Bachelor Thesis

A web-based interface for DUUI

Cedric Borkowski

March 18, 2024

written at the
Department of Text Technology Lab
at the Goethe-Universität Frankfurt am Main

Abstract

Natural Language Processing (NLP) for big data requires an efficient and sophisticated infrastructure to complete tasks both fast and correctly. Providing an intuitive and lightweight interaction with a framework that abstracts and simplifies complex tasks assists in reaching this goal. This bachelor thesis extends the NLP framework Docker Unified UIMA Interface (DUUI) by an API and a web-based graphical user interface to control and manage pipelines for automated analysis of large quantities of natural language. The extension aims to reduce the entry barrier into the field as well as to accelerate the creation and management of pipelines according to UIMA standards. Pipelines can be executed in the browser or using the web API directly and then monitored on a document level. The evaluation in usability and user experience indicates that the implementation benefits the framework by making its usage more user friendly, lightweight, and intuitive while also making the management of pipelines more efficient.

Contents

1 Introduction

1.1. Motivation

The automated analysis of natural language in large quantities is a major challenge for existing tools and frameworks for *Natural Language Processing* (NLP) as scalability, performance and usability are essential for big data analysis. As more and more data, be it in form of speech, text, or video, becomes available for processing, the infrastructure has to meet new requirements. With the increase in data also comes the possibility to greatly increase the capabilities of language models, one of the major building blocks for NLP, that can be used in areas like real time translation or processing of text and speech, automated chat bots in customer service, and search engines algorithms to make better and faster suggestions for users. Although frameworks for NLP exist and are in active use, many do not meet the requirements to efficiently handle big data analysis. Reasons include the lack of scalability and flexibility to support a wide range of existing tools, security concerns, the requirement to have proficient knowledge in computer science and NLP itself, as well as being outdated (Leonhardt et al. 2023). To handle the mentioned shortcomings in existing frameworks, the Text Technology Lab presents *Docker Unified UIMA Interface* (DUUI), a new approach to big data analysis using NLP. DUUI is designed in a modular way to allow usage by a wide range of users not limited to experts (Leonhardt et al. 2023, p. 388) and supports the operation of existing and new *Analysis Engine*s (AE) without any need for additional libraries (Leonhardt et al. 2023, p. 390). Though promising, the framework is still in development and not yet fully realized. As of now the barrier of entry is still a concern as knowledge in the programming language Java, Docker, and NLP are required to use the software. This thesis presents a solution that mostly removes this knowledge requirement by providing a web-based *graphical user interface* (GUI), that handles the creation, management, and execution of pipelines with DUUI using a simple online editor. The website communicates via RESTful (Fielding 2000) with an *application programming interface* (API) that handles the usage and storage of pipelines and their AEs. Since the communication is done via a RESTful API, the usage of DUUI is not limited to the browser or Java, but also supports practically any other programming language a user may be familiar with. While a GUI can not replace deep understanding in a subject, it can significantly reduce the learning curve for beginners. By leveraging the performance and capabilities of DUUI in a user-friendly environment, users can benefit from a centralized, secure, and scalable system capable of big data analysis.

1.2. Contributions

With this thesis, first steps towards large scale usability for a wide target audience are taken. To do so, a RESTful API for communicating with DUUI as well as a web-based, platform-independent graphical user interface have been developed. NLP Pipelines can be created, updated, executed, and monitored directly in the browser, improving efficiency and usability. Without the need for programming knowledge, the framework is approachable for users with little or no experience in programming. Additionally, because no setup or installation is required by using the web interface, users can get started quickly. Monitoring and error reporting are important parts of usability which is why the display of status and performance on a document level has been a major focus for this work. The web interface provides an easy way to stay updated on the status per processed document.

1.3. Outline

Chapter 2 introduces fundamental concepts and technologies used in the implementation of both DUUI and its supporting software - i.e, the web interface and RESTful API. Existing frameworks and editors for the creation of automated workflows as well as monitoring solutions are introduced in chapter 3. Afterwards, chapter 4 documents and describes the implementation details for the web API and interface. The evaluation of usability is discussed in chapter 5 followed by the final chapter 6 that explains next possible steps and features for both the API and web interface.

2 Fundamentals

Natural Language Processing and its application in big data analysis is a complex system of frameworks and technologies complementing and enhancing each others capabilities. The following chapter introduces core concepts required as a basis for this thesis and gives a broad overview about auxiliary technologies that have been used.

2.1. Natural Language Processing

Natural Language Processing (NLP) is an application of artificial intelligence and machine learning that deals with the analysis and interpretation of natural or human language by computers. It is used to empower machines to understand, interpret, and generate human language by analyzing text, speech, or video in a scope that is not limited to the purely literal. NLP technologies utilize complex algorithms and machine learning to develop models that can understand and respond to language. These models learn from large amounts of annotated data to recognize patterns and relationships in language. NLP is widely used in various fields, including chat bots such as ChatGPT, automatic translation (e.g. DeepL), information extraction or search engine optimisation. There are different levels of analysis ranging from syntactic or grammatical to semantic analysis that vary in complexity (Liddy 2001). An NLP analysis or task may be used to extract information and gain a deeper understanding of nuances and indirect information as well as the extraction of specific information such as time, place, entities, or author sentiment. Many higher levels of analysis depend on information gained from syntactic or lexical analysis like tokens or sentences (Nadkarni et al. 2011). Tasks are often combined in pipelines, a modular system in which data can be analyzed by multiple tasks in sequence (Nadkarni et al. 2011) and outputs of one task passed to subsequent ones.

Natural language is a subset of unstructured information, a type of information that encompasses a spectrum of data types, ranging from text documents and multimedia files to sensor-generated datasets. Unlike structured data, which can be organized in well-defined formats and relational databases, unstructured information poses distinct challenges owing to its inherent lack of organization. As there is no predefined structure for this type of data, automatic analysis by machines is difficult or even impossible. NLP, machine learning algorithms, and advanced data analytics techniques have proven to be important tools for deciphering patterns and extracting meaningful information from unstructured data sets. To overcome the challenges that result from the lack of structure in natural language, the *Unstructured Information Management Architecture* (Ferrucci and Lally 2004)[1] (UIMA), project was launched by IBM in 2005. Since 2007 the project is managed by Apache and is used to standardize unstructured information in a common format for automatic and efficient machine processing. UIMA defines the *Analysis Engine* (AE) as the central processing unit.

[1]`https://uima.apache.org/`, last accessed 16/02/2024 14:07 MET

An AE annotates its input data and stores these annotations in a *Common Analysis Structure* (CAS) object for further processing by following AEs. The CAS object is a tree like data structure commonly represented in the *Extensible Markup Language* (XML) format, that contains the *subject of analysis* (sofa). The sofa itself contains the data to be analyzed (Ferrucci, Lally, et al. 2009) and all performed annotations - i.e. metadata. The CAS gains great flexibility by the use of a user-defined type system for application specific use cases. Each CAS object is paired with a type system that defines the structure of the data, such as tokens, sentences, or parts of speech. UIMA provides a minimal or base type system that provides frequently occurring and platform-independent types (Ferrucci, Lally, et al. 2009).

2.2. Representation State Transfer

Representation State Transfer (REST) (Fielding 2000) is an architectural style for building networked applications and is used for communication between clients and servers. In REST, resources are identified by a *Uniform Resource Identifier* (URI), and interactions with these resources are performed using standard *Hypertext Transfer Protocol* (HTTP) methods such as GET, POST, PUT and DELETE. These methods correspond to the *CRUD* operations (Create, Read, Update, Delete) that are usually associated with database systems. One of the key features of REST is that each request made by a client to a server must contain all information required to understand and respond to the request. The server does not store any client state between requests. Additionally, REST often employs a representation format, such as *JavaScript Object Notation* (JSON) or *Extensible Markup Language* (XML), to serialize and deserialize data exchanged between clients and servers. JSON is also the preferred form of data transmission through HTTP for the implemented API in section 4.2. The design decisions bring benefits in visibility because requests are self sufficient by providing all necessary information to be fully understood, reliability because errors can be recovered from much easier, and scalability because no resources have to be spent on state management by the server (Fielding 2000, p. 79).

2.3. Docker Unified UIMA Interface

Docker Unified UIMA Interface is a lightweight NLP framework developed by the Text Technology Lab. It serves as a new approach for analyzing natural language for big data applications. At its core the framework is used to create and run pipelines that contain UIMA based annotators. One of the main concepts is the unification of and support for annotators developed in many different environments. This allows for the integration of existing tools while also providing a way to easily test and include newly developed tools. DUUI has already proven to be efficient and fast (Leonhardt et al. 2023, pp. 392–393) by utilizing a native Docker Swarm implementation for horizontal scaling and Lua[2] as a scripting language for communication with annotators that are not native to UIMA. The framework can be used to process plain text or large quantities of files in parallel. While the infrastructure

[2]`https://www.lua.org/`, last accessed: 14/02/2024 13:20 MET

and foundation has been implemented, key features like integrating cloud storage for reading and writing data are not yet available. Additionally, as described by Leonhardt et al. (2023, p. 390), usage is limited to Java and the command line. A web-based interface and API complemented by a sophisticated monitoring and error reporting system are the next steps to allow for lightweight usability.

2.4. Auxiliary Technologies

This project is built with the help of tools and frameworks that offered great flexibility in the development phase. Some of the most important technologies are MinIO, MongoDB, Spark Framework, and Svelte.

2.4.1. MinIO

MinIO[3] is an s3 compatible object storage system designed for environments with heavy workloads that require reliable performance and availability. It is both fast and flexible due to its horizontal scaling capabilities using so-called server pools (MinIO, Inc. 2022). Access management is achieved by creating users that authorize using access (username) and secret (password) keys. Each user receives policies that define what resources are visible and can be changed (MinIO, Inc. 2022). Performance is one of the biggest selling points for MinIO which is backed by benchmark results (MinIO, Inc. 2022) showing much faster runtime when compared to *Hadoop Distributed File System* (Borthakur 2007), a distributed file system that itself has many applications in big data due to its reliability.

2.4.2. MongoDB

MongoDB[4] is a non-relational database that stores data in collections of documents. Documents are stored in a format very similar to JSON, a map of key value pairs. Because collections do not have to follow a pre-defined schema, documents can quickly adapt to changes in the software that can occur frequently during development. A feature that significantly reduces server loads is the *Aggregation Framework* (Ken W. Alger 2022). The *Aggregation Framework* is used to build pipelines that enable developers to perform complex data transformations and queries directly in the database. Its flexibility, performance, and the JSON-like structure make MongoDB fit perfectly in many modern environments and web-based applications.

MongoDB has been chosen as the database for this project because of its adaptability in a frequently changing application. Additionally, MongoDB offers built in support for the conversion of JSON into simple data structures for usage in Java.

[3]https://min.io/, last accessed 04/02/2024 19:57 MET
[4]`https://www.mongodb.com/`, last accessed: 04/02/2024 18:47 MET

2.4.3. Spark Framework

"Spark Framework is a simple and expressive Java/Kotlin web framework DSL built for rapid development." (Spark Framework 2024). Instead of objects and classes, Spark[5] makes use of static methods allowing for very concise and readable code. Defining an API endpoint is as simple as creating a static method named after a request method, e.g. GET or POST, and providing a callback that handles the request. Spark proves to be extremely flexible and easy to use during development, allowing changes to be implemented without the need for complex refactoring. Using Spark, the entire */pipelines* path group can be defined in just 16 lines of code as seen in figure 2.1.

```
path("/pipelines", () -> {
    before("/*", (request, response) -> {
        DUUIHTTPMetrics.incrementPipelinesRequests();
        boolean isAuthorized = DUUIRequestHelper.isAuthorized(request);
        if (!isAuthorized) {
            halt(401, "Unauthorized");
        }
    });
    get("/:id", DUUIPipelineRequestHandler::findOne);
    get("", DUUIPipelineRequestHandler::findMany);
    post("", DUUIPipelineRequestHandler::insertOne);
    put("/:id", DUUIPipelineRequestHandler::updateOne);
    post("/:id/start", DUUIPipelineRequestHandler::start);
    put("/:id/stop", DUUIPipelineRequestHandler::stop);
    delete("/:id", DUUIPipelineRequestHandler::deleteOne);
});
```

Figure 2.1. Defining a path group with Spark

2.4.4. Svelte

Svelte[6] is a JavaScript framework designed to build web-based user interfaces. It was first released in 2016 and has steadily gained popularity because of its unique design. Svelte is distinct from many other frontend frameworks in that it shifts much of the work from the browser to the build step. Code is written in a Svelte specific language that allows for reactivity and management of state with minimal effort. In the build step, this code is compiled into highly optimized JavaScript. Development with Svelte has proven to be both fast and reliable. While the framework does not have a community as large as React[7], there are still many well designed component libraries available that help with development speed and consistency. The 2023 Developer Survey (StackOverflow 2023) revealed, that although not

[5]`https://sparkjava.com/`, last accessed: 04/02/2024 18:47 MET
[6]`https://svelte.dev`, last accessed: 04/02/2024 18:47 MET
[7]`https://react.dev/`, last accessed: 05/02/2024 00:16 MET

used as much as long standing frameworks like React, Svelte is well-liked by its community. Around 75% of developers that used Svelte would consider using it again.

3 Related Work

The work done in this thesis is split into the backend and frontend parts that interact with each other to improve the usability of DUUI. As a starting point, different workflow automation tools have been explored in regards to what information about workflows is presented and how it is visualized.

3.1. Workflow Automation

Web based user interfaces that offer visual feedback and a way for users to interact with complex software is common in most areas of technology. This holds for software designed to automate repetitive tasks with workflows for data management, logging, and processing. Automation is important for many reasons including the increase in productivity and homogenization of workloads under one software which makes maintenance much simpler. There are many tools available that allow for the creation or usage of automated workflows. A simple annotation tool is Sparv (Borin et al. 2016) that can be used from the the command line or in a web interface to annotate a text document with support for sentence, token, lemma, and part of speech tagging. Tools like *Flyte* and *argo* are designed for large scale data processing and machine learning tasks powered by Kubernetes with Natural Language Processing being a subset of possible use cases. Workflow automation is applicable in other areas as demonstrated by *Zapier*, a tool used by over two million businesses (Zapier 2024), or *Power Automate* that is part of the *Microsoft 365* ecosystem. Both focus on automating business logic and managing communication between different third party applications with countless integrations making them very flexible. While use cases differ, the core concept for automation software is the same: Automate repetitive tasks efficiently. Accessibility and usability are vital for both user experience and efficient maintenance by developers.

Providing an intuitive and simple way to interact with software is essential for usability. Steep learning curves and high barriers of entry discourage the use of software early on for many potential users. To prevent these issues from arising, graphical user interfaces for visual feedback paired with intuitive drag and drop editors are a common practice for workflow building tools. argo, Zapier and Power Automate have web-based editors that make the creation of potentially complex workflows efficient and quick. Workflows can be created from scratch or from a wide range of templates that serve as a starting point and help new users to explore existing features. In the editor a workflow is visually represented by a *Directed Acyclic Graph* (DAG). In contrast, Flyte workflows are created in code, meaning expertise in at least one of the supported programming languages is mandatory. The framework aims to reduce overhead and increase production speed in a scalable and hosted environment that runs reproducible workflows (Gale 2020). While Flyte is a comprehensive, well-documented framework in active development, getting started is difficult due to the entry barrier being higher than competing technologies. Installation and creation of workflows having the requirement of at least some knowledge of the command line and programming may prevent users from utilizing its full potential. Most mentioned tools include the option to monitor the progress and status of workflows visually in a web-based interface. In case of errors, presenting meaningful information that helps with the correction of such errors is invaluable especially for less experienced users unfamiliar with the underlying software. Most existing tools use very individual solutions to offer their users insights in the inner workings of the software. While each tool has their own design and displays different information, the state of individual tasks and their duration are provided by all mentioned tools. Power Automate and Flyte have been explored in greater detail with a focus on GUI design, inspiring parts of the web interface for DUUI.

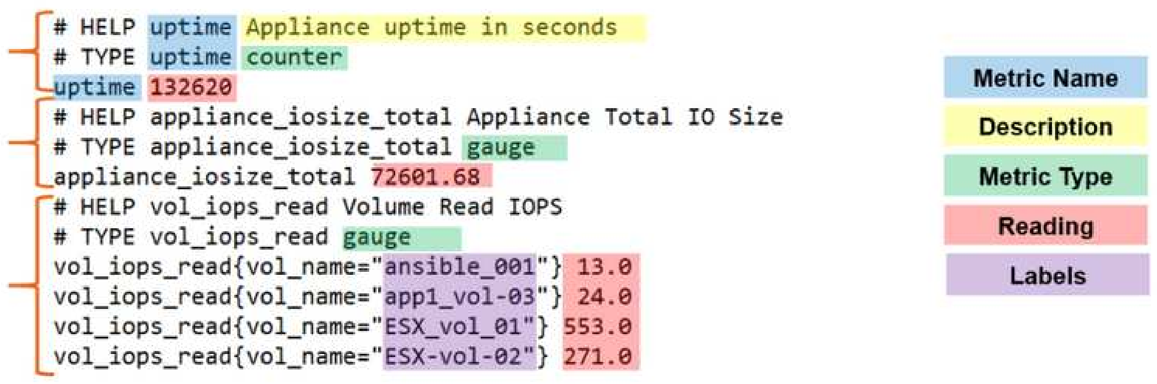
3.2. Monitoring

Monitoring describes the process of tracking and interpreting application metrics by transforming the information into a measurement of system integrity and user experience (Turnbull 2018, pp. 6–7). It relies on the applications to expose relevant metrics that can then be scraped for further processing and possibly visualization. A popular, highly customizable monitoring system is *Prometheus* which is used in this project due to its simple setup, compatibility, and easy integration with many visualization tools. Prometheus works by scraping metrics from HTTP endpoints exposed by applications (Turnbull 2018, p. 48) and storing them in time series for further analysis. There are four types of metrics used by Prometheus (Prometheus 2024):

- *Counter*, a metric for numerical values that can only be increased.
- *Gauge*, a metric for numerical values that can be increased and decreased.
- *Histogram*, a metric that groups observations into customizable ranges and tracks the total count in each range.
- *Summary*, similar to a histogram but instead of in ranges, values are grouped in quantiles instead.

These metrics are exposed in a text-based format as seen in figure 3.1 and transformed into meaningful data that can be visualized in a time series.

The visualization can be done by the local Prometheus dashboard or sent to a specialized monitoring and visualization software. Two very flexible and well established metric visualization tools are Grafana and Datadog, both of which have integrations for Prometheus. Grafana is a tool for visualizing a diverse set of application and system related metrics in intuitive dashboards (Grafana 2024). These dashboards allow developers to gain real time insights in the integrity of servers, memory usage, network latency, or response times. The representation of metrics in charts in a centralized dashboard helps with the fast identification of critical failures and high system loads which increases the reliability and availability of applications. Due to its intuitive interface, dashboards can be designed for one's individual needs, making Grafana flexible. The platform's versatility lies in its ability to connect



```
# HELP uptime Appliance uptime in seconds
# TYPE uptime counter
uptime 132620
# HELP appliance_iosize_total Appliance Total IO Size
# TYPE appliance_iosize_total gauge
appliance_iosize_total 72601.68
# HELP vol_iops_read Volume Read IOPS
# TYPE vol_iops_read gauge
vol_iops_read{vol_name="ansible_001"} 13.0
vol_iops_read{vol_name="app1_vol-03"} 24.0
vol_iops_read{vol_name="ESX_vol_01"} 553.0
vol_iops_read{vol_name="ESX-vol-02"} 271.0
```

Figure 3.1. Prometheus metrics format, `https://vexpose.blog/2022/02/18/prometheus-exporter-for-powerstore/`, last accessed: 16/02/2024 15:46 MET (Screenshot from 16/02/2024 15:46 MET)

with various data sources (Grafana 2024), including databases, cloud services, and monitoring tools like Prometheus.

Datadog is similar to Grafana as it also provides a platform to create dashboards that visualize metrics for more informative and robust monitoring of applications. Compared to Grafana, Datadog is a more advanced tool with additional features that make it an all in one monitoring solution. The platform's alerting system is a standout feature, allowing users to set up notifications based on specific thresholds and conditions (Datadog 2024). With machine learning algorithms, Datadog's alerting becomes more intelligent over time. This ensures that developers are quickly notified of potential issues, reducing downtime and enhancing the overall reliability of the system. While this can be useful in many cases, the complexity also increases significantly and many features may remain unused in projects that do not require them.

Although both technologies offer very similar capabilities for visualizing application and system metrics, Grafana has been chosen for this project because creating intuitive dashboards is simple and quick while most features that make Datadog stand out are simply not required for DUUI at this point.

4 Implementation

Building an interface for DUUI requires multiple steps and prerequisites. Features and data structures extending the framework to create the possibility for big data analysis that is not limited to local files have to be added. Furthermore, having control over and presenting detailed insights about the state of processes is only possible if methods and data to achieve these goals are available. The following sections provide an overview into the changes made to the framework and how these changes were used in the design of both the backend and frontend parts of the web API. The code written for this thesis can be found on GitHub[1].

4.1. Updates to DUUI

Docker Unified UIMA Interface is the basis for the implementation of both the API and the web-based interface. The framework must provide a way to extract the integrity of pipelines and progress of documents. As described in section 2.3, one key feature is the integration of cloud storage providers as input sources and output locations for documents. Instead of relying on local files and plain text as data sources, the option to read from and write to cloud providers greatly enhances productivity and enables DUUI to be used for real world big data analysis.

4.1.1. Input and Output

The addition of the *IDUUIDocumentHandler* interface to DUUI provides easy integration with a user's cloud storage of choice. An implementation of the interface for a specific provider includes five basic methods that are used to interact with the API of the service. These five methods offer a way to read, write, and list files at a specific location with the option to do so recursively. The read and list methods return *DUUIDocument*s that are the container for the files to be processed while also storing metrics during processing. There is also an implementation called *DUUILocalDocumentHandler* for reading from and writing to disk. The simple interface design allows for the implementation of practically any third party cloud storage as a provider as long as an API to interact with the data exists. As a starting point, two cloud storage services namely, Dropbox and MinIO, have been implemented for DUUI. Using these implementations requires the user to provide credentials for authorization. Example code for the usage of MinIO can be found in the appendix at A.1. When used in a process with DUUI, a handler is always used by a *DUUIDocumentReader*, a class that is responsible for the pre-processing of documents and managing both read and write operations for the installed handlers. When the composer's run method for using a *DUUIDocumentReader* is called, file metadata is retrieved by calling the *listDocuments* method. This initial

[1]`https://github.com/texttechnologylab/DUUIController`, last accessed 12/03/2024 12:55 MET

listing of documents in the source location is followed by multiple filters that may reduce the number of files to be processed, depending on the settings that were passed to the reader. The remaining documents are then read and processed by one or multiple threads or workers. Figure 4.1 visualizes these steps in a flow chart. The actual file content is stored as raw bytes in a *DUUIDocument* object during processing and cleared when the document has been uploaded to the output location or is otherwise finished. Clearing the bytes after processing the document is done to reduce peaks in memory usage that would occur if all files were stored in memory at once. The final part of writing files is planned to be extracted into a separate class (*DUUIDocumentWriter*) in the future but has been implemented here for simplicity.

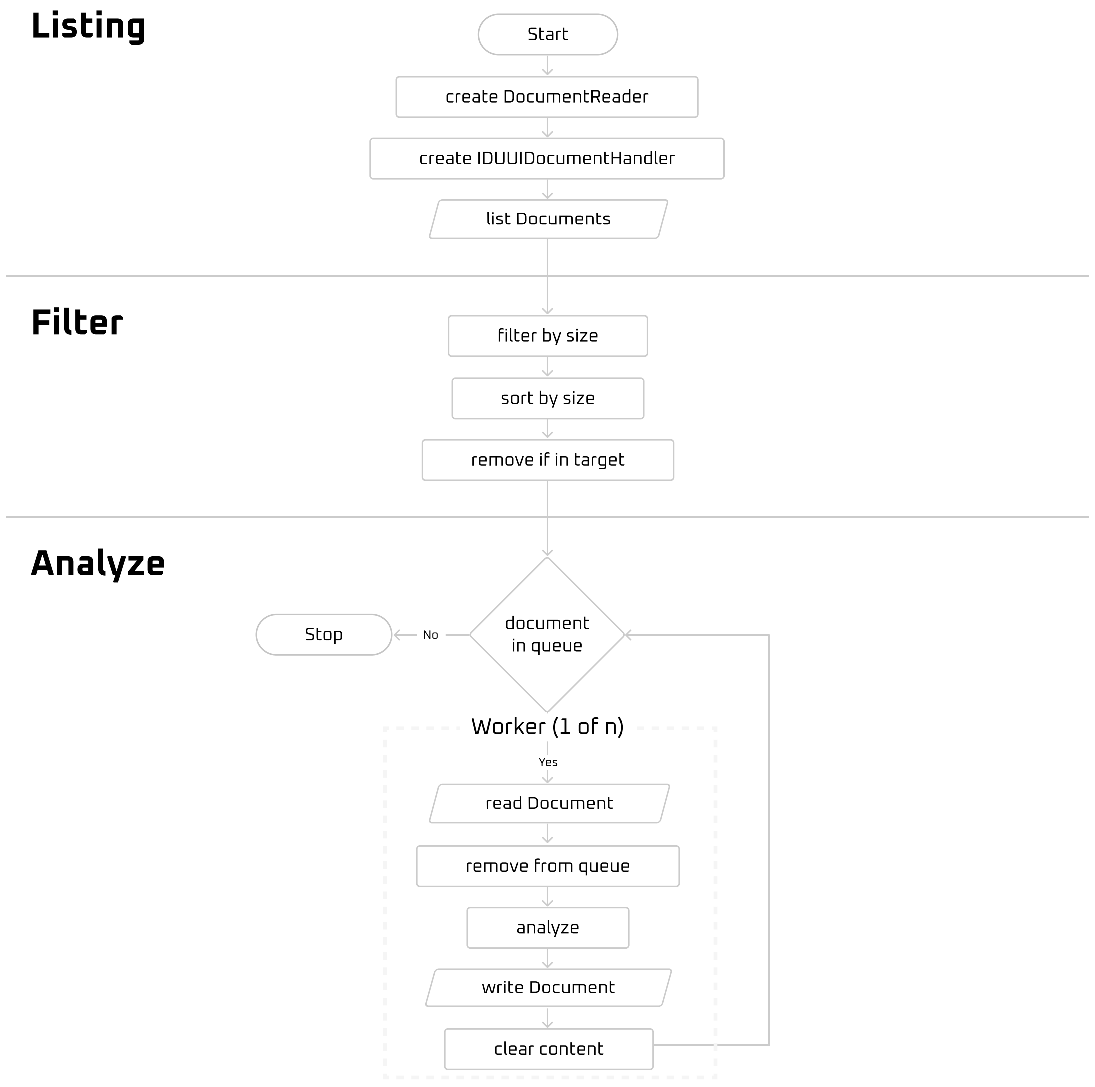


Figure 4.1. Flow of data through a process using the *DUUIDocumentReader* class and *IDUUIDocumentHandler* interface.

4.1.2. Monitoring and Error Reporting

Docker Unified UIMA Interface used simple logging to the console as a way to provide a way of monitoring the state on progress of a pipeline. To allow for more sophisticated logging of important stages and tracking the exact timestamp at which these events occur, an event system has been added. Events can be added at any desired stage of a pipeline by calling the composer's *addEvent* method. The method then creates a *DUUIEvent* object that holds a timestamp, sender, and message. In the *addEvent* method, the event can be processed further, stored in a database, or simply logged to the console by setting the composer's *DebugLevel* which acts as a filter. In the process of adding events in various places, the possibility to cancel a process has been added as well by providing the composer's shutdown flag in multiple processing steps. Cancelling a process is then done by simply calling the *shutdown* method on the composer which signals drivers and components to stop as soon as possible. Through the addition of a *DUUIDocument* class, a per file monitoring is achieved. A *DUUIDocument* object not only contains file metadata and content but also tracks many metrics including progress, status, file size, durations for the different steps in the pipeline, and errors.

4.1.3. Reusability

The final addition to the framework is an alternate approach to the instantiation and usage of pipelines. With the exception of native UIMA based annotators, instantiating components takes up to ten seconds per component because communication via RESTful has to be established before it can be used. Additionally each instance of a component running as a Docker container has to be started individually by the Docker Daemon which further increases the duration of instantiation. Although not as much of an issue for simple pipelines, instantiating a complex pipeline every time a process is started is both unnecessary and delays the actual process. There have been multiple attempts to circumvent this issue that each had their downsides. The first attempt was to prevent the composer from going through its default shutdown procedure by setting a flag that would skip the shutdown of the pipeline. Although this approach worked, it revealed a different issue caused by the fact that a pipeline is tied to the composer that instantiated it. Because of this coupling, stopping one process would also cancel all other active processes using this pipeline. One way to resolve this problem is to reduce coupling between pipelines and composers and share instantiated pipelines among different composer instances. In the final implementation, methods have been added to the composer class that instantiate and return a pipeline for future processing. These instantiated pipelines can then be stored and passed to another composer by calling the *withInstantiatedPipeline* method. Using this approach, processing can start immediately without any time spent on creating the type system or starting docker containers. Pipelines can be instantiated sending a POST request to the */pipelines/start* endpoint or through the web interface.

Running a pipeline in the background could be extended by a scheduled job that periodically checks for files in a specific input location. When unprocessed files are detected, the pipeline would then automatically process these files without the need to manually start the pipeline each time.

4.2. Web API

The backend is responsible for handling the communication between clients, the database, and DUUI. It has been implemented in Java using the Spark Framework by defining a set of routes that are organized in path groups. Routes can be split into three categories that are responsible for different tasks. The first type of routes is used to interact with the database by reading and writing resources that are then displayed to the user. The second type communicates with DUUI directly, providing functionalities to start and cancel processes. Additionally, instantiating and shutting down pipelines, as described in section 4.1.3, is possible using routes in the pipelines path group. These routes operate through two classes, a controller and a request handler. The request handler validates the incoming request and transforms its data into a usable format before passing it to the appropriate controller. The controller then uses this transformed data to perform the requested action. The structure of DUUI is reflected in the web API through five controllers that represent different building blocks of analysis. Lastly there are two routes that can be used for uploading and downloading files using the *IDUUIDocumentHandler* interface and one that exposes metrics that are scraped by Prometheus for monitoring purposes. The web API integrates Prometheus by exporting a variety of different metrics that can be scraped from the */metrics* endpoint and visualized with a tool like Grafana or Datadog introduced in section 3.2. Prometheus metrics should follow a specific text-based format that is automatically applied to metrics when using the Prometheus Java-Driver. Figure 4.2 shows an example of the creation and export of a metric as a static variable using methods provided by Prometheus. Expanding the set of exported metrics is simply done be registering new metrics as variables and providing methods to update them.

4.2.1. Database Interaction

MongoDB has in many ways improved the readability of code used in the backend by providing a simple API for database operations and handling of data in the JSON format. The MongoDB Java driver works with the *Document* class that in its core is a wrapper around a *Map* object in Java. Documents can be created from a JSON string using the *parse* method and also transformed into a JSON string with the *toJSON* method. This capability has been very useful for most routes by reducing the amount of code and therefore improving readability. With the transformed data available, making changes to the database is as simple as calling a method like *insertOne*, *updateOne*, or *deleteOne* for the appropriate collection. Write operations are, in most cases, straightforward and can be executed by providing the required data. Read operations on the other hand can be more challenging because not all available entries in the database are of interest or should be accessible to a user. Additionally, retrieving a large dataset all at once from the database causes performance issues because more data has to be sent over the network. This is especially problematic in graphical user interfaces that have to deal with longer loading times. The solution to this issue is the reduction of returned entries by applying filters to a dataset. MongoDB has a feature called the *Aggregation Framework* (Ken W. Alger 2022), a powerful tool for complex data transformations used in the implementations for most controllers in this project and especially important in

```java
    private static final Gauge requestsActive = Gauge.build()
        .name("duui_requests_active")
        .help("The number of active requests")
        .register();

    public static void incrementActiveRequests() {
        requestsActive.inc();
    }

    public static void decrementActiveRequests() {
        requestsActive.dec();
    }

    public static String export() throws IOException {
        StringWriter writer = new StringWriter();
        CollectorRegistry registry = CollectorRegistry.defaultRegistry;

        TextFormat.write004(writer, registry.metricFamilySamples());
        return writer.toString();
    }
```

Figure 4.2. Creation and export of a Prometheus metric in Java.

the implementation of the pagination system for tables in the web interface. The aggregation framework is used to build so-called aggregation pipelines that consist of one or more stages, each of which applies a specific operation to entries in the database. These operations include filters, projections for adding and removing fields from entries, grouping, sorts, and many more that are case specific. Database read operations are complemented by the *MongoDB-Filters* convenience class that groups common filters applied to collections including limit, skip, sort, order, search, and comparison filters into a single object for easy access. Filters are extracted from the request and added to a *MongoDBFilters* object that is then used in database operations. Figure 4.3 shows the different steps and interactions in data base operations. The web interface makes use of these filters with the *PaginationSettings* type that allows for the retrieval of a specific range of entries from the database. Only three values are required for a functioning pagination system: the page number, the page size (limit), and the total number of matching entries.

4.2.2. Processing

With the database interaction implemented, the next step is being able to use pipelines with DUUI. To do so, the *DUUISimpleProcessHandler* class has been created. An instance of a process handler encapsulates the execution of a pipeline by administrating all steps from acquiring data to running the pipeline, possibly writing the result to the output location, and cleaning up resources once the process has finished successfully or failed due to an error. A

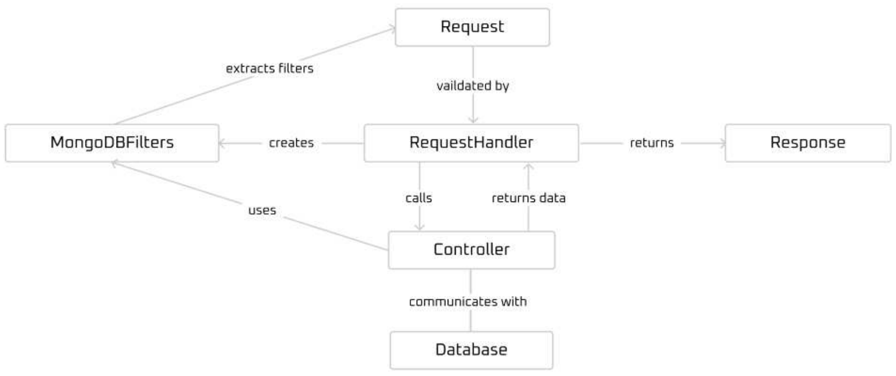


Figure 4.3. Interactions between different classes and objects during a request.

flow chart representing a process can be found in the appendix in figure A.1.

A process is initiated by sending a POST request to the */processes* endpoint. The request body must include the id of the pipeline to be executed and an object containing the information for the data source (input). Optionally, an object for the output location and additional settings for the process can be sent in the request as well. Before creating an instance of a *DUUISimpleProcessHandler*, the controller checks if a pipeline with the specified id is available and instantiated already. If so, the instantiated pipeline is passed in the handler's constructor for reduced setup time. Otherwise, a new pipeline is instantiated and used for processing instead. Once an instantiated pipeline is available, the process handler starts the execution by reading data from the input location, applying filters from the settings object to possibly reduce the number of documents that need to be processed. The remaining documents are then processed by one or more workers in parallel until the queue is empty. If an output handler and location have been declared during setup, each document is written to the output location as soon as it has finished processing. The output step is performed per document and not after all documents have finished to reduce memory loads. Memory usage can be high because the document stores file contents in bytes during processing. The bytes are cleared right after uploading as storing them is no longer required. In each stage the progress and status of documents is updated in the database using a separate, scheduled thread to reflect the current state in the web interface. The final stage in a process is called exit and cleans up resources that are no longer used. The exit step can be reached in three states:

Completed The process completed without fatal errors. This does not guarantee that all documents have been processed successfully. If the *ignore_errors* flag has been set, documents encountering errors are skipped without throwing an exception.

Failed The process has failed during execution due to a fatal error. Fatal errors can occur at any stage of a process and immediately interrupt the pipeline.

Cancelled The process has been cancelled by sending a PUT request directly to */processes/:id* or by submitting a cancel request from the web interface. A cancellation is not an immediate interruption but prevents further documents from being processed by signalling the composer to shutdown as soon as possible.

Regardless of the state that lead to the exit stage, one last update is performed , setting the final result of both the process and each document. If the pipeline has been reused, it is not shut down but remains active for future requests.

4.2.3. Authorization

To utilize cloud solutions in a process, users need to authorize DUUI to make requests on their behalf. For Dropbox, this involves undergoing an OAuth 2.0 authorization process on the Dropbox website, granting DUUI access to create a dedicated folder for performing IO operations. Upon user acceptance, Dropbox returns a one-time code used to generate an access token. Because access tokens have a limited lifespan, a more practical approach involves generating a refresh token alongside the access token using the code returned after the OAuth 2.0 authorization. The refresh token allows DUUI to generate a new access token as needed without requiring the user to repeat the authorization flow. This method ensures continued access while still allowing the user to revoke access at any time. For MinIO, users must provide DUUI with an access key and a secret key for authorized requests in addition to the endpoint at which the service can be reached. The user creates an account for DUUI in their s3 solution, allowing DUUI to store both a username (access key) and a password (secret key) for a specified endpoint. These credentials, which define the scopes and buckets DUUI can access, can be managed by the user. Unlike Dropbox, there is no need for repeated authorization flows, as the username and password alone are sufficient for making authorized requests on behalf of the user.

As anyone can send requests to a URL, the security of data depends on the verification of the clients identity (authentication) and if the client is allowed to access the resources (authorization). To prevent unauthorized requests and unwanted changes in data, clients are asked to send an authorization header with each request. The website uses a session id as its method of authorization which means that the user must register and be logged in to use the application. The session id is automatically sent to the server as proof of identity. Unauthorized requests are rejected by the server with the status code 401 - Unauthorized, meaning only the owner can modify their data. Because the session id is created in the web interface after logging in and is therefore tied to the browser, it cannot be used for authorization in any other environment. This limits the support for the usage of DUUI from other programming languages.

4.2.4. Usage with Python

To allow communication with the API from outside the web application, other authentication methods are required. A common solution is the usage of API keys, a unique proof of identity, as an alternative to the session id. API keys allow the communication with DUUI using any programming language the user may be familiar with which broadens the target audience significantly. As an example, the usage of Python[2] with DUUI has been explored. Python is a well-liked and widely used interpreted programming language as can be seen in the Developer Survey done by StackOverflow (2023). It has an easy to learn syntax and because of that offers rapid development speed. It is also considered one of the most important languages in machine learning, which includes the field of natural language processing, as many frameworks and libraries such as Scikit-learn, TensorFlow, PyTorch, spaCy and Keras are available for rapid development of machine learning applications. In addition, libraries such as Pandas and NumPy offer the possibility of efficiently transforming large amounts of data and performing mathematical operations. Given that Python is so widely used and beginner friendly, the target audience can be increased by providing simple bindings for the usage of DUUI. These bindings should abstract the inner workings of the API by hiding raw requests behind intuitive and well documented functions. The usage of Python for creating and using a pipeline with the API can be seen in A.1.

4.3. Graphical User Interface

The design of the web interface went through many iterations while always reflecting the frequently changing capabilities of the web API. Its primary goal is to increase the usability of DUUI and offer the possibility of using the framework without having to code or even know how to write code in the first place. This aligns with the purpose of a GUI to significantly reduce the barrier of entry and open the possibility for inexperienced users to effectively use a software (Cardinali 1994, p. 7). To do so, a multi-page website [3] with many capabilities for the analysis of large quantities of data has been created.

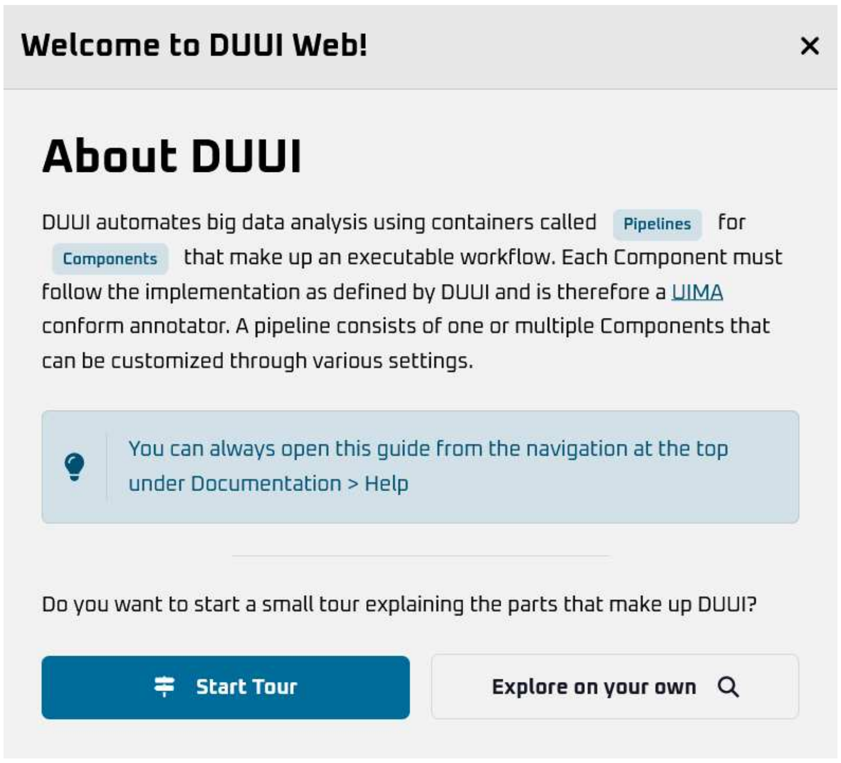
4.3.1. Account

The account page is the first page a user visits after the registration has been completed successfully. Profile information as well as preferences for the website's behavior and appearance are available here. Additionally, the generation, deletion, and retrieval of the user's API key and connections to external cloud providers are managed on this page. Users with the *admin* role can manage permissions of other accounts in an additional view that is hidden for regular users.

[2]`https://www.python.org/`, last accessed 16/02/2024 16:37 MET
[3]The web interface can be found at `https://duui.texttechnologylab.org/`.

4.3.2. Documentation and Help

The web interface includes a documentation page for the web API, its endpoints, and the usage of the website itself. The website documentation introduces capabilities and features in a linear structure by guiding a user through the process of creating pipelines and components interactively. In addition, a modal dialog as seen in figure 4.4 can be opened from the navigation at the top of any page that explains how to create and use pipelines. The API reference includes detailed documentation for every endpoint that can be reached by users including possible parameters and code examples as seen in figure A.17.


Welcome to DUUI Web!

About DUUI

DUUI automates big data analysis using containers called Pipelines for Components that make up an executable workflow. Each Component must follow the implementation as defined by DUUI and is therefore a UIMA conform annotator. A pipeline consists of one or multiple Components that can be customized through various settings.

You can always open this guide from the navigation at the top under Documentation > Help

Do you want to start a small tour explaining the parts that make up DUUI?

Figure 4.4. A modal that is shown after registration or when opened from the navigation under Documentation > Help.

4.3.3. Pipelines

Pipelines are created in the pipeline builder at /pipelines/build in a three step form. First the user is asked if they want to create a pipeline from scratch, meaning all settings are set to default and no components are predefined. If the user decides to start from one of the templates provided by DUUI, all settings are copied from the template to a new pipeline that can then be further customized. Regardless of the starting point, the user proceeds to the second step, at which pipeline related information, including name, description, tags, and pipeline settings, can be set. In the final step, components can be edited and added to the pipeline. A component can also be created from scratch or from a template. There are a few settings that are mandatory for components. These include the driver and the target

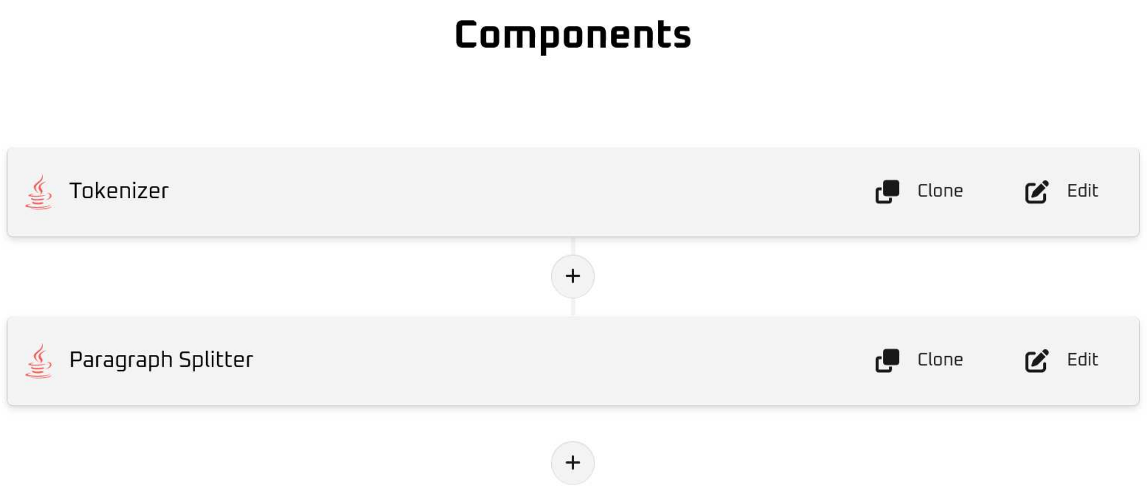
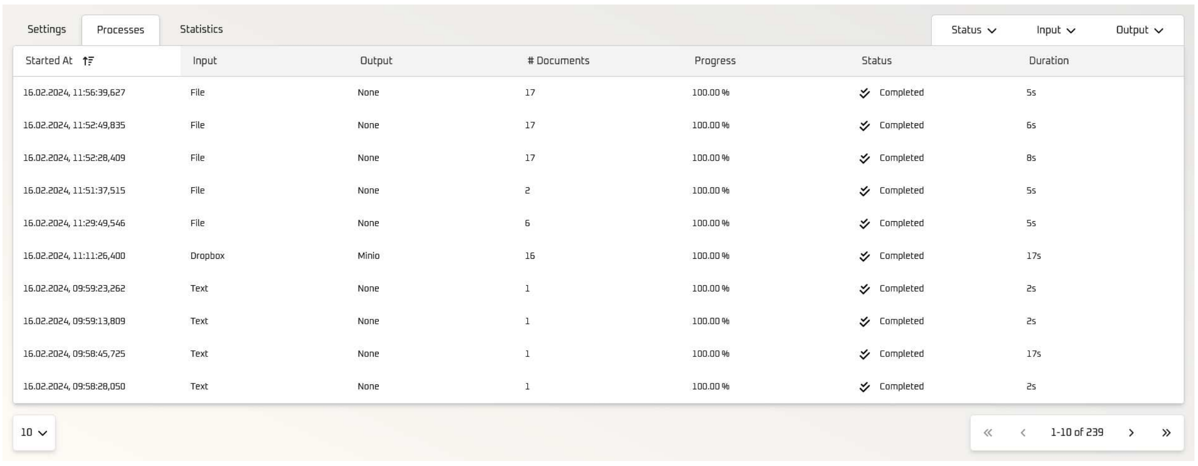


Figure 4.5. Representation of a pipeline with two components in the web interface. Components can be added at the end or between existing components.

which can be a docker image name, a remote URL to the running component, or a class path depending on the selected driver. Every other setting is optional but filling in meaningful values in the description or tags input fields is recommend to help with documentation and reusability. Depending on the selected driver, a set of options can be set that influence the component's behavior. All types of components offer the scale option that replicates the *Analysis Engine* to increase processing speed. Other important options are *use_GPU* and *docker_image_fetching*. The latter allows DUUI to download the image from the docker hub if it is not available locally.

All user created pipelines are found at */pipelines* where a dashboard showing a grid of cards with broad information about a pipeline can be found. This includes name, description, tags, how often the pipeline has been used, and how many components are part of the pipeline. By clicking one of the cards the website navigates to a pipeline specific page that provides a more detailed look in the specific pipeline. The page for an individual pipeline is reached at */pipelines/id* and split into the three tabs: *settings*, *processes*, and *statistics*. The *settings* tab allows the user to manage options and properties for the pipeline and its component. The pipeline name, description, tags, and settings can be updated here. Additionally, this tab lists all pipeline components in a DAG that can be rearranged to change the order of execution. Components can also be added to the pipeline either at the end or at a specific position both from scratch or using one of the available templates. Each component can be edited in a *Drawer* (figure A.13) on the right side of the screen. This drawer shows all settings related to this specific component and offers actions to update and delete the component. When focusing certain input elements, a tooltip is displayed that clarifies the expected input. The *processes* tab lists all started processes in a table that uses a pagination system to reduce the amount of data that has to be retrieved from the database at once. By default, the limit is set to 10 processes per page. MongoDB aggregation pipelines as described in section 4.2.1 are used



Started At	Input	Output	# Documents	Progress	Status	Duration
16.02.2024, 11:56:39,627	File	None	17	100.00 %	✓ Completed	5s
16.02.2024, 11:52:49,835	File	None	17	100.00 %	✓ Completed	6s
16.02.2024, 11:52:28,409	File	None	17	100.00 %	✓ Completed	8s
16.02.2024, 11:51:37,515	File	None	2	100.00 %	✓ Completed	5s
16.02.2024, 11:29:49,546	File	None	6	100.00 %	✓ Completed	5s
16.02.2024, 11:11:26,400	Dropbox	Minio	16	100.00 %	✓ Completed	17s
16.02.2024, 09:59:23,262	Text	None	1	100.00 %	✓ Completed	2s
16.02.2024, 09:59:13,809	Text	None	1	100.00 %	✓ Completed	2s
16.02.2024, 09:58:45,725	Text	None	1	100.00 %	✓ Completed	17s
16.02.2024, 09:58:28,050	Text	None	1	100.00 %	✓ Completed	2s

Figure 4.6. Processes that used a specific pipeline are listed in a table that can be sorted by any column and filtered by multiple criteria. The table can be traversed using the pagination system below.

by the backend to filter and sort data that is requested for the current page in the table while at the same time keeping track of the total amount of matching documents. This is required for the pagination system to know how many pages it can display. Charts that display details about the usage of pipelines can be found in the *statistics* tab. There are currently four charts available that display the status of executions, errors that have been encountered, usage of different types of input and output, and the usage of the pipeline per month. Regardless of the current tab, the page includes a group of actions that perform different actions like copying and exporting a pipeline. Additionally, pipelines can be instantiated as a service running in the background as described in section 4.1.3.

4.3.4. Processes

When starting a new process from a pipeline, the user is first asked about the input source of documents. There are currently four input options available: Text, files, Dropbox, and MinIO. Each option comes with settings that can be applied, changing the way the input is pre-processed. The simplest way to use a pipeline is to process plain text that is entered into a text area. Internally a new CAS object with the supplied text as the subject of analysis is created and processed. Choosing files as the input source allows to process local files on the client's machine. Optionally, input files can be uploaded to one of the available cloud storage systems to make them available across multiple machines. Dropbox and MinIO directly obtain files from the respective storage at the specified path. Input files are wrapped by the *DUUIDocument* class during processing to store metrics and update their progress in the database. With the exception for the processing of text, documents can be filtered and sorted as a pre-processing step to reduce the number of documents that need to be processed in the pipeline. These settings include the exclusion of files smaller that a certain size in bytes or documents that are already present in the output location. Additionally, documents can be sorted in ascending order by their size so that smaller documents that are faster to process are prioritized. The result of a process can either be ignored by setting the output to none, or written to a cloud storage by providing the full path to the target location, excluding the Apps/Docker Unified UIMA Interface component that is created when connecting with Dropbox. A process is then started by clicking the submit button.

Once submitted, the process is started by the web API and the user is redirected to a page that is used to monitor the progress and status of documents and the process itself. The page can also be reached by clicking on the appropriate row in the table that displays the processes using the pipeline. All settings that were used in the creation of the process can also be viewed on this page and copied in the JSON format for usage outside the web interface. Another functionality on this page is the option to restart a process with the exact same settings by redirecting to the creation page and copying all settings used. This speeds up the creation of identical or similar processes for which only a single setting may change. In case of small errors during the creation of a process, it can be cancelled and restarted quickly without the need to manually repeat all required steps.

Processes are used as a container to control the analysis of documents with a specific pipeline. Because documents contain the data to be analyzed, they are monitored especially thoroughly. Documents are listed in a table that can be sorted, filtered, searched, and traversed similar to the one listing processes for pipelines. It includes columns for the document name, its progress, status, size in bytes, and process duration. Clicking on a row in the table opens a drawer that displays additional information about a single document. If an output location has been specified and the document has been processed successfully, it can be downloaded directly from the dialog in the web interface. There are currently two sections with metrics and results available each of which includes a chart. The annotates that have been added by the pipeline's components are displayed in text format as well as in form of a reactive and filterable treemap. The duration of each step in the process is measured, stored, and displayed in a timeline chart. Examples can be seen in figure 4.7 and A.16.

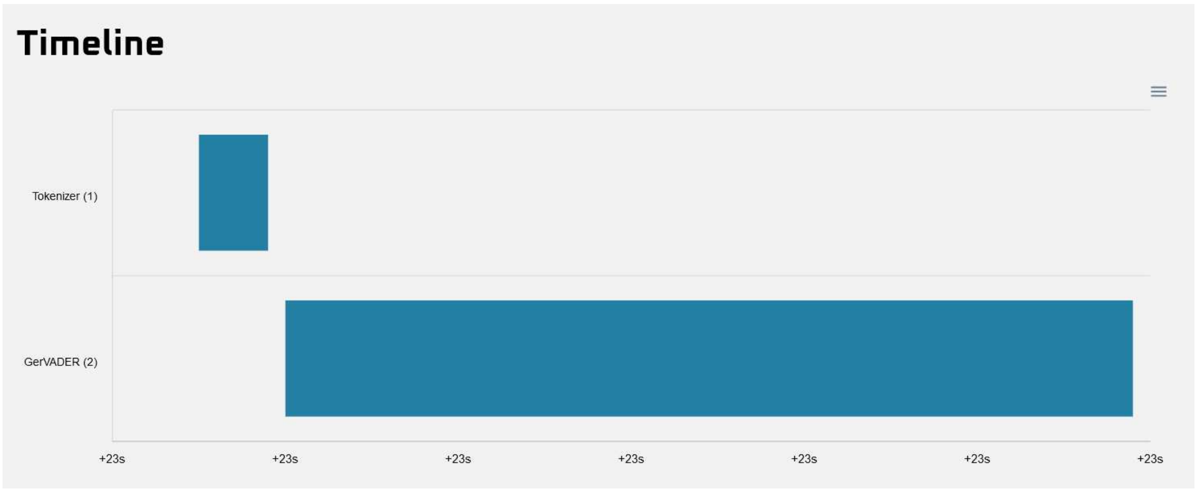
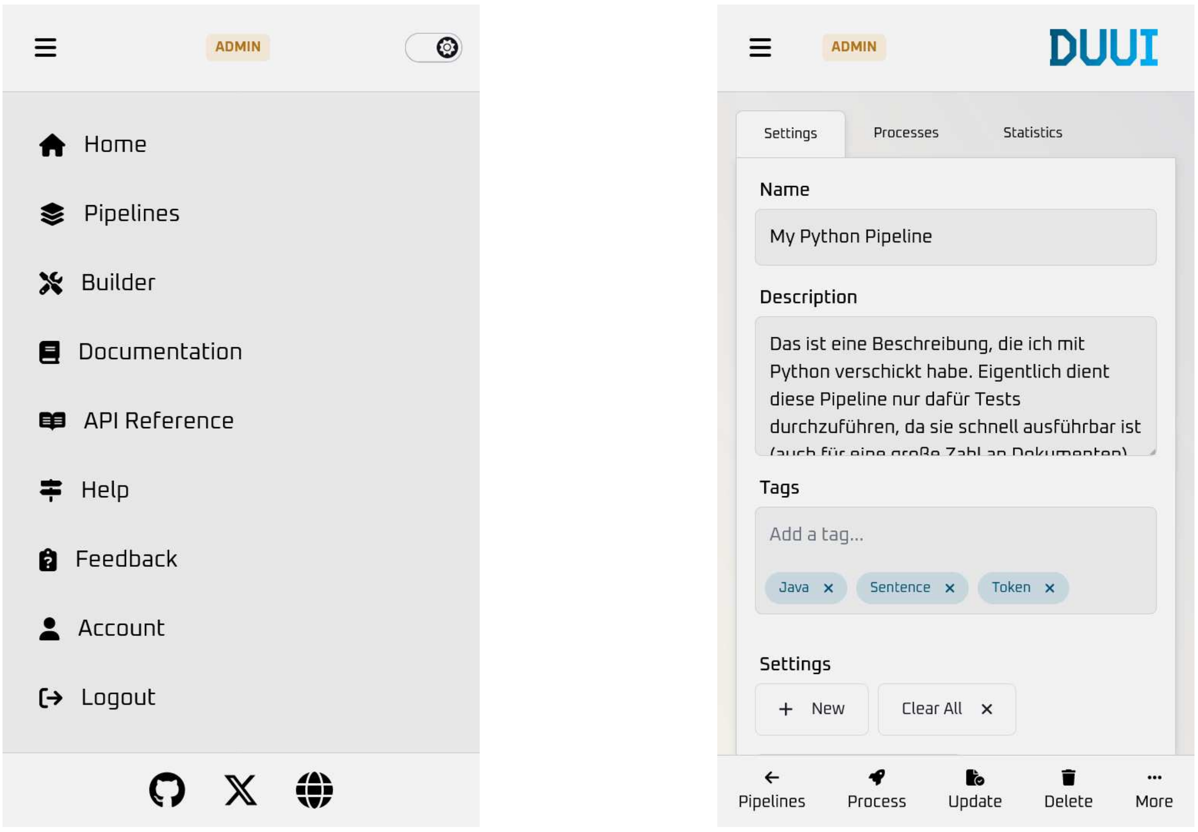


Figure 4.7. Timeline of individual steps in a process (processing by each component only took milliseconds to complete in this case). This documents has been processed by the components *BreakIteratorSegmenter* (*Tokenizer*) and *GerVADER* for sentiment analysis.

4.3.5. Mobile Support

Although the web interface is designed to be used on desktop, making it responsive and therefore usable on smaller screen sizes adds to usability. All features are available and can be used on mobile devices without the need to install a platform specific app. On mobile devices, users typically interact with the screen using their thumbs or a pen. Placing important buttons towards the bottom of the screen ensures that users can easily reach and tap them without having to stretch or adjust their grip. Reacting to changes in screen size and orientation by rearranging or resizing elements is part of *Responsive Web Design* as described by E. Marcotte (2010). While working on mobile might be less desirable overall because there is less space to show important details and charts, the ability to check the progress of a process or quickly resubmit on the phone can be convenient nonetheless.



Name

My Python Pipeline

Description

Das ist eine Beschreibung, die ich mit Python verschickt habe. Eigentlich dient diese Pipeline nur dafür Tests durchzuführen, da sie schnell ausführbar ist (auch für eine große Zahl an Dokumenten)

Tags

Add a tag...

Java × Sentence × Token ×

Settings

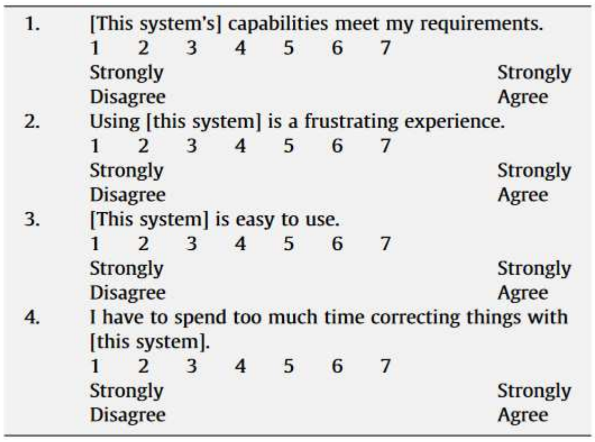
Figure 4.8. Navigation and controls on mobile devices adjusted to be more usable on small screens.

5 Evaluation

The evaluation discusses the web interface's benefits and shortcomings regarding usability and if lowering the barrier of entry for inexperienced users is achieved. In order to rate the effectiveness of the web interface its usability is rated by users with different background experience with DUUI, NLP, and programming.

5.1. Approach

Determining whether the web interface is effective and usable as an extension to DUUI has been set as the goal for the evaluation in this thesis. The ISO 9241-11 standard defines usability as follows: "Extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use." (Lopez et al. 1998, p. 2). Measuring the usability of an application can offer detailed insights about what features and design decisions are well received as well as highlight flaws that should be reviewed. An important part of this process is testing by different users (Lopez et al. 1998, p. 5) by asking to complete a specific task that represents a representative or real use case. A common way to ask for feedback that can easily be evaluated is the use of a *Likert scale.* A Likert scale is an evaluation tool that uses a numbered scale to determine agreement with a question or statement. The evaluation done for this thesis uses Likert Scales as part of the *Usability Metric for User Experience.* "The Usability Metric for User Experience (UMUX) is a four-item Likert scale used for the subjective assessment of an application's perceived usability." (Finstad 2010, p. 323). All four items are simple statements about the user experience in different categories on a scale of 1 (*Strongly Disagree*) up to 7 (*Strongly Agree*). Figure 5.1 shows the four statements proposed by the UMUX which are part of the feedback form in a slightly modified way. Statements can be worded positively or negatively, which influences the evaluation of the rating. To evaluate the available data, it is transformed by scoring positively worded statements as *score - 1*, while negatively worded statements are scored as *7 - score* (Finstad 2010, p. 326). This is done because a low value for negatively worded statements is considered a desirable score, e.g. when rating the experienced frustration with the system. The transformed scores are then summed, normalized, and finally scaled between 0-100. The mean score across all submissions is the result, representing the perceived usability of the system. To allow for accurate and useful feedback for the web interface, a set of test users with different background experience in topics like programming, NLP, and DUUI have been asked to participate in a simple assessment split into two parts. Participants start by completing a task that requires different functionalities of the web interface to be used. The experience is then evaluated in a short feedback form on the website, which consists of a total of eleven questions and statements. In addition to the four statements from the UMUX, the participants' background knowledge is also recorded. The full list of questions and statements in the form is available in table 5.1.



1. [This system's] capabilities meet my requirements.
   1 2 3 4 5 6 7
   Strongly Disagree — Strongly Agree
2. Using [this system] is a frustrating experience.
   1 2 3 4 5 6 7
   Strongly Disagree — Strongly Agree
3. [This system] is easy to use.
   1 2 3 4 5 6 7
   Strongly Disagree — Strongly Agree
4. I have to spend too much time correcting things with [this system].
   1 2 3 4 5 6 7
   Strongly Disagree — Strongly Agree

Figure 5.1. The four statements from the UMUX (Finstad 2010, p. 326) without modifications.

5.1.1. Participants

A total of 19 submissions of the feedback form have been used to evaluate the usability and overall perception of the web interface. Eight of the participants are familiar with and have used DUUI before. The same is true for experience with NLP, which is expected due to the connection between DUUI and NLP. Most participants are experienced in programming with Python being more prevalent than Java. Participants with different levels of expertise can provide a well-rounded insight into the usability and effectiveness of the web interface. This helps mitigate the potential bias that may arise from solely relying on experienced users of DUUI. The evaluation becomes more representative of the broader user base.

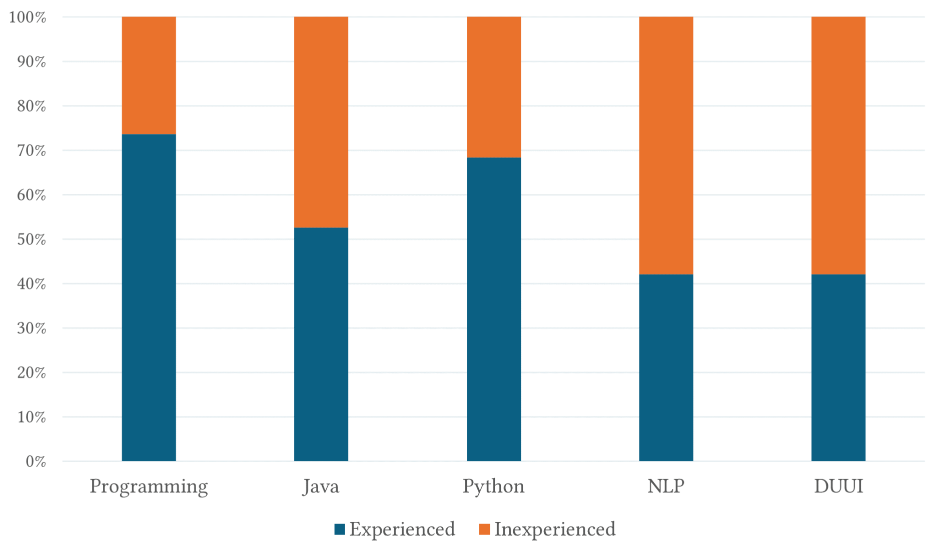


Figure 5.2. A stacked bar chart displaying the distribution of experienced and inexperienced participants in programming, NLP, and DUUI.

5.1.2. Task

Each participant has been asked to complete a specific task with the web interface. The chosen task requires most important functionalities provided by the web interface and encourages the usage of optional but performance increasing capabilities such as copying pipelines or using templates. The task consists of six steps followed by the submission of the feedback form. The individual steps are:

1. Log in with the provided credentials
2. Verify a connection to MinIO has been established or establish one with the provided credentials
3. Create a pipeline that contains spaCy as its only component
4. Start a process that analyzes text files stored in MinIO. The result should also be written to MinIO.
5. Create another pipeline that contains spaCy as its first and GerVADER as its second component.
6. Start another process that has the same input and output but only processes files that are larger than 1000 bytes.
7. Fill out and submit the feedback form.

Credentials for logging in and connecting to MinIO, the source and output path, and docker image names have been provided as part of the instructions.

Table 5.1. The questions and statements in the feedback form used for the evaluation of the web interface's perceived usability.

Question	Type
I have experience in programming	Scored
I have experience in Natural Language Processing	Scored
I will work with Natural Language Processing in the future	Yes / No
I have used DUUI before	Yes / No
Using DUUI has been pleasant with a graphical user interface	Scored
I can program in Java	Scored
I can program in Python	Scored
The website's capabilities meet my requirements	Scored
Using the website is a frustrating experience.	Scored
The website is easy to use	Scored
I have to spend too much time reading the documentation	Scored

5.2. Discussion

The web interface has generally been well received and has an average score of *5.92/7* with an exceptionally high rating of *6.5/7* by experienced users of DUUI. UMUX scores of individual participants lie between *29.17* and *95.83*, with more than two thirds of participants giving good to excellent feedback. The distribution can be seen in figure 5.3. While the feedback with a total UMUX score of *79.61* is positive, the lower ratings primarily come from participants with little or no experience in topics related to the framework. Figure 5.4 shows the average of score for the web interface and four UMUX statements split by experience with DUUI. The labels *Requirements*, *Frustration*, *Ease*, and *Correction* represent the UMUX statements seen in 5.1. The last statement (*Correction*) has been changed to represent extensive reading of documentation. This is because rather than correcting parts of the system, users were encouraged to read the documentation to complete the task. The score therefore better reflects how intuitive the website is to use and how much extra reading is required. Scores for frustration and correction are expected and desired to be lower because of the negative wording. The results show that the task has been completed correctly in most cases. Sometimes the setting for skipping files smaller than 1000 bytes in the second process has not been used. Although small errors happened, the data has been analyzed and stored in MinIO by both experienced and inexperienced users without major issues. There is, however, a notable difference in the experienced frustration with the web interface between participants with and without experience with DUUI. Knowledge about the inner workings of DUUI and flow of data appears to be an advantage that reduces the overall frustration with the system. Additionally, inexperienced participants had to spend more time reading documentation to resolve confusion and errors. Although this is expected and experience can only be gained over time, this target audience needs to be more effectively introduced to the interface and related topics, e.g. by increased guidance for users who are just getting started with NLP.

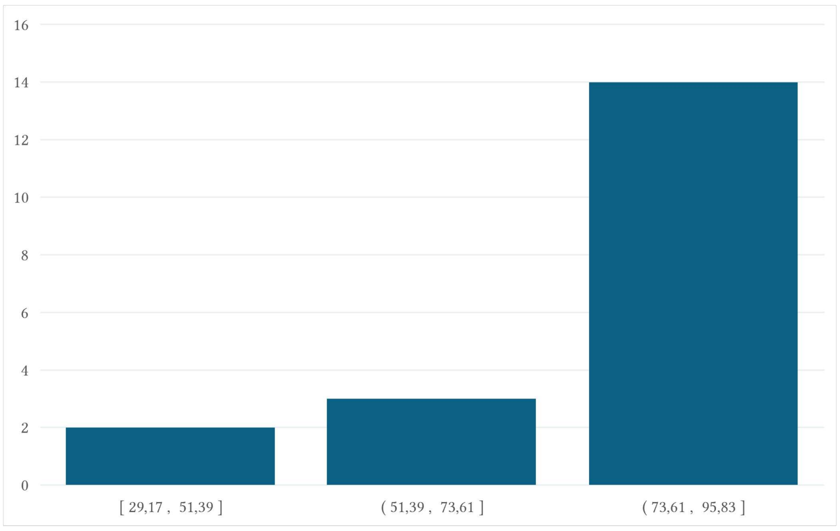


Figure 5.3. The distribution of UMUX scores in a histogram showing a clear trend towards positive reception of the web interface.

5.2.1. Accomplishments

The fact that most participants, despite varying levels of experience, were able to complete the task correctly is a great success for the project. Although the levels frustration of inexperienced users may have been higher, the web interface opened the possibility to work with NLP regardless of prior experience. Doing the same work without the web interface in Java likely would not have or hardly been possible for inexperienced users.

Because the web API is not part of the evaluation directly, no direct feedback is available. However, participants who are more proficient in Python than Java provide indirect feedback. They make up an interesting subset because of the connection to the support for different programming languages introduced in section 4.2.4. An audience of experienced Python programmers that is not or not as proficient in Java indicates that using alternative programming languages may lead to improved productivity and reduce hesitation to get started with DUUI.

Table 5.2. The average score for the UMUX statements and web interface split into experience and inexperience with programming and DUUI.

Group	Web Interface	Requirements	Frustration	Ease	Correction/ Documentation	UMUX
DUUI	6.50	6.13	1.75	5.88	1.50	86.46
No DUUI	5.45	5.45	2.45	5.18	2.27	74.62
Programming	6.29	6.21	2.00	5.86	1.64	85.12
No Programming	4.80	4.40	2.60	4.40	2.80	64.17
Average	5.89	5.74	2.16	5.47	1.95	79.61

5.2.2. Shortcomings

While the result from the UMUX can be used to get a sense of the perceived usability of the web interface, it is unclear what caused frustration or made it difficult to use for certain participants. However, to enable feedback with concrete and specific comments, an optional text field has been added at the end of the form. Based on responses by some participants the web interface is not intuitive to use in some places. Navigating through the creation of pipelines and processes has taken more time for some users, because important buttons e.g. for navigating to the next and previous steps are easy to overlook. This caused confusion as to how they should proceed. Another issue has been missing or insufficient explanation in places that use terminology from programming like for example key-value pairs. While experienced programmers understand what is being asked for, a lack of programming knowledge may cause confusion and the feature to be dismissed due to not understanding it. Some feedback responses indicate that a more comprehensive introduction for DUUI and NLP is desired for a better understanding of the system. Being unfamiliar with core concepts decreases overall

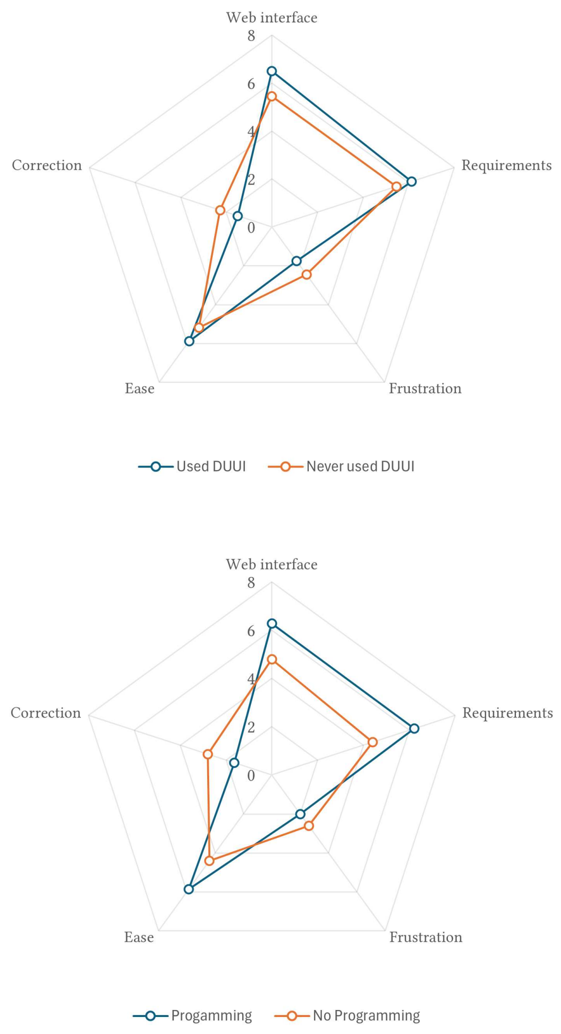


Figure 5.4. Radar charts showing the average score for the UMUX statements and web interface split into experience and inexperience with programming and DUUI.

usability and satisfaction. Adding an introduction to NLP in the documentation and extending the existing documentation of the system and its parts is therefore an important step towards improved usability.

Some of the lower ratings for the web interface did not provide a feedback message as to what went wrong or caused the frustration, making the interpretation difficult. These participants may have faced issues with understanding the task or available documentation and therefore experienced higher levels of frustration because of the system not working as expected. In such cases, participants might rush through the task without fully understanding it, leading to confusion, errors, and ultimately frustration. As a result, they may provide lower ratings due to their negative experience with the interface. The existing user guidance in form of an introductory modal dialog mentioned in 4.3.2 may have been skipped or is insufficient. Table 5.3 contains ratings for all participants with little or no programming experience[1]. Because only a small amount of data is available, the reliability is somewhat limited. Further studies and surveys should therefore be conducted after existing feedback has been taken into account.

Table 5.3. Ratings of participants with little or no programming experience highlighting the subjectivity of the evaluation.

Web Interface	Requirements	Frustration	Ease	Correction/ Documentation	UMUX
7	6	1	6	1	91.67
7	6	1	6	1	91.67
2	2	5	2	3	33.33
3	3	5	2	5	29.17
5	5	1	6	4	75.00

5.3. Conclusion

The web interface and API serve as a first step towards large scale usability for big data analysis of natural language. The graphical user interface introduced in chapter 4.3 allows inexperienced users to get started quickly without the need to invest additional time in installing software, which can be a challenge in itself. Additionally, the standalone web API enables communication with DUUI by using any programming language and therefore increasing the framework's flexibility. By leveraging the performance and capabilities of DUUI in a user-friendly environment, users can benefit from a centralized, secure, and scalable system capable of big data analysis. Chapter 5 highlights that both experienced and inexperienced users were able to use the web interface with satisfaction while also being able to mostly complete work correctly. There is however a notable difference in overall satisfaction between participants with and without experience with programming and DUUI. Experienced users, having an advantage due to being familiar with core concepts and inner workings of the framework, are less frustrated with the web interface. This indicates that a more comprehensive introduction into DUUI, NLP, and their use cases are necessary to further improve user experience and usability.

[1]The entire evaluation dataset can be found on GitHub.

6 Future Work

The web interface is a first step towards improved usability and user experience when analyzing text documents with Docker Unified UIMA interface. Essential capabilities, as described in chapter 4, have been implemented and allow for the direct control over pipelines and processes from the browser. Despite the improvement in usability, further development for both the web interface and the API are necessary to handle shortcomings and add beneficial or desired capabilities.

6.1. Notifications

During development most processes did not take long to complete so there was no need for sending notifications when errors occurred or the process finished. Because the goal is to allow for the analysis of large amounts of data, longer processing times are expected in a production environment. Having a notification system for important events during a process is mandatory at that point and adds to a positive user experience. Power Automate, a workflow automation tool introduced in section 3.1, uses an email notification system that informs users when workflows have failed. DUUI could head in a similar direction by providing error or summary emails when processes finished execution.

6.2. API Bindings

To make the framework efficient and easy to work with in programming languages other than Java, implementing API bindings for languages like Python or JavaScript could reach a large target audience. Both of these languages are used extensively and are well-liked by developers (StackOverflow 2023). Especially Python is relevant to NLP as it is one of the most used languages for machine learning related work. Additionally, as mentioned in section 5.2, users not as proficient in Java as in Python would benefit from this alternative. This can be achieved by abstracting away requests to the API by providing simple functions that hide the implementation details. Not only is the user not confronted with making requests to a server directly but is also provided with better documentation on what parameters are available as function parameters. Showing possible properties as optional function parameters is much clearer than working with raw requests that do not provide hints or autocompletion. A simple set of bindings have been written to provide an example for future implementations. These bindings are incomplete and not optimized but should nevertheless showcase the simplification that can be achieved by abstracting away raw requests. The usage can be seen in section A.1 of the appendix.

6.3. Cloud Storage

Another possible addition to the backend is support for other cloud providers like OneDrive, Google Drive, or NextCloud. Due to the simple design of the *IDUUIDocumentHandler* interface, any cloud provider that offers a way to list, read, and write files is a candidate. It also makes sense to give users the option of using several different inputs and outputs in a single process. This could include several locations from a single cloud provider as well as multiple different cloud providers.

6.4. Branching

Implementing branching in workflows to allow for an alternate flow in case of errors may be a useful feature. In cases where a component encounters an error or can not be reached, providing a backup component that can do the same or similar work prevents the failure of the entire pipeline. This is especially important for pipelines where components depend on each others outputs. For example, a component that detects the language of sentences depends upon the existence of annotations for sentences and can not function properly otherwise. Branching can be implemented with simple try-catch logic in the composer by providing a backup component that is only executed when the original component encounters an error.

6.5. Interaction with Documents

While the annotations that resulted from a process are displayed both as text and visually in charts, there is currently no way to know what part of the text the annotations reference. The generated output is not intuitive if one is not familiar with UIMA and the structure of the CAS object, but can be transformed into a more visual and intuitive representation due to it being stored in a standardized XML format. Both internal and third party software could be used to visualize the output.

Bibliography

Borin, Lars et al. (2016). “Sparv : Språkbanken ’ s corpus annotation pipeline infrastructure”. In: URL: https://api.semanticscholar.org/CorpusID:198906001.

Borthakur, Dhruba (2007). “The hadoop distributed file system: Architecture and design”. In: *Hadoop Project Website* 11.2007, p. 21.

Cardinali, Richard (1994). “Productivity improvements through the use of graphic user interfaces”. In: *Industrial Management & Data Systems* 94.4, pp. 3–7.

Datadog (2024). *See it all in one place.* https://www.datadoghq.com/product/, last accessed on 01/03/2024 11:13 MET.

E. Marcotte (2010). *Responsive Web Design.* https://alistapart.com/article/responsive-web-design/, last accessed on 16/02/2024 13:20 MET.

Ferrucci, David and Adam Lally (2004). “UIMA: an architectural approach to unstructured information processing in the corporate research environment”. In: *Natural Language Engineering* 10.3-4, pp. 327–348.

Ferrucci, David, Adam Lally, Karin Verspoor, and Eric Nyberg (Mar. 2009). *Unstructured Information Management Architecture (UIMA) Version 1.0.* OASIS Standard. URL: https://docs.oasis-open.org/uima/v1.0/uima-v1.0.html.

Fielding, Roy Thomas (2000). “Architectural Styles and the Design of Network-based Software Architectures.” PhD thesis. University of California, Irvine.

Finstad, Kraig (Sept. 2010). “The Usability Metric for User Experience”. In: *Interacting with Computers* 22, pp. 323–327. DOI: 10.1016/j.intcom.2010.04.004.

Gale, Allyson (2020). *Introducing Flyte: A Cloud Native Machine Learning and Data Processing Platform.* https://eng.lyft.com/introducing-flyte-cloud-native-machine-learning-and-data-processing-platform-fb2bb3046a59, last accessed on 01/03/2024 11:12 MET.

Grafana (2024). *Why Grafana?* https://grafana.com/grafana/?plcmt=footer, last accessed on 01/03/2024 11:12 MET.

Ken W. Alger (2022). *Introduction to the MongoDB Aggregation Framework.* https://www.mongodb.com/developer/products/mongodb/introduction-aggregation-framework/, last accessed 09/02/2024 10:24 MET.

Leonhardt, Alexander, Giuseppe Abrami, Daniel Baumartz, and Alexander Mehler (Dec. 2023). “Unlocking the Heterogeneous Landscape of Big Data NLP with DUUI”. In: *Findings of the Association for Computational Linguistics: EMNLP 2023.* Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, pp. 385–399. DOI: 10.18653/v1/2023.findings-emnlp.29. URL: https://aclanthology.org/2023.findings-emnlp.29.

Liddy, Elizabeth D (2001). “Natural language processing”. In.

Lopez, Crhistian Michael Ramos, Jhon Edinson Castro Lopez, Alberto Bravo Buchely, and Dayner Felipe Ordoñez Lopez (1998). “Ergonomic requirements for office work with visual display terminals (VDTs) —”. In: URL: https://api.semanticscholar.org/CorpusID:14947257.

MinIO, Inc. (2022). *High Performance Multi-Cloud Object Storage*. https://min.io/resources/docs/MinIO-High-Performance-Multi-Cloud-Object-Storage.pdf, last accessed on 27/01/2024, 12:57 MET.

Nadkarni, Prakash M, Lucila Ohno-Machado, and Wendy W Chapman (Sept. 2011). “Natural language processing: an introduction”. In: *Journal of the American Medical Informatics Association* 18.5, pp. 544–551. ISSN: 1067-5027. DOI: 10.1136/amiajnl-2011-000464. eprint: https://academic.oup.com/jamia/article-pdf/18/5/544/5962687/18-5-544.pdf. URL: https://doi.org/10.1136/amiajnl-2011-000464.

Prometheus (2024). *Metric Types*. https://prometheus.io/docs/concepts/metric_types/, last accessed on 01/03/2024 11:13 MET.

Spark Framework (2024). *Spark - A micro framework for creating web applications in Kotlin and Java 8 with minimal effort*. https://sparkjava.com/, last accessed on 30/01/2024, 09:11 MET.

StackOverflow (2023). *2023 Developer Survey*. https://survey.stackoverflow.co/2023/, last accessed on 27/01/2024, 12:57 MET.

Turnbull, J. (2018). *Monitoring with Prometheus*. James Turnbull. ISBN: 9780988820289.

Zapier (2024). *Zapier newsroom*. https://zapier.com/press, last accessed on 01/03/2024 11:12 MET.

A Appendix

A.1. Code Examples

The following code example shows an example usage of a MinIO document handler with Java. The handler is then used by a *DUUIDocumentReader* both as the input and output handler. Other settings are provided to filter documents before processing. Below the code is an excerpt of events that are printed to the console because the *DebugLevel* for the composer has been set to *DEBUG*.

```
DUUIComposer composer = new DUUIComposer()
        .withLuaContext(new DUUILuaContext().withJsonLibrary())
        .withSkipVerification(true)
         // Prints all events with a DebugLevel >= DEBUG
        .withDebugLevel(DUUIComposer.DebugLevel.DEBUG)
        .withWorkers(5)
        .withIgnoreErrors(true);

// Create a DUUIMinioDocumentHandler with an endpoint, username and password
DUUIMinioDocumentHandler minio = new DUUIMinioDocumentHandler(
    endpoint,
    username,
    password
);

DUUIDocumentReader reader = DUUIDocumentReader
    .builder(composer)
    .withInputHandler(minio)
    .withInputPath("input/sample_txt")
    .withInputFileExtension(".txt")
    .withOutputHandler(minio)
    .withOutputPath("output/xmi")
    .withOutputFileExtension(".xmi")
    .withRecursive(true)              // Look for documents recursively
    .withSortBySize(true)             // Sort files in ascending order
    .withCheckTarget(true)            // Filter already processed documents
    .withAddMetadata(true)
    .withMinimumDocumentSize(1024 * 3) // Files must be at least 3 kB in size
    .build();

composer.addDriver(new DUUIUIMADriver());
composer.addDriver(new DUUIDockerDriver());

composer.add(new DUUIUIMADriver.Component(
    createEngineDescription(BreakIteratorSegmenter.class))
    .withName("Tokenizer"));

composer.add(new DUUIDockerDriver.Component(
    "docker.texttechnologylab.org/gervader_duui:latest")
    .withName("GerVADER"));

composer.run(reader, "example-minio");

/**Excerpt from the debug output of events in the process.
 * Timestamp     [SENDER]   : Message
 *
 * 1707413608359 [READER]   : Skip files smaller than 3072 bytes.
 * 1707413608359 [READER]   : Number of files before skipping 17.
 * 1707413608360 [READER]   : Number of files after skipping 13.
 * 1707413608361 [READER]   : Sorted files by size in ascending order
 * 1707413608361 [READER]   : Checking output location output/xmi for existing documents.
 * 1707413608372 [READER]   : Found 0 documents in output location.
 *                            Keeping all files from input location.
 * 1707413608372 [READER]   : Processing 13 files.
 *
 * 1707413628184 [READER]   : Decoding document input/sample_txt/sample_14_92120.txt
 * 1707413628184 [READER]   : Document input/sample_txt/sample_14_92120.txt
 *                            decoded after 0 ms
 * 1707413628184 [READER]   : Deserializing document input/sample_txt/sample_14_92120.txt
 * 1707413628190 [READER]   : Document input/sample_txt/sample_14_92120.txt
 *                            deserialized after 6 ms
 * 1707413628190 [DOCUMENT]: Starting to process input/sample_txt/sample_14_92120.txt
 * 1707413628190 [DOCUMENT]: input/sample_txt/sample_14_92120.txt
 *                            is being processed by component Tokenizer
 * 1707413628212 [DOCUMENT]: input/sample_txt/sample_14_92120.txt
 *                            has been processed by component Tokenizer
 * 1707413628212 [DOCUMENT]: input/sample_txt/sample_14_92120.txt
 *                            is being processed by component GerVADER
 * 1707413629297 [DOCUMENT]: input/sample_txt/sample_09_34261.txt
 *                            has been processed by component GerVADER
 * 1707413629300 [READER]   : Uploading document input/sample_txt/sample_09_34261.txt
 * 1707413629357 [DOCUMENT]: input/sample_txt/sample_09_34261.txt
 *                            has been processed after 3370 ms
 * 1707413629357 [COMPOSER]: 5 Documents have been processed
 */
```

Python Bindings The following code example uses Python to create a pipeline, instantiate it and start a process. All these actions performed with Python are also reflected in the web interface. This code is part of a small implementation of API bindings for Python. The full code can be found on GitHub at PythonClient.

```python
from duui.client import DUUIClient
from duui.config import API_KEY

CLIENT = DUUIClient(API_KEY)


def main() -> None:
    my_pipeline = CLIENT.pipelines.create(
        name="My Pipeline",
        components=[
            {
                "name": "Tokenizer",
                "tags": ["Token", "Sentence"],
                "description": """Split the document into Tokens and Sentences
                  using the DKPro BreakIteratorSegmenter AnalysisEngine.""",
                "driver": "DUUIUIMADriver",
                "target": "de.tudarmstadt.ukp.dkpro.core.tokit.BreakIteratorSegmenter",
            },
            {
                "name": "GerVADER",
                "description": """GerVADER is a German adaptation of the sentiment
                  classification tool VADER. Classify sentences into positive,
                  negative or neutral statements.""",
                "tags": ["Sentiment", "German"],
                "driver": "DUUIDockerDriver",
                "target": "docker.texttechnologylab.org/gervader_duui:latest",
                "options": {"scale": 2, "use_GPU": True},
            },
        ],
        description="""This pipeline has been created using the API with Python.
        It splits the document text into Tokens and Sentences
        and then analyzes the Sentiment of these Sentences.""",
        tags=["Python", "Sentence", "Sentiment"],
    )

    pipeline_id = my_pipeline.get("oid")
    # Instantiate the pipeline so it can be used multiple times
    # without the need to restart Docker components
    CLIENT.pipelines.instantiate(pipeline_id)
    # Start a process that finds .txt files with a minimum size of 500 bytes
    # recursively start from the /input directory in Dropbox.
    CLIENT.processes.start(
        pipeline_id,
        input={
            "provider": "Dropbox",
            "path": "/input",
            "file_extension": ".txt"
        },
        output={
            "provider": "Dropbox",
            "path": "/output/python",
            "file_extension": ".txt",
        },
        recursive=True,
        sort_by_size=True,
        minimum_size=500,
        worker_count=3
    )


if __name__ == "__main__":
    main()
```

The status and progress of a process and its documents can be retrieved by using the schedule Python library. *Schedule* provides a simple API for the execution of tasks in a regular interval. In this case, process and document information is retrieved and printed to the console every five seconds.

```
...
ID = create_process(...)
# Code from the previous example has been omitted to reduce the space taken up.

import schedule
import sys
import time


def update() -> None:
    process = CLIENT.processes.findOne(ID)
    result = CLIENT.processes.documents(
        ID, status_filter=["Failed"], include_count=True
    )
    total = len(process["document_names"])

    print(
        f"""Progress: {round(process['progress'] / total * 100)}%
        \t{result['count']} Documents have failed.\r""",
        end="",
    )

    if process["status"] in ["Completed", "Failed", "Cancelled"]:
        print(
            f"""Progress: {round(process['progress'] / total * 100)}
            %\t{result['count']} Documents have failed.""",
        )

        print(f"Process finished with status {process['status']}.")
        sys.exit(0)

schedule.every(5).seconds.do(update)

while True:
    schedule.run_pending()
    time.sleep(1)
```

A.2. Process Flow Chart

The following flow chart represents a process and all its stages. When an exception occurs, the process is interrupted and its status set to *Failed*. This can happen at any stage indicated by the line on the left side.

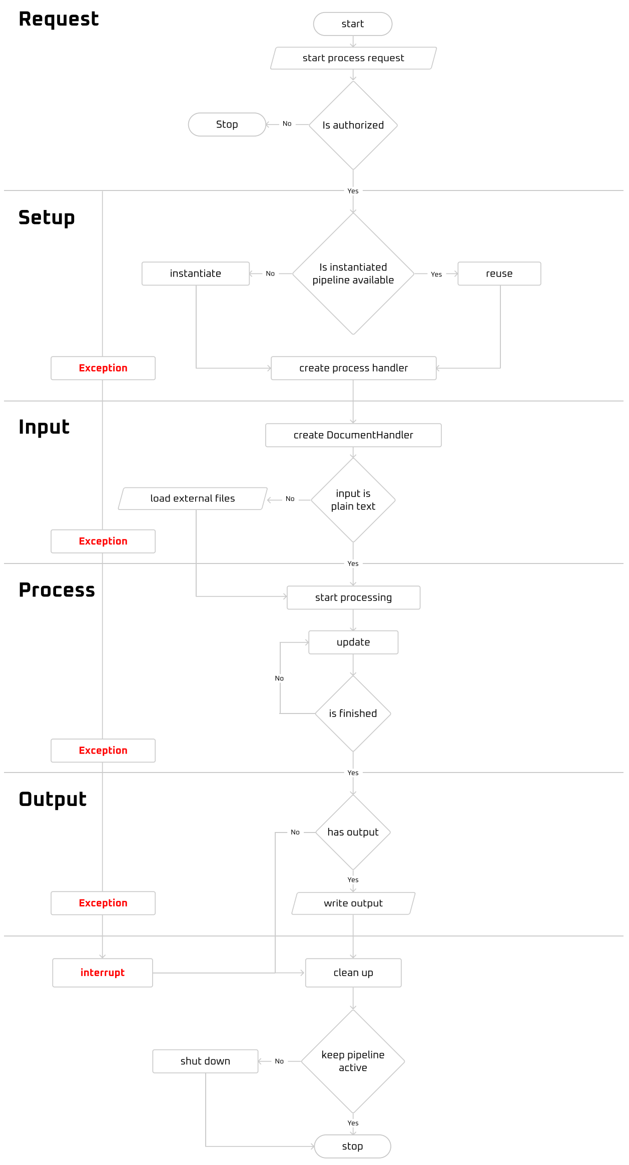
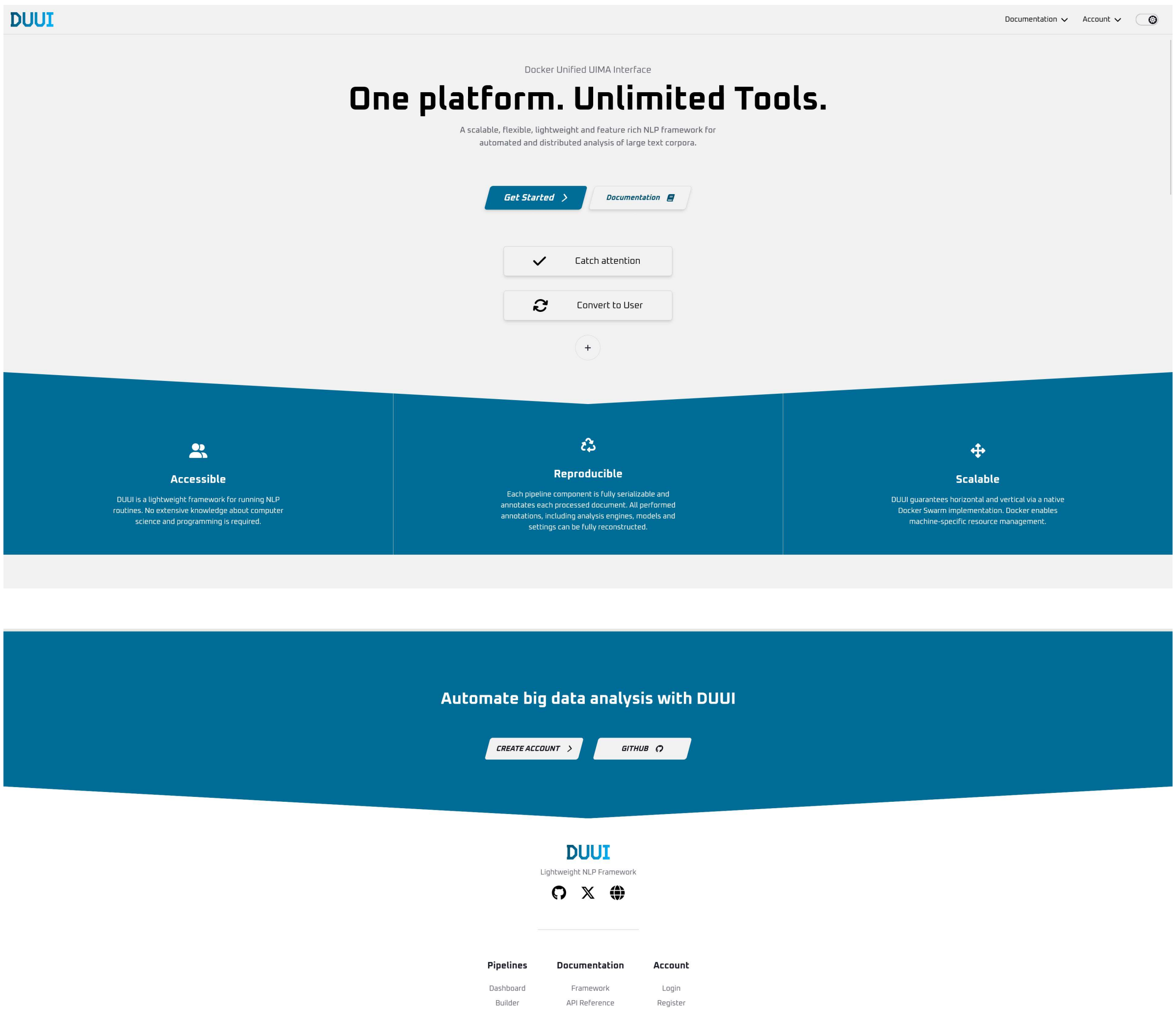


Figure A.1. Flow Chart for a process. The process can be cancelled at any stage by sending a PUT request to */processes/:id* and providing the id of the process. Cancelling has been omitted in the chart for clarity.

A.3. Web Interface

This section includes additional screenshots and illustrations from the web interface not part of chapter 4.3. For a live demonstration visit `https://duui.texttechnologylab.org/`.



One platform. Unlimited Tools.

A scalable, flexible, lightweight and feature rich NLP framework for automated and distributed analysis of large text corpora.

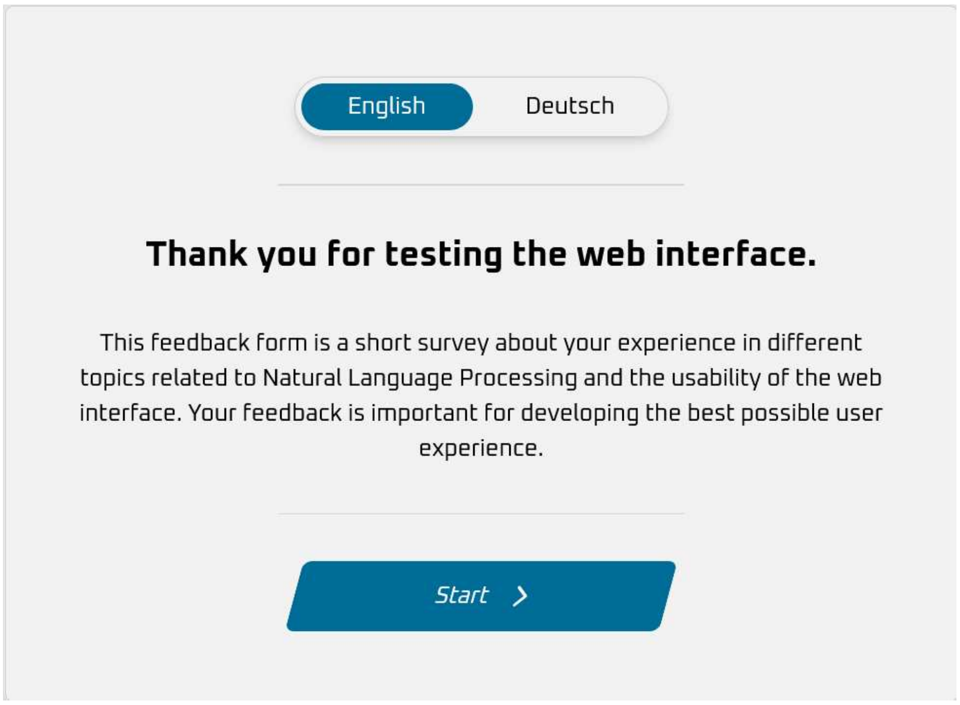
DUUI is a lightweight framework for running NLP routines. No extensive knowledge about computer science and programming is required.

Each pipeline component is fully serializable and annotates each processed document. All performed annotations, including analysis engines, models and settings can be fully reconstructed.

DUUI guarantees horizontal and vertical via a native Docker Swarm implementation. Docker enables machine-specific resource management.

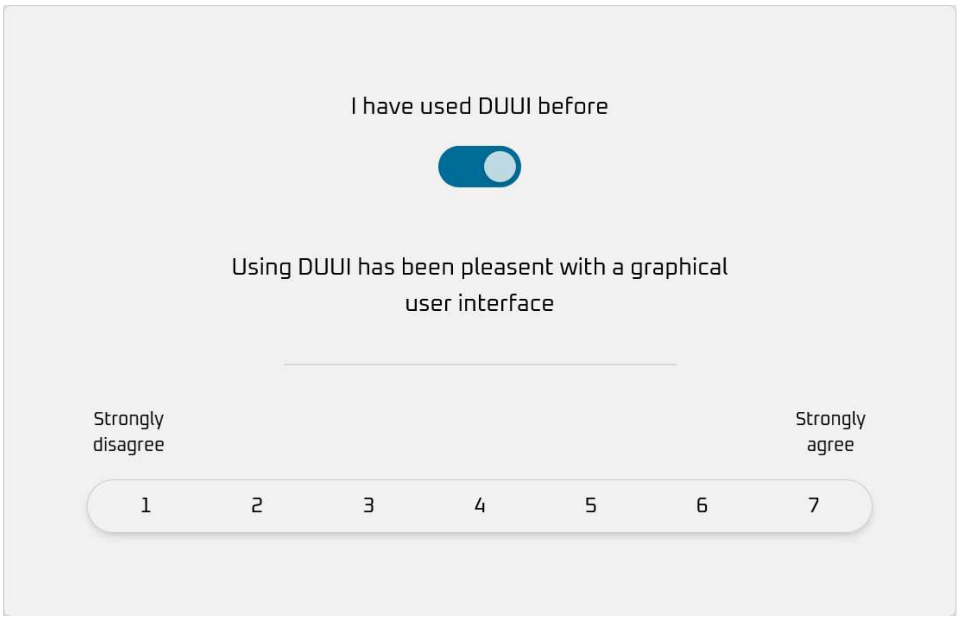
Automate big data analysis with DUUI

Figure A.2. The home page of the web interface including the hero and footer sections.



Thank you for testing the web interface.

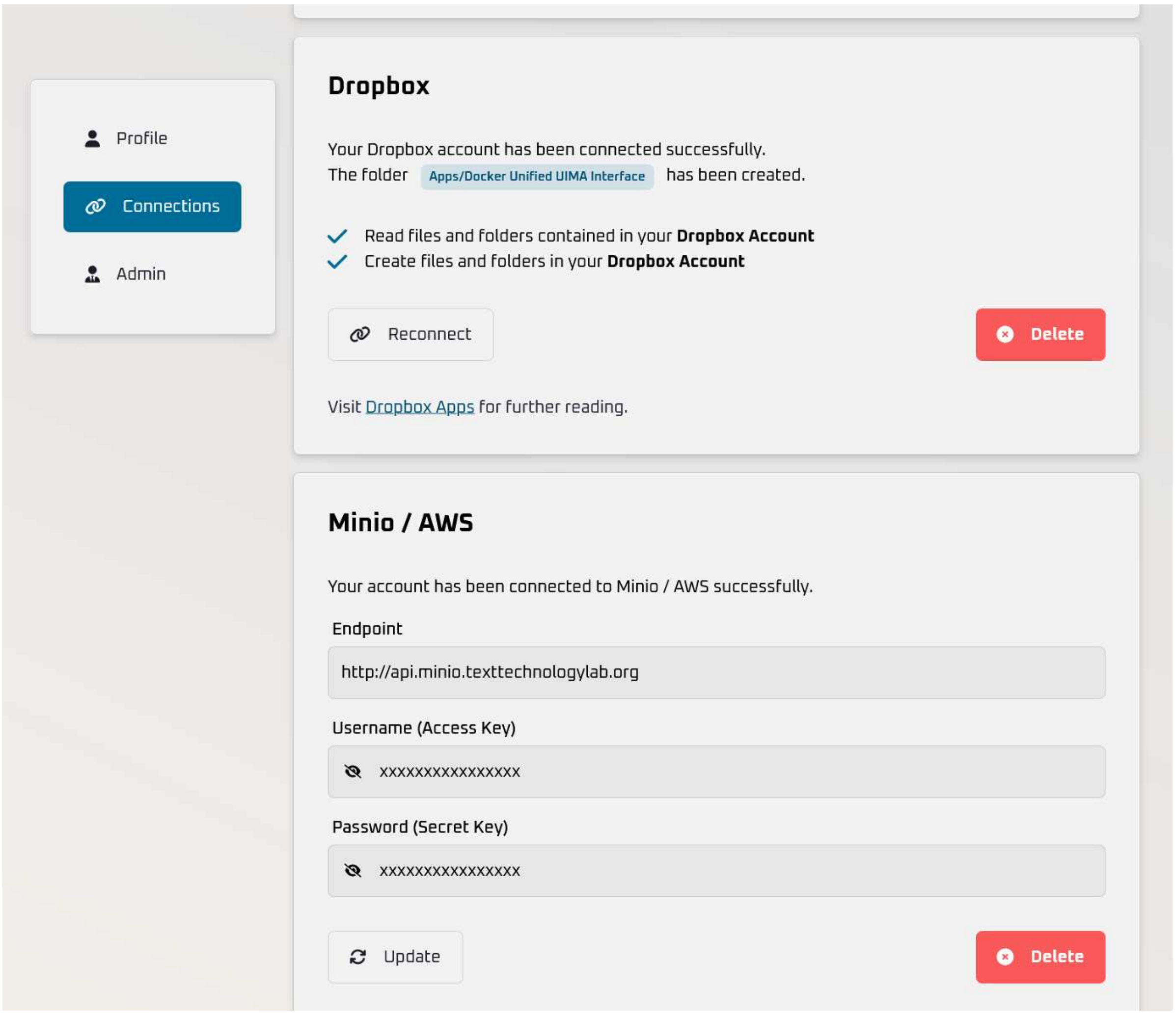
This feedback form is a short survey about your experience in different topics related to Natural Language Processing and the usability of the web interface. Your feedback is important for developing the best possible user experience.



I have used DUUI before

Using DUUI has been pleasent with a graphical user interface

Figure A.3. Introduction to the feedback form available in English and German and an example for a question posed in the feedback survey described in section 5.1. This is one of eleven total question and statements.



Dropbox

Your Dropbox account has been connected successfully.
The folder Apps/Docker Unified UIMA Interface has been created.

✓ Read files and folders contained in your **Dropbox Account**
✓ Create files and folders in your **Dropbox Account**

Visit Dropbox Apps for further reading.

Minio / AWS

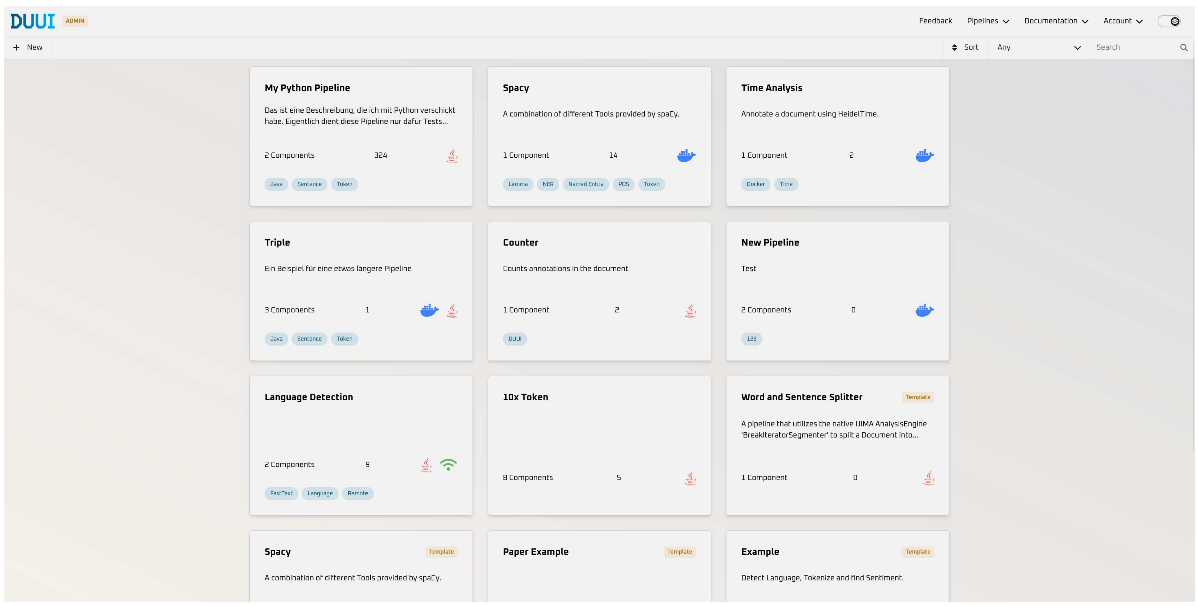
Your account has been connected to Minio / AWS successfully.

Endpoint: http://api.minio.texttechnologylab.org

Username (Access Key): xxxxxxxxxxxxxxxx

Password (Secret Key): xxxxxxxxxxxxxxxx

Figure A.4. The account page for an admin user. Connections to Dropbox and MinIO have been established.



My Python Pipeline
Das ist eine Beschreibung, die ich mit Python verschickt habe. Eigentlich dient diese Pipeline nur dafür Tests...
2 Components 324
Java Sentence Token

Spacy
A combination of different Tools provided by spaCy.
1 Component 14
Lemma NER Named Entity POS Token

Time Analysis
Annotate a document using HeidelTime.
1 Component 2
Docker Time

Triple
Ein Beispiel für eine etwas längere Pipeline
3 Components 1
Java Sentence Token

Counter
Counts annotations in the document
1 Component 2
DUUI

New Pipeline
Test
2 Components 0
123

Language Detection
2 Components 9
FastText Language Remote

10x Token
8 Components 5

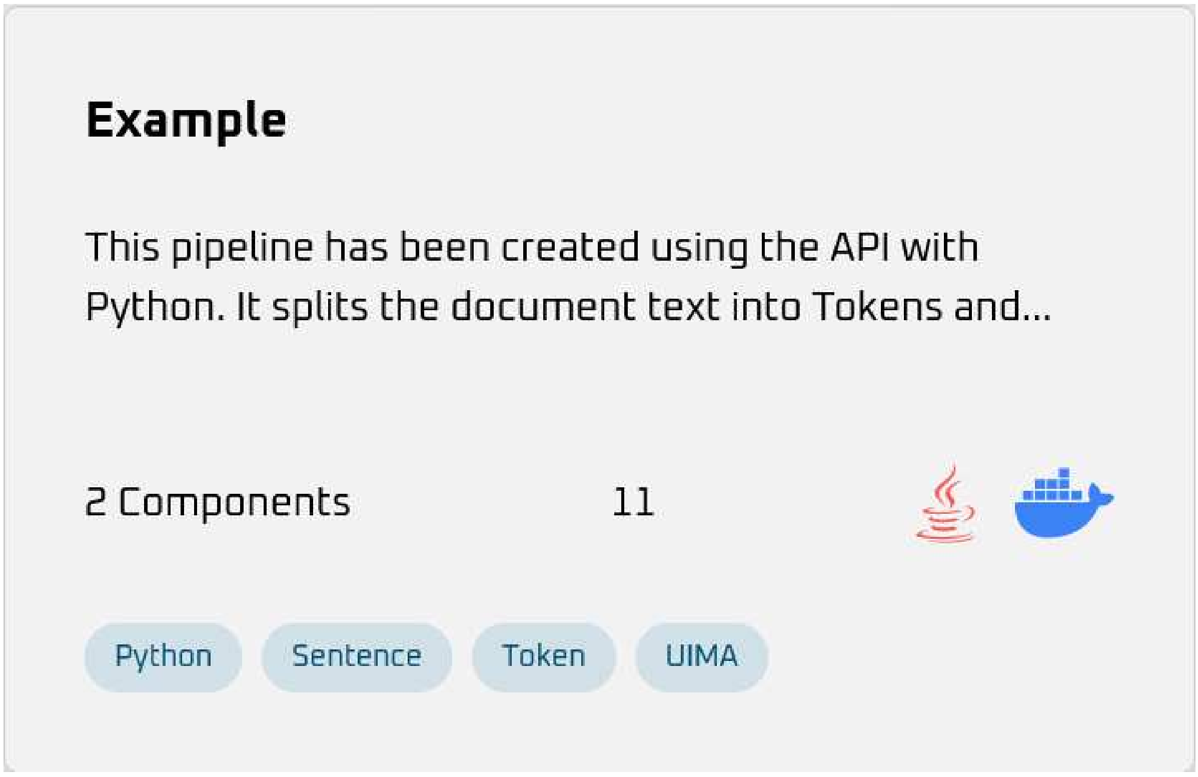
Word and Sentence Splitter Template
A pipeline that utilizes the native UIMA AnalysisEngine 'BreakIteratorSegmenter' to split a Document into...
1 Component 0

Spacy Template
A combination of different Tools provided by spaCy.

Paper Example Template

Example Template
Detect Language, Tokenize and find Sentiment.

Figure A.5. The dashboard that displays a grid of intractable card elements. Each card represents a pipeline that has been created by the user. Users with the *admin* role can also view pipeline templates.



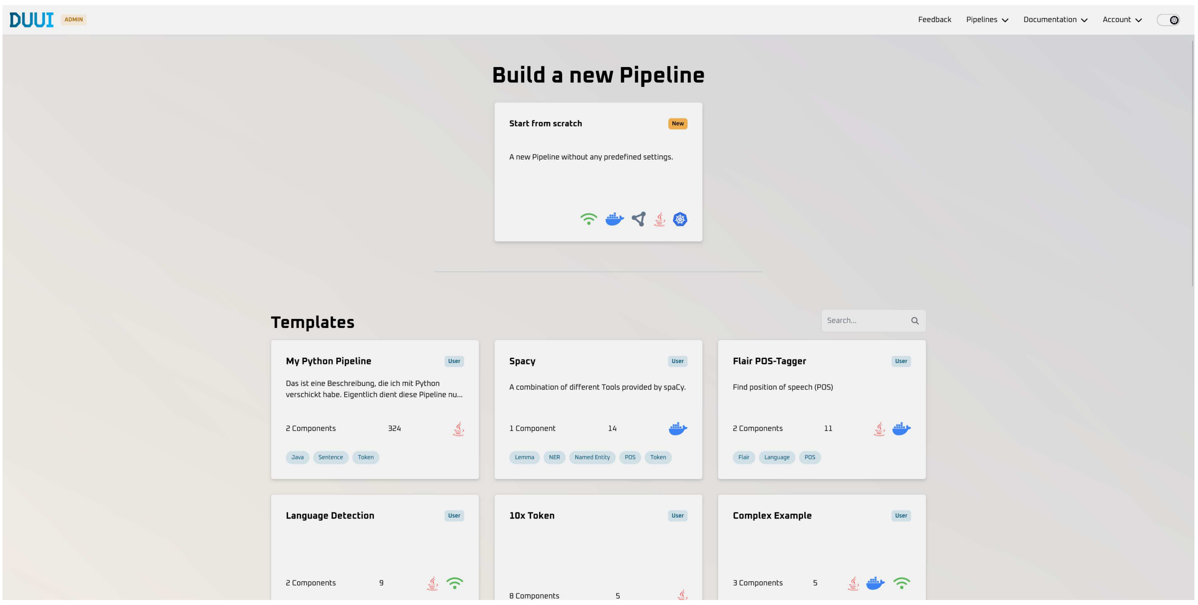
Example

This pipeline has been created using the API with Python. It splits the document text into Tokens and...

2 Components 11

Python Sentence Token UIMA

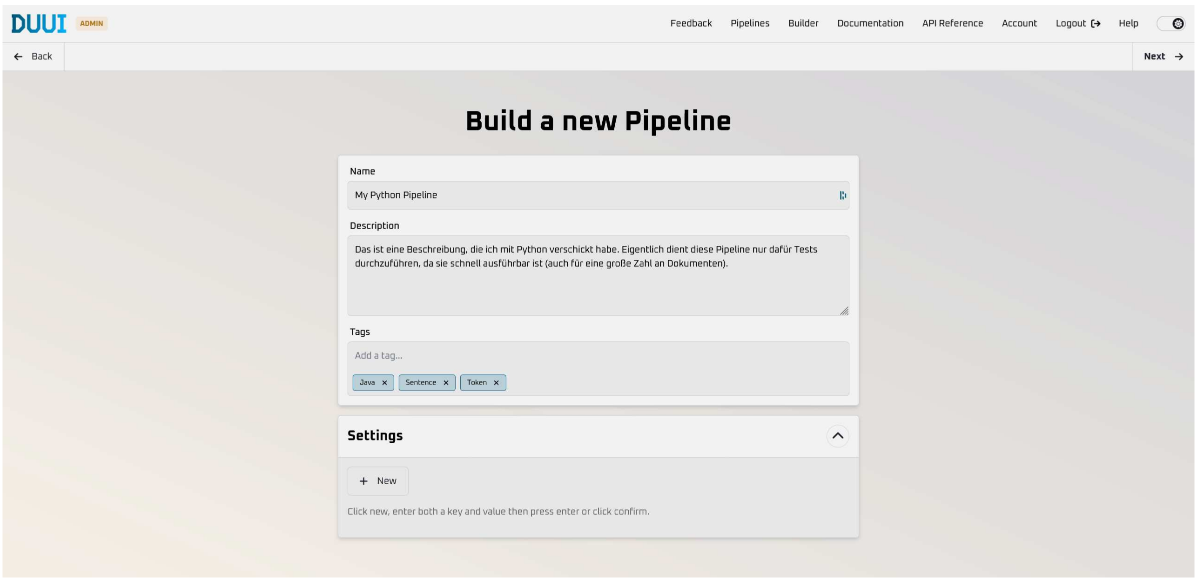
Figure A.6. A card element representing a pipeline.



A new Pipeline without any predefined settings.

My Python Pipeline	Das ist eine Beschreibung, die ich mit Python verschickt habe. Eigentlich dient diese Pipeline nu...	2 Components 324	Java, Sentence, Token
Spacy	A combination of different Tools provided by spaCy.	1 Component 14	Lemma, NER, Named Entity, POS, Token
Flair POS-Tagger	Find position of speech (POS)	2 Components 11	Flair, Language, POS
Language Detection		2 Components 9	
10x Token		8 Components 5	
Complex Example		3 Components 5	

Figure A.7. The first step in the creation of a new pipeline is asking the user whether to start from scratch or from a template that can either be a user defined pipeline created earlier or provided by DUUI.



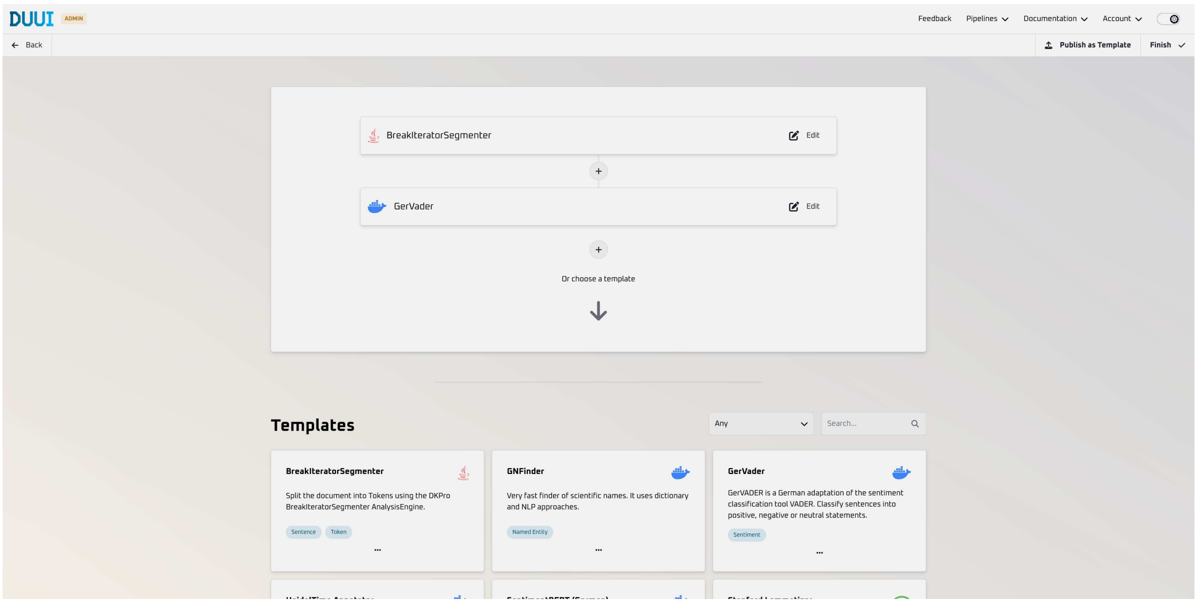
Name: My Python Pipeline

Description: Das ist eine Beschreibung, die ich mit Python verschickt habe. Eigentlich dient diese Pipeline nur dafür Tests durchzuführen, da sie schnell ausführbar ist (auch für eine große Zahl an Dokumenten).

Tags: Add a tag... Java × Sentence × Token ×

Click new, enter both a key and value then press enter or click confirm.

Figure A.8. Second step of the pipeline builder used to set pipeline specific properties.

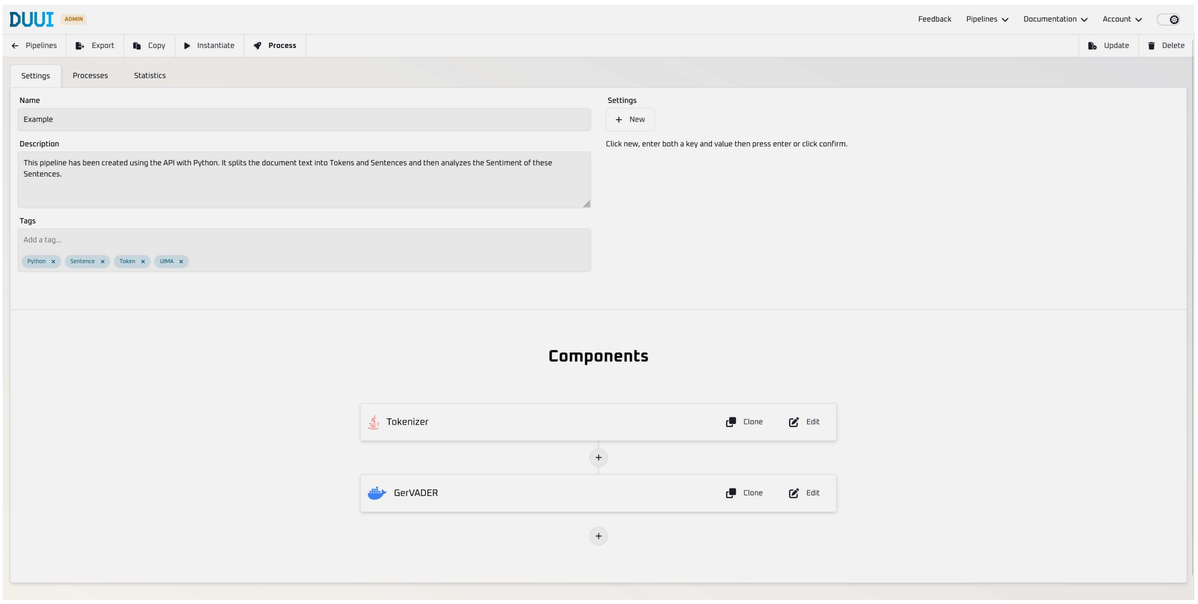


BreakIteratorSegmenter
Split the document into Tokens using the DKPro BreakIteratorSegmenter AnalysisEngine.
Sentence Token

GNFinder
Very fast finder of scientific names. It uses dictionary and NLP approaches.
Named Entity

GerVader
GerVADER is a German adaptation of the sentiment classification tool VADER. Classify sentences into positive, negative or neutral statements.
Sentiment

Figure A.9. In the final step, components can be added, removed, and edited. The order of components can be changed via drag & drop.



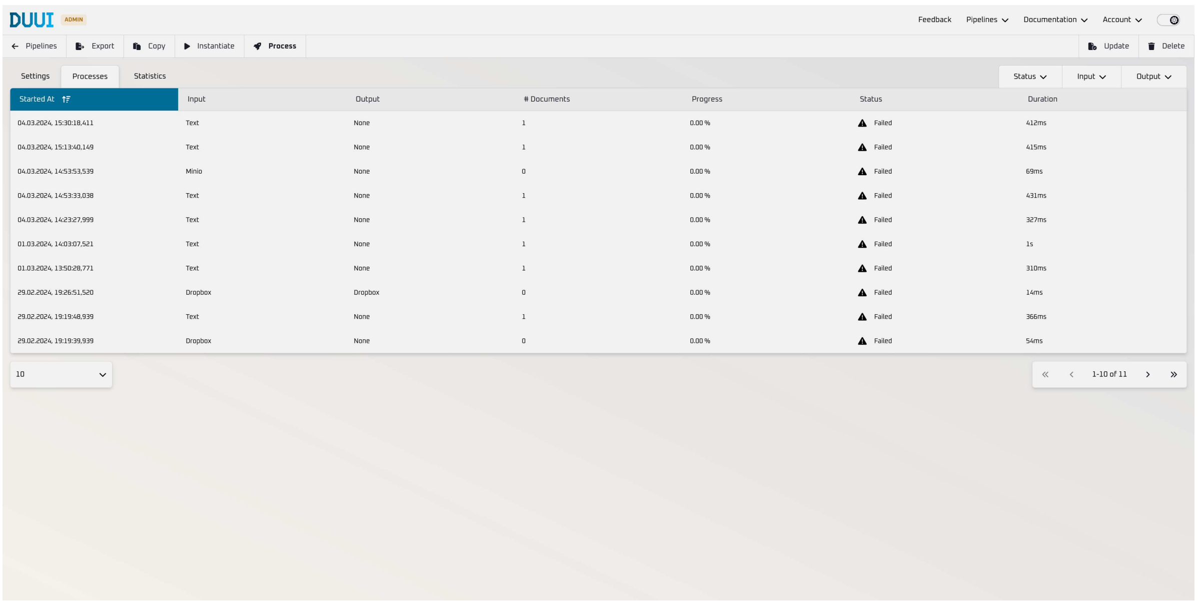
Name
Example

Description
This pipeline has been created using the API with Python. It splits the document text into Tokens and Sentences and then analyzes the Sentiment of these Sentences.

Tags
Add a tag...
Python Sentence Token UIMA

Settings
New
Click new, enter both a key and value then press enter or click confirm.

Figure A.10. Settings tab for a pipeline



Started At	Input	Output	# Documents	Progress	Status	Duration
04.03.2024, 15:30:18,411	Text	None	1	0.00 %	Failed	412ms
04.03.2024, 15:13:40,149	Text	None	1	0.00 %	Failed	415ms
04.03.2024, 14:53:53,539	Minio	None	0	0.00 %	Failed	69ms
04.03.2024, 14:53:33,038	Text	None	1	0.00 %	Failed	431ms
04.03.2024, 14:23:27,999	Text	None	1	0.00 %	Failed	327ms
01.03.2024, 14:03:07,521	Text	None	1	0.00 %	Failed	1s
01.03.2024, 13:50:28,771	Text	None	1	0.00 %	Failed	310ms
29.02.2024, 19:26:51,520	Dropbox	Dropbox	0	0.00 %	Failed	14ms
29.02.2024, 19:19:48,939	Text	None	1	0.00 %	Failed	366ms
29.02.2024, 19:19:39,939	Dropbox	None	0	0.00 %	Failed	54ms

Figure A.11. Processes tab for a pipeline

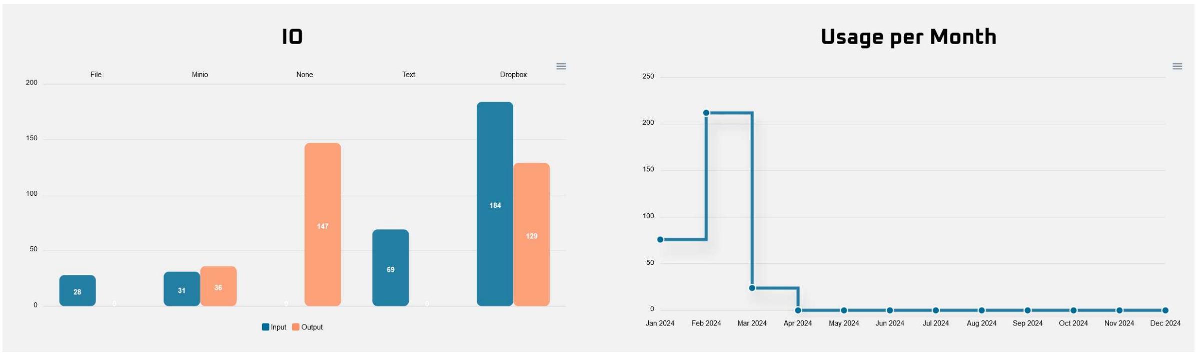
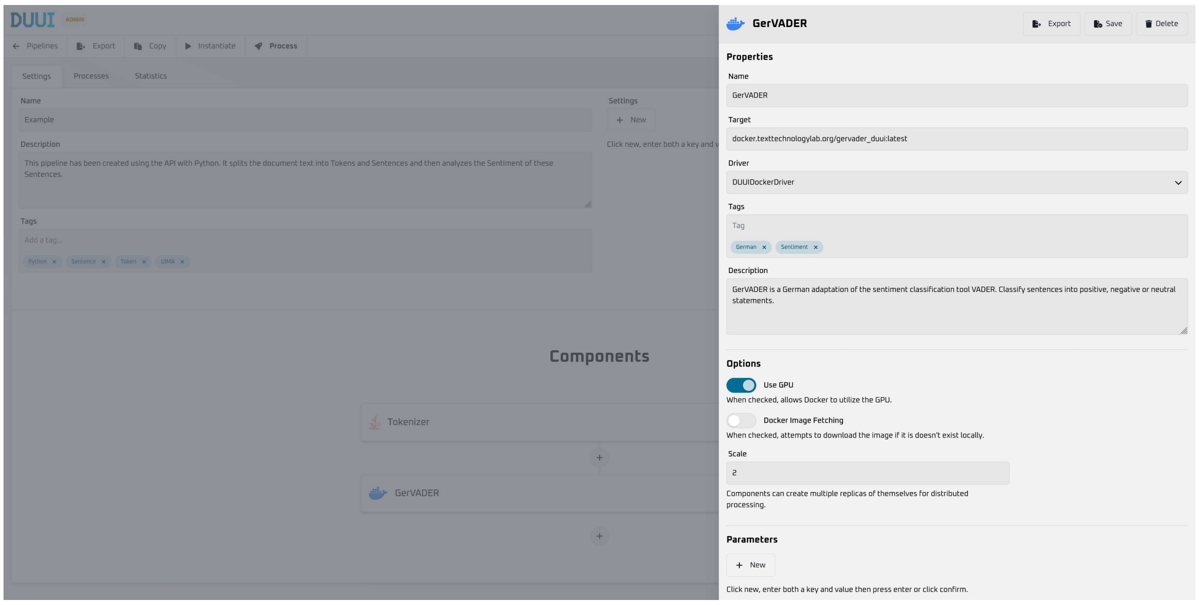


Figure A.12. Two of the four charts used to visualize statistics for a pipeline.



Name: Example

Description: This pipeline has been created using the API with Python. It splits the document text into Tokens and Sentences and then analyzes the Sentiment of these Sentences.

Tags: Add a tag... Python, Sentence, Token, UIMA

Settings: Click new, enter both a key and v

GerVADER

Properties

Name: GerVADER

Target: docker.texttechnologylab.org/gervader_duui:latest

Driver: DUUIDockerDriver

Tags: Tag German, Sentiment

Description: GerVADER is a German adaptation of the sentiment classification tool VADER. Classify sentences into positive, negative or neutral statements.

Options

Use GPU: When checked, allows Docker to utilize the GPU.

Docker Image Fetching: When checked, attempts to download the image if it is doesn't exist locally.

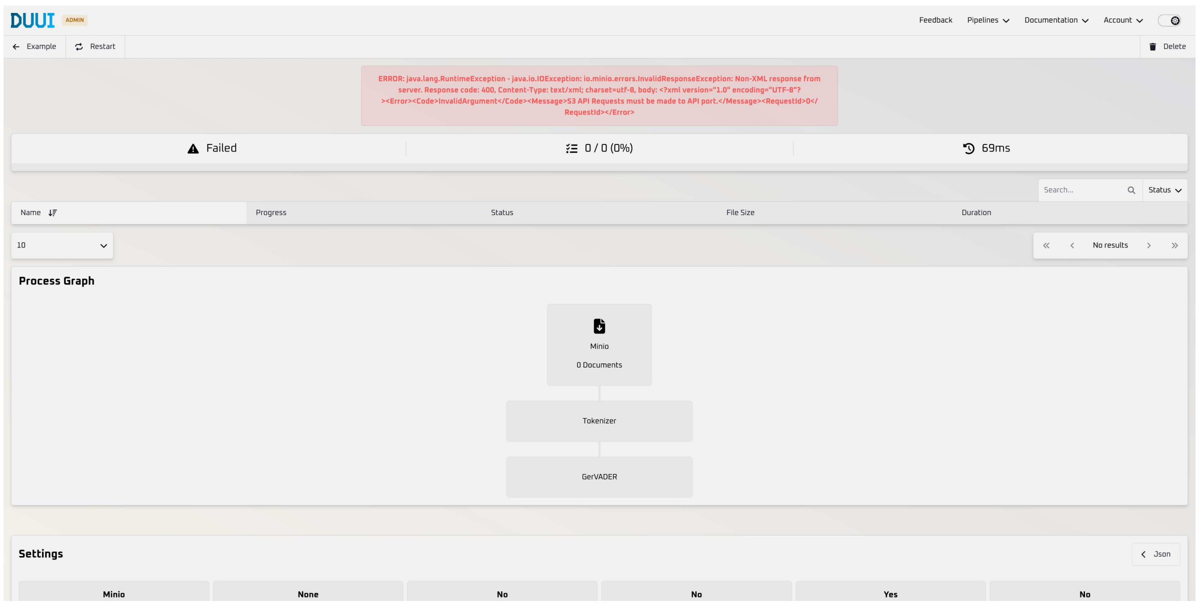
Scale: 2

Components can create multiple replicas of themselves for distributed processing.

Parameters

Click new, enter both a key and value then press enter or click confirm.

Figure A.13. A component named *GerVADER* using the *DUUIDockerDriver* is edited in a sidebar drawer.



ERROR: java.lang.RuntimeException - java.io.IOException: io.minio.errors.InvalidResponseException: Non-XML response from server. Response code: 400, Content-Type: text/xml; charset=utf-8, body: <?xml version="1.0" encoding="UTF-8"?><Error><Code>InvalidArgument</Code><Message>S3 API Requests must be made to API port.</Message><RequestId>0</RequestId></Error>

Failed | 0 / 0 (0%) | 69ms

Name	Progress	Status	File Size	Duration

10 No results

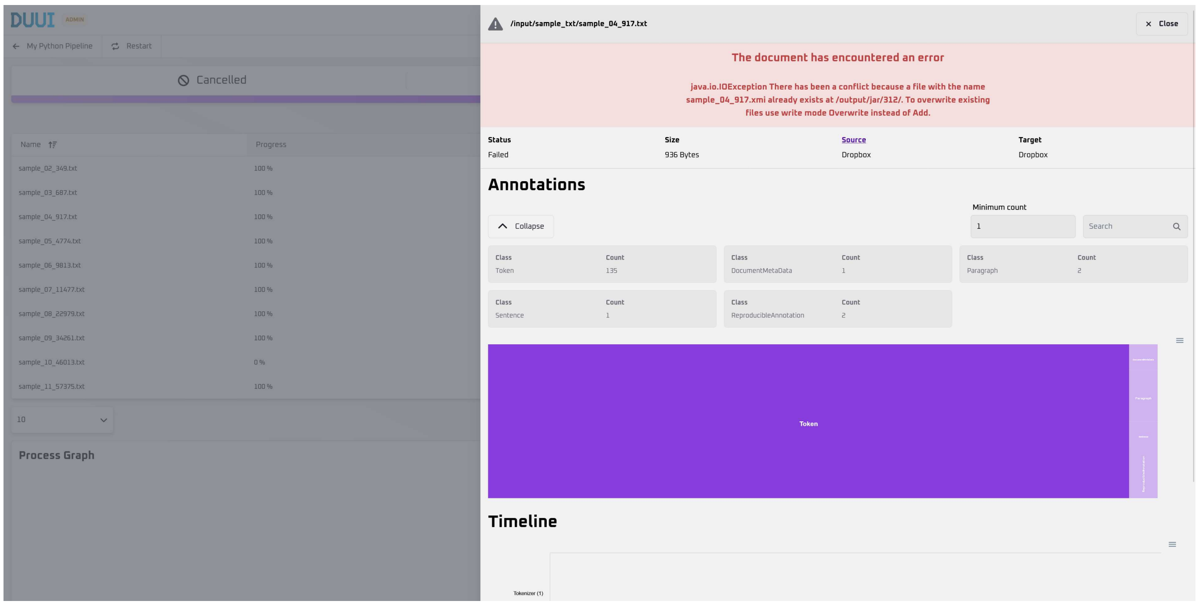
Process Graph

Minio
0 Documents
Tokenizer
GerVADER

Settings

Minio	None	No	No	Yes	No

Figure A.14. A process that completed successfully.



Cancelled

Name	Progress
sample_02_349.txt	100 %
sample_03_687.txt	100 %
sample_04_917.txt	100 %
sample_05_4774.txt	100 %
sample_06_9813.txt	100 %
sample_07_11477.txt	100 %
sample_08_22979.txt	100 %
sample_09_34261.txt	100 %
sample_10_46013.txt	0 %
sample_11_57375.txt	100 %

/input/sample_txt/sample_04_917.txt

The document has encountered an error

java.io.IOException There has been a conflict because a file with the name sample_04_917.xmi already exists at /output/jar/312/. To overwrite existing files use write mode Overwrite instead of Add.

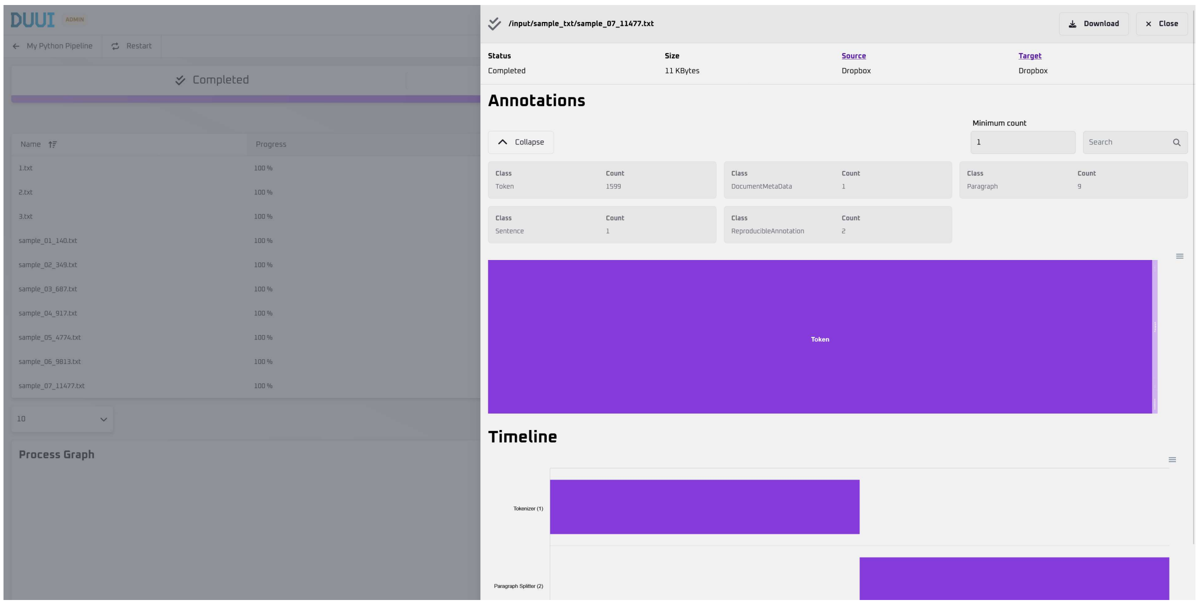
Status	Size	Source	Target
Failed	936 Bytes	Dropbox	Dropbox

Annotations

Minimum count: 1

Class	Count
Token	135
DocumentMetaData	1
Paragraph	2
Sentence	1
ReproducibleAnnotation	2

Figure A.15. A document that encountered an error.



Completed

Name	Progress
1.txt	100 %
2.txt	100 %
3.txt	100 %
sample_01_140.txt	100 %
sample_02_349.txt	100 %
sample_03_687.txt	100 %
sample_04_917.txt	100 %
sample_05_4774.txt	100 %
sample_06_9813.txt	100 %
sample_07_11477.txt	100 %

/input/sample_txt/sample_07_11477.txt

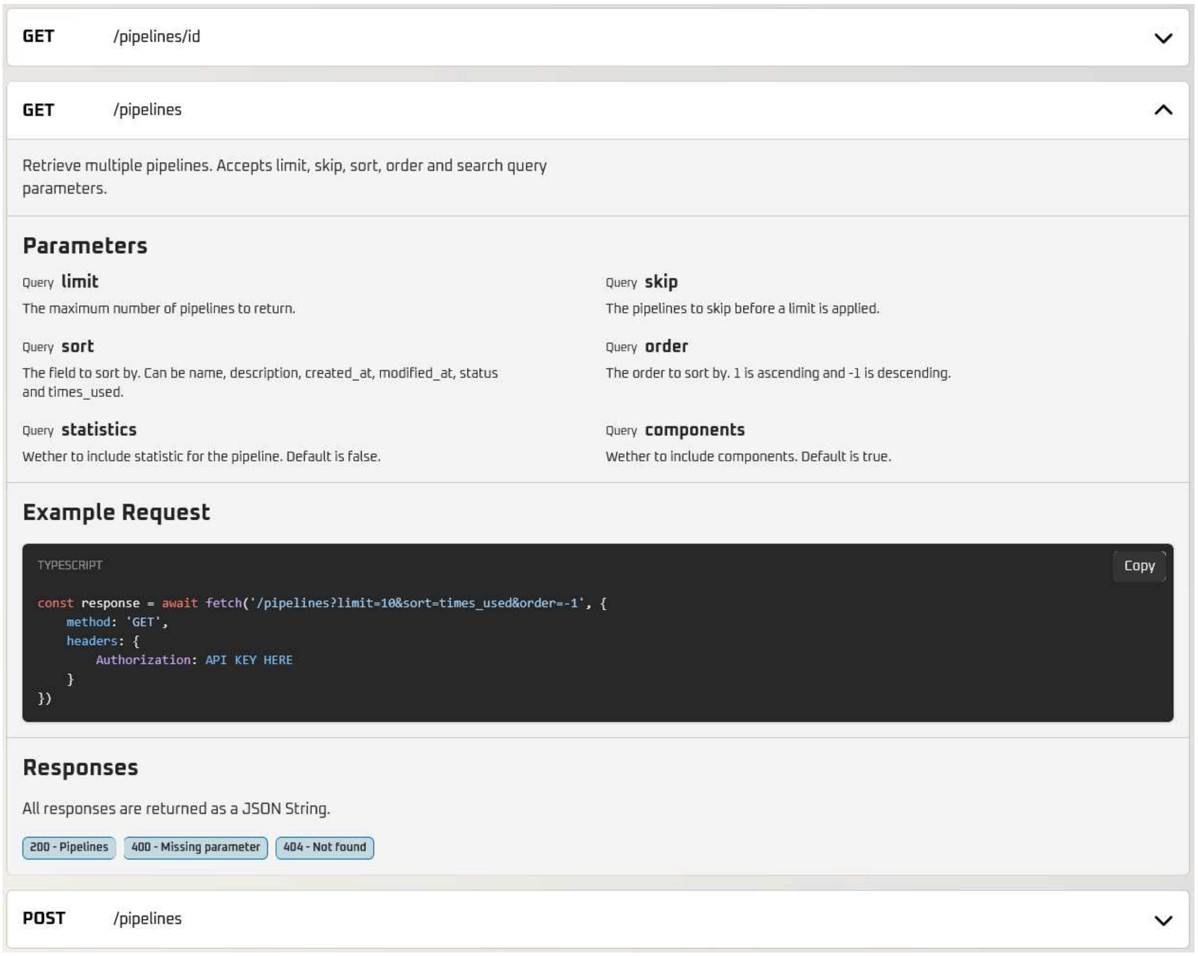
Status	Size	Source	Target
Completed	11 KBytes	Dropbox	Dropbox

Annotations

Minimum count: 1

Class	Count
Token	1599
DocumentMetaData	1
Paragraph	9
Sentence	1
ReproducibleAnnotation	2

Figure A.16. A document that completed successfully. The download button on the top right allows a direct download of the processed file from the target location. The annotations that have been added to a document are displayed in text format and in a treemap. The chart reacts to filters.


GET /pipelines/id

GET /pipelines

Retrieve multiple pipelines. Accepts limit, skip, sort, order and search query parameters.

Parameters

Query **limit**
The maximum number of pipelines to return.

Query **skip**
The pipelines to skip before a limit is applied.

Query **sort**
The field to sort by. Can be name, description, created_at, modified_at, status and times_used.

Query **order**
The order to sort by. 1 is ascending and -1 is descending.

Query **statistics**
Wether to include statistic for the pipeline. Default is false.

Query **components**
Wether to include components. Default is true.

Example Request

TYPESCRIPT

```
const response = await fetch('/pipelines?limit=10&sort=times_used&order=-1', {
    method: 'GET',
    headers: {
        Authorization: API KEY HERE
    }
})
```

Responses

All responses are returned as a JSON String.

Figure A.17. Example for the documentation of endpoints in the web interface.