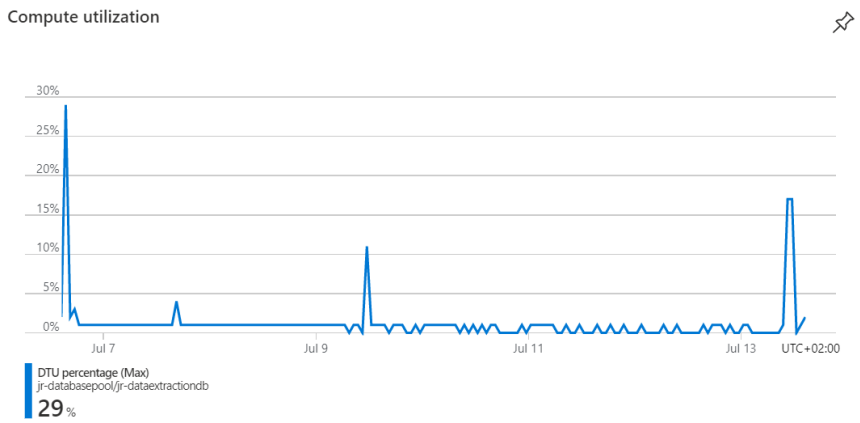

Figure 7.1: Compute Utilization of the dataextractiondb

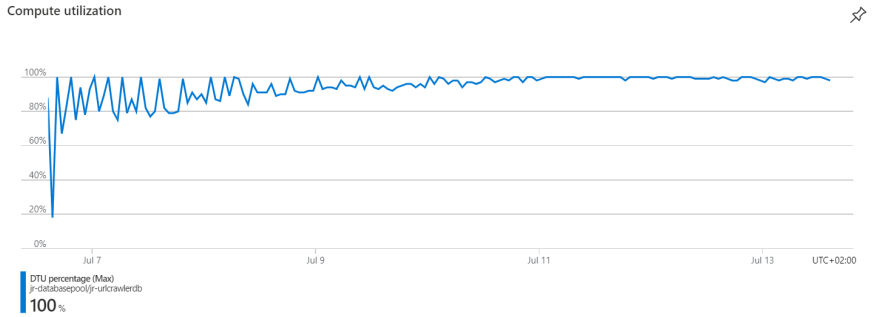

Figure 7.2: Compute Utilization of the urlcrawlerdb

We also analyzed how often an Azure function was called during our experiment (see Figure 7.3). We can see that the number of calls per time unit decreases over the length of this experiment. This is due to the fact that the more entries were stored in the links_to_crawl database table, the less extraction processes were initiated due to the delay on accessing the overloaded table. Incidentally, there is no risk in making too many requests with regard to the Azure functions, because when the request frequency increases, several instances of the respective functions are simply created dynamically up to a certain limit depending on the chosen pricing model.

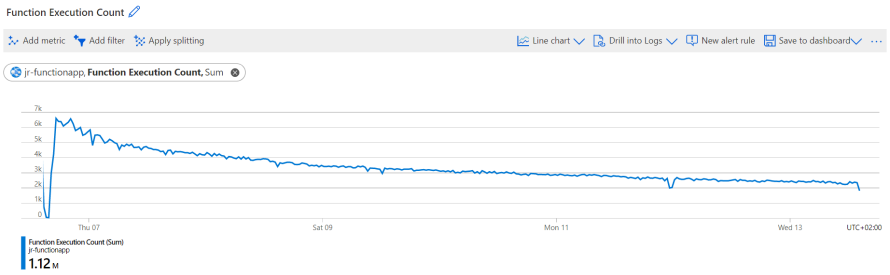

Figure 7.3: Execution Count of all Azure Functions

Finally, we analyzed the computational utilization of the CosmosDB database, which stores and updates the resulting data-documents, and found that no problems were detected during the experiment.

3. *What are the resource bottlenecks and how can performance be improved in terms of the resources and technologies used?*
   In our experiment we could clearly see that a bottleneck can quickly emerge in the links_to_crawl database table. If we would now extend the scope of our experiment to a much larger number of Web resources from many different domains this would lead to an even bigger problem. To solve this issue, we could buy more computation resources in the form of DTUs or use another storage medium for the management of the links to crawl. Some No-SQL databases, for example, can perform queries on such large amounts of data much more efficiently. The challenges that have been identified during this experiment can therefore certainly be overcome.

8 Conclusion

8.1 Summary

In brief, this thesis deals with the problem of data extraction from diverse Web resources. While first highlighting the significance of this issue, it also touches on the difficulties which are encountered when developing a software solution. Subsequently, close attention was paid to the tasks required for the automated extraction of data from Web pages. State-of-the-Art approaches on Web crawling and information extraction were discussed. Furthermore, ways to improve the comparison between different datasets on the same entities were assessed. Based on this analysis, requirements for the development of a flexible framework for data extraction on a broad scale were identified. Thereby it was important that the system can be used very flexibly and is easily extendable. Furthermore, the system's working quality is ensured by meeting modern software development requirements and hosting it in a scalable cloud environment. Fulfilling all of these criteria, a prototype was developed in the course of this work. The system's architecture and the functionality of the individual components were described here to facilitate understanding of the entire information extraction process. The technologies used were explained and some of their unique characteristics were discussed in more detail.
The developed system was then evaluated in terms of its flexibility by creating an extraction process for eleven different Web resource groups. As a result, information was extracted from semi-structured sections within HTML pages, HTML elements which were annotated with markup standards like RDFa or Microdata, and websites which provided serialized data in the JSON-LD format. This demonstrated the system's application to a wide variety of websites which publish information about a broad range of different entities. Furthermore, a durability test was carried out to simulate the system's response to a large amount of workload. The result showed that the use of alternative database technologies could possibly increase the efficiency of the system.
In summary, a highly versatile system was developed which can extract a large amount of information from various websites with little configuration effort. The developed framework can therefore be used for the successful collection of published information from different content providers for the purpose of data comparison of similar entities.

8.2 Future Work

The information extraction framework developed in this thesis provides a solid foundation for processing a diverse set of Web resources. However, there is some room for improvement and many possibilities to extend the system. During the evaluation of the system, it was

discovered that a SQL database with limited processing capacities would quickly reach its limits when it comes to managing the Web pages to be visited by the URLCrawler. By using an alternative database technology, the access speed could be optimised and thus, a more efficient operation of the system could be guaranteed.
The proposed framework bears great potential for development in terms of the data representations to be processed. It is already possible to extract some common data representations in semi-structured HTML layouts. However, there are many other representations that cannot yet be captured with the developed services. There are also many other data representation formats that have not been addressed in this project. A possible extension would be, for example, the processing of plain text in which the HTML elements used do not indicate any meaning of the content. Furthermore, multimedia information such as images and videos could be processed to extract structured data. The framework presented in this paper serves as an ideal starting point for thinking about such extensions and the system's design already leaves room for such expansions.

Bibliography

[1] Hannes Mühleisen and Christian Bizer. Web Data Commons - extracting Structured Data from two large Web Corpora. In *LDOW*, 2012.

[2] Maurice de Kunder. Estimated size of the World Wide Web. `https://www.worldwidewebsize.com/`, July 2022.

[3] Muhammd Jawad Hamid Mughal. Data Mining: Web Data Mining Techniques, Tools and Algorithms: An Overview. *International Journal of Advanced Computer Science and Applications*, 9(6), 2018.

[4] Tim Berners-Lee, Robert Cailliau, Ari Luotonen, Henrik Frystyk Nielsen, and Arthur Secret. The World-Wide Web. *Communications of the ACM*, 37(8):76–82, August 1994.

[5] Tim Berners-Lee. Universal Resource Identifiers in WWW, 1994.

[6] Tim Berners-Lee and Dan Connolly. Hypertext Markup Language-2.0. Technical report, 1995.

[7] Mark Frauenfelder. A smarter Web. *TECHNOLOGY REVIEW-MANCHESTER NH-*, 104(9):52–59, 2001.

[8] RDF Working Group. Resource Description Framework (RDF). `https://www.w3.org/RDF/`.

[9] RDFa Working Group. RDF in Attributes (RDFa). `https://www.w3.org/2001/sw/wiki/RDFa`.

[10] Gavin Carothers and Eric Prud'hommeaux. RDF 1.1 Turtle. `https://www.w3.org/TR/2014/REC-turtle-20140225/`, February 2014. https://www.w3.org/TR/2014/REC-turtle-20140225/.

[11] Gregg Kellogg, Manu Sporny, and Markus Lanthaler. JSON-LD 1.0. `https://www.w3.org/TR/2020/SPSD-json-ld-20201103/`, November 2020. https://www.w3.org/TR/2020/SPSD-json-ld-20201103/.

[12] Pascal Hitzler. A Review of the Semantic Web Field. *Communications of the ACM*, 64(2):76–83, 2021.

[13] Oren Etzioni. The World-Wide Web: quagmire or gold mine? *Communications of the ACM*, 39(11):65–68, November 1996.

[14] X. Yuan, M.H. MacGregor, and J. Harms. An efficient Scheme to remove Crawler Traffic from the Internet. In *Proceedings. Eleventh International Conference on Computer Communications and Networks*, pages 90–95, 2002.

[15] Elda Xhumari and Izaura Xhumari. A Review of Web Crawling Approaches. In *RTA-CSIT*, 2021.

[16] Soumen Chakrabarti, Martin van den Berg, and Byron Dom. Focused Crawling: a new Approach to Topic-specific Web Resource Discovery. *Computer Networks*, 31(11):1623–1640, 1999.

[17] Michelangelo Diligenti, Frans Coetzee, Steve Lawrence, C Lee Giles, and Marco Gori. Focused Crawling using Context Graphs. In *VLDB*, 2000.

[18] Soumen Chakrabarti, Kunal Punera, and Mallela Subramanyam. Accelerated Focused Crawling through Online Relevance Feedback. In *Proceedings of the 11th international conference on World Wide Web*, pages 148–159, 2002.

[19] DOM. `https://www.w3.org/DOM/`, June 2022.

[20] Microformats. `http://microformats.org/`.

[21] Ian Hickson, Chaals Nevile, and Dan Brickley. HTML Microdata. `https://www.w3.org/TR/2021/NOTE-microdata-20210128/`, January 2021. https://www.w3.org/TR/2021/NOTE-microdata-20210128/.

[22] Common Crawl Foundation. Common Crawl. `https://commoncrawl.org/`, 2022.

[23] Robert Meusel, Petar Petrovski, and Christian Bizer. The Webdatacommons Microdata, RDFa and Microformat Dataset Series. In *International Semantic Web Conference*, pages 277–292. Springer, 2014.

[24] Adanma Cecilia Eberendu et al. Unstructured Data: an Overview of the Data of Big Data. *International Journal of Computer Trends and Technology*, 38(1):46–50, 2016.

[25] Nicholas Kushmerick. *Wrapper Induction for Information Extraction*. PhD thesis, 1997. Copyright - Database copyright ProQuest LLC; ProQuest does not claim copyright in the individual underlying works; Last updated - 2021-09-20.

[26] Ion Muslea, Steven Minton, and Craig Knoblock. A Hierarchical Approach to Wrapper Induction. 09 1999.

[27] Stephen Soderland. Learning Information Extraction Rules for Semi-Structured and Free Text. *Machine Learning*, 34(1/3):233–272, 1999.

[28] Chia-Hui Chang, Chun-Nan Hsu, and Shao-Cheng Lui. Automatic Information Extraction from Semi-structured Web Pages by Pattern Discovery. *Decision Support Systems*, 35(1):129–147, 2003.

[29] Andrew Carlson and Charles Schafer. Bootstrapping Information Extraction from Semi-structured Web Pages. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 195–210. Springer, 2008.

[30] Rajasekar Krishnamurthy, Yunyao Li, Sriram Raghavan, Frederick Reiss, Shivakumar Vaithyanathan, and Huaiyu Zhu. SystemT: a System for declarative Information Extraction. *ACM SIGMOD Record*, 37(4):7–13, March 2009.

[31] Stefan Dipl.-Ing. (FH) Luber. Was ist BERT? `https://www.bigdata-insider.de/was-ist-bert-a-1116088/`, May 2022.

[32] Ruixue Zhang, Wei Yang, Luyun Lin, Zhengkai Tu, Yuqing Xie, Zihang Fu, Yuhao Xie, Luchen Tan, Kun Xiong, and Jimmy Lin. Rapid Adaptation of BERT for Information Extraction on Domain-Specific Business Documents. 2020. Publisher: arXiv Version Number: 1.

[33] Hong-Hai Do. Schema Matching and Mapping-based Data Integration. 2006.

[34] Schema.org. `https://schema.org/`, August 2022.

[35] Ramanathan V Guha, Dan Brickley, and Steve Macbeth. Schema.org: Evolution of Structured Data on the Web. *Communications of the ACM*, 59(2):44–51, 2016.

[36] Kody Moodley, Josef Hardi, John Graybeal, Mark A Musen, and Michel Dumontier. Assessing Schema. org's Coverage of Terms from Key Biomedical Datasets. In *Bio-Ontologies*, 2018.

[37] Nuno Freire, Valentine Charles, and Antoine Isaac. Evaluation of Schema.org for Aggregation of Cultural Heritage Metadata. In *European Semantic Web Conference*, pages 225–239. Springer, 2018.

[38] Boran Taylan Balcı, Umutcan Şimşek, Elias Kärle, and Dieter Fensel. Analysis of Schema.org Usage in the Tourism Domain. *arXiv preprint arXiv:1802.05948*, 2018.

[39] Peter Kalchgruber, Wolfgang Klas, Nour Jnoub, and Elaheh Momeni Roochi. FactCheck - Identify and Fix Conflicting Data on the Web. In *18TH INTERNATIONAL CONFERENCE ON WEB ENGINEERING JUNE, 2018*, 18TH INTERNATIONAL CONFERENCE ON WEB ENGINEERING, pages 5–8, June 2018.

[40] Stack Overflow. 2021 Stack Overflow Developer Survey. `https://insights.stackoverflow.com/survey/2021`.

[41] P M Mell and T Grance. The NIST Definition of Cloud Computing. Technical Report NIST SP 800-145, National Institute of Standards and Technology, Gaithersburg, MD, 2011. Edition: 0.

[42] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Anish Karmarkar, and Yves Lafon. SOAP, April 2007.

[43] R. Fielding and J. Reschke. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. RFC 7230, RFC Editor, June 2014. `http://www.rfc-editor.org/rfc/rfc7230.txt`.

[44] Jean Paoli, Tim Bray, Eve Maler, Michael Sperberg-McQueen, and François Yergeau. Extensible Markup Language (XML) 1.0 (Fifth Edition). W3C recommendation, W3C, November 2008. https://www.w3.org/TR/2008/REC-xml-20081126/.

[45] Hugo Haas, David Orchard, Christopher Ferris, Eric Newcomer, David Booth, Francis McCabe, and Mike Champion. Web Services Architecture, February 2004. https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/.

[46] Martin Helmich. RESTful Webservices (1): Was ist das überhaupt? `https://www.mittwald.de/blog/webentwicklung-design/restful-webservices-1-was-ist-das-uberhaupt`, March 2013.

[47] Roy T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*, chapter 10. 2000.

[48] James Lewis and Martin Fowler. Microservices. `https://martinfowler.com/articles/microservices.html`, March 2014.

[49] Devndra Ghimire. *Comparative Study on Python Web Frameworks: Flask and Django.* PhD thesis, May 2020.

[50] Christof Strauch, Ultra-Large Scale Sites, and Walter Kriha. NoSQL Databases. *Lecture Notes, Stuttgart Media University*, 20(24):79, 2011.

[51] NoSQL-Datenbank: Was ist NoSQL? `https://azure.microsoft.com/de-de/resources/cloud-computing-dictionary/what-is-nosql-database/`, 2022.

[52] Informationen zu NoSQL-Datenbanken. `https://www.mongodb.com/de-de/nosql-explained`, 2022.

[53] John Roach. Microsoft's Virtual Datacenter Grounds 'the cloud' in Reality. `https://news.microsoft.com/innovation-stories/microsofts-virtual-datacenter-grounds-the-cloud-in-reality/`, April 2021.

[54] Azure Products. `https://azure.microsoft.com/en-us/products/`, August 2022.

[55] Azure Service Bus Messaging Documentation. `https://docs.microsoft.com/en-us/azure/service-bus-messaging/`, August 2022.

[56] Azure Cosmos DB Documentation. `https://docs.microsoft.com/en-us/azure/cosmos-db/`, August 2022.

[57] MongoDB Documentation. `https://www.mongodb.com/docs/`, August 2022.

[58] MongoDB Wire Protocol. https://www.mongodb.com/docs/manual/reference/mongodb-wire-protocol/, August 2022.

[59] Azure Functions Documentation. https://docs.microsoft.com/en-us/azure/azure-functions/, August 2022.

[60] Azure Event Hubs Documentation. https://docs.microsoft.com/en-us/azure/event-hubs/, August 2022.

[61] Azure Event Grid Documentation. https://docs.microsoft.com/en-us/azure/event-grid/, August 2022.

[62] Timer Trigger for Azure Functions. https://docs.microsoft.com/en-us/azure/azure-functions/functions-bindings-timer?tabs=in-process&pivots=programming-language-python, August 2022.

[63] Aziz Atif. NCrontab: Crontab for .NET. https://github.com/atifaziz/NCrontab, 2021.

[64] Azure Functions HTTP Triggers and Bindings Overview. https://docs.microsoft.com/en-us/azure/azure-functions/functions-bindings-http-webhook, August 2022.

[65] Azure Service Bus Bindings for Azure Functions. https://docs.microsoft.com/en-us/azure/azure-functions/functions-bindings-service-bus, August 2022.

[66] Azure SQL Database Documentation. https://docs.microsoft.com/en-us/azure/azure-sql/database/, August 2022.

[67] App Service Documentation. https://docs.microsoft.com/en-us/azure/app-service/, August 2022.

[68] Kenneth Reitz. Python Requests. https://pypi.org/project/requests/, June 2022.

[69] Leonard Richardson. Python Beautifulsoup 4. https://pypi.org/project/beautifulsoup4/, April 2022.

[70] Ivan Herman. Python pyRdfa. https://pypi.org/project/pyRdfa/, August 2010.

[71] Martin Blech. Python xmltodict. https://pypi.org/project/xmltodict/, May 2022.

[72] Ed Summers. Python microdata. https://github.com/edsu/microdata, February 2022.

[73] Mark Otto and Jacob Thornton. Bootstrap 5.1.3. https://github.com/twbs/bootstrap, October 2021.

9 Appendix

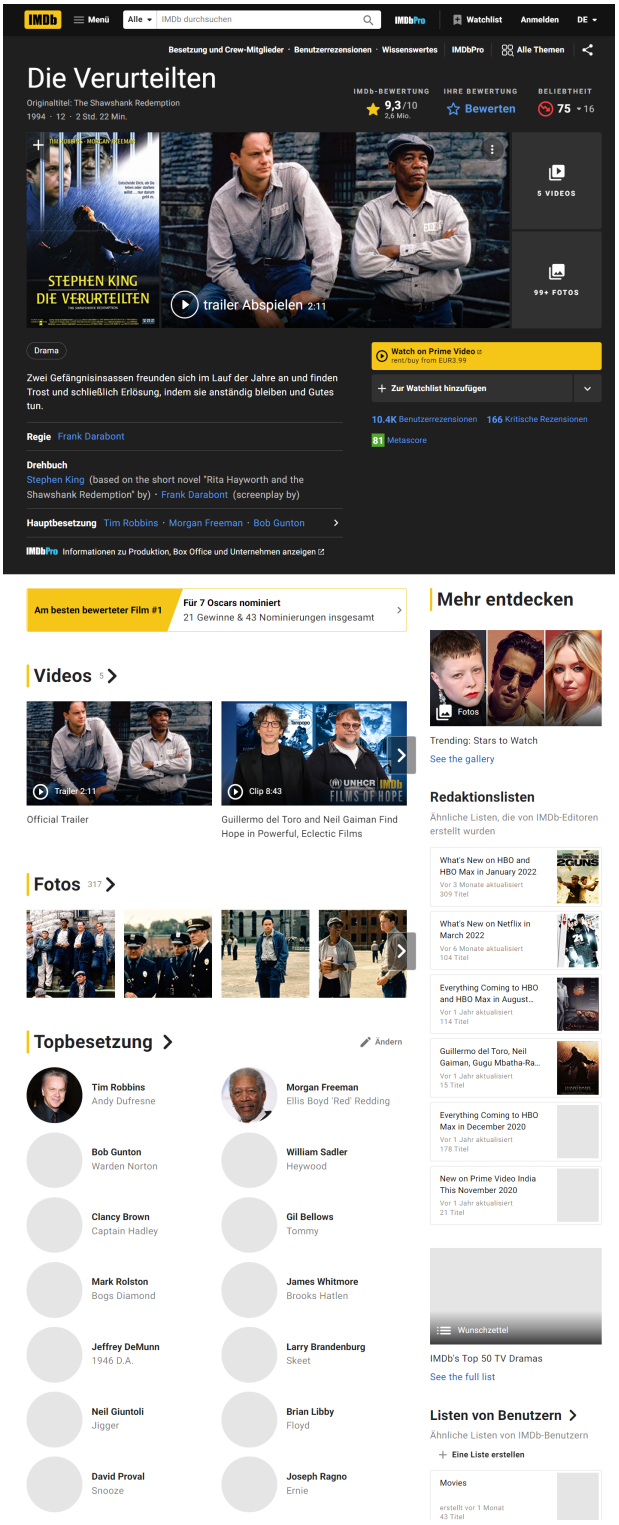

Figure 9.1: First Part of the Web Page: `https://www.imdb.com/title/tt0111161`

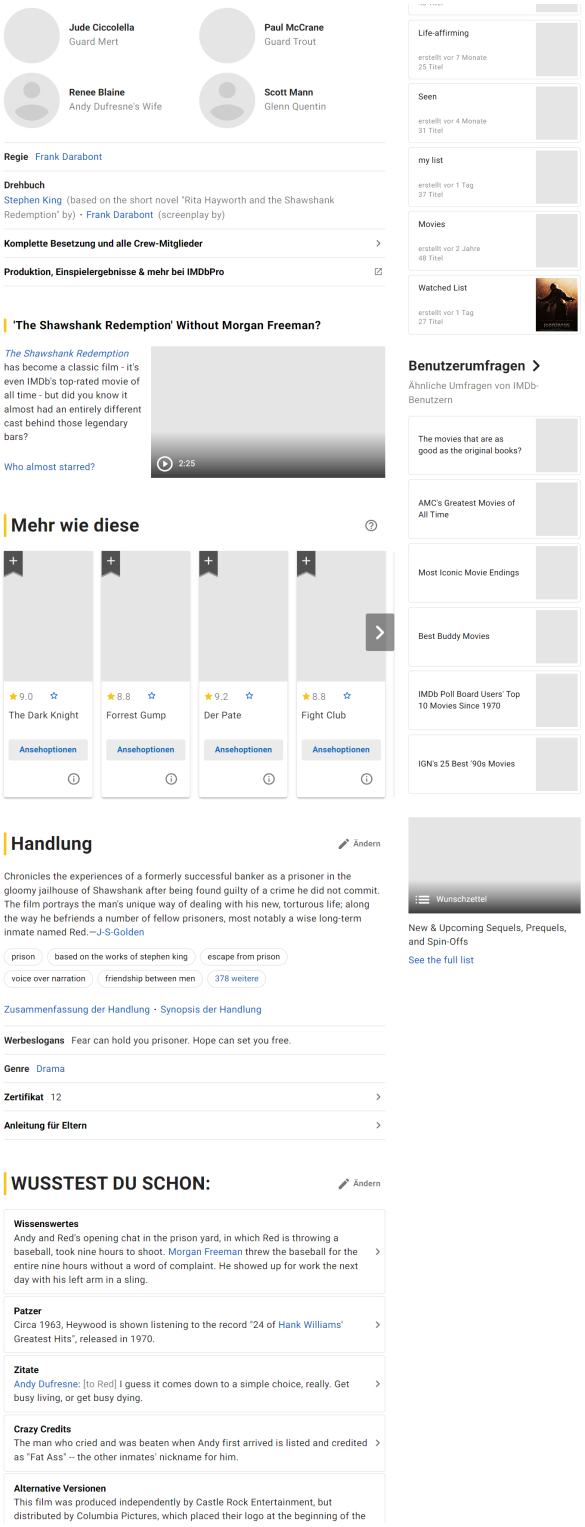

Figure 9.2: Second Part of the Web Page: `https://www.imdb.com/title/tt0111161`

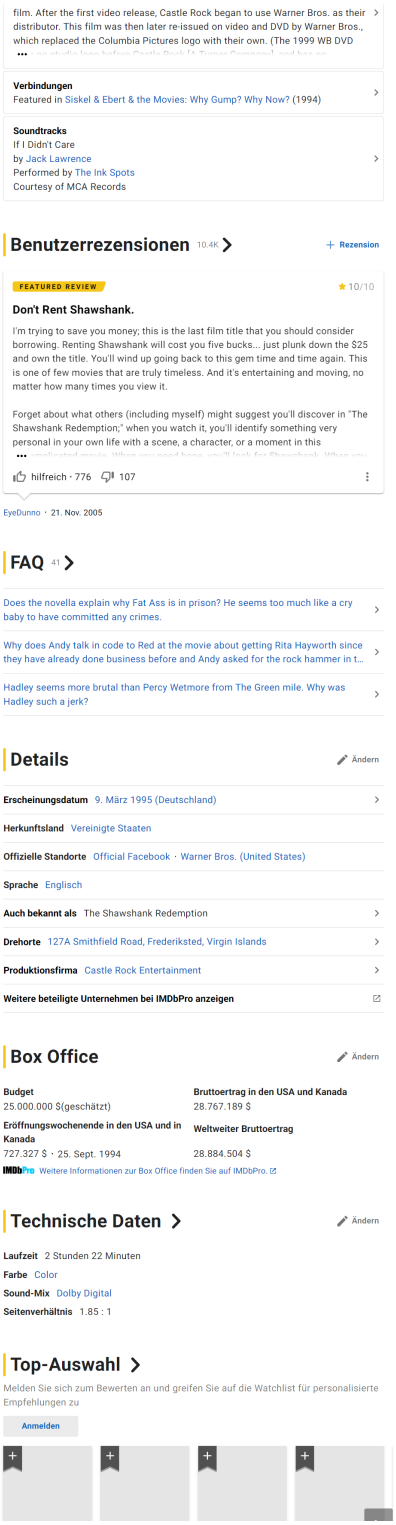

Figure 9.3: Third Part of the Web Page: `https://www.imdb.com/title/tt0111161`