Srđan Ljepojević

Web application for alignment of sentence pairs from parallel corpora

Bachelor's Thesis

to achieve the university degree of

Bachelor of Science

Bachelor's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Ass.Prof. Dipl.-Ing. Dr.techn. Roman Kern

Institute for Interactive Systems and Data Science
Head: Univ.-Prof. Dipl.-Inf. Dr. Stefanie Lindstaedt

Graz, January 2021

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present bachelor's thesis.

14.02.2021

Date

Signature

Abstract

Text simplification as a field is steadily increasing, but a lot of the problems are still far from being solved. The application which will be described in this thesis, aims to help with the one of the problems of text simplification, which is gathering enough data to train the models for automatic text simplification. This application should provide the framework for easier visualization, effortless editing and exporting of the annotated data, from articles written in German. The process of development and important design decisions were noted in the thesis, together with functional requirements and use cases. Main focuses of the application, during the development, was performance, easiness of use for novice users and correctness of the outputted data. To test these quality attributes and to evaluate if functional requirements are fulfilled, user testing was conducted. The results showed that all of the set requirements were fulfilled and users highly praised the performance and usability of the system, with some minor remarks as to how the system can be further improved.

Contents

List of Figures

1. Introduction

Text simplification is a process of making some text easier to understand while still making it grammatically correct and without leaving out any important information.

In the early days, motivation for text simplification (TS) was to help systems that use natural language as an input [5]. Such systems are for example parsers, which sometimes failed if complex text was used. Additionally, Chandrasekar, Doran and Srinivas [5] also mention systems which perform machine translation, summarization and informational retrieval that could also benefit from simpler inputs.

Since then, some researchers focused their work towards helping specific target groups for example children [6], non-native speakers, and low-literacy readers [2, 18]. Furthermore, in the research done by Cao et al. [3], it is shown that TS could also be beneficial to bridge the knowledge gap between laymen and experts. Text simplification is also useful for people with disabilities like autism [7], dyslexia [13], or for people who suffered brain damage (aphasia[1]) and cannot process the language the same way as a healthy individual could [4].

TS is usually done with syntactic and/or lexical simplification [16]. The former method includes the elimination of multiple embedded prepositional and relative phrases, removing the passive voice and generally the replacement of longer sentences with two or more shorter ones [4]. The latter approach includes replacing of difficult words with more common synonyms or adding the dictionary definition to difficult words [15].

The survey, done by Shardlow [14], shows that the text simplification as a field is steadily increasing. Despite that, the problem of TS is far from being solved and even though there are a lot of automatic methods for output evaluation (BLEU,

[1] https://en.wikipedia.org/wiki/Aphasia

Flesch–Kincaid, SARI), these simplified texts still need human intervention if they are meant to be used in practice.

The primary goal of this thesis is to help the researchers and language experts with the evaluation of different outputs produced by automatic text simplification. The web application presented in this thesis can import data, display, side-by-side, the original and the simplified versions of the text, and export matching sentence pairs from both texts. These sentence pairs are important as they will be used for model training in the future. Moreover, it is possible to edit both simplified and original text.

The source of the data that is used to develop and test this application is from APA - Austrian Press Agency. This data set consists of news articles in German, which contains both complex and simplified versions of the text. The application which is described in this thesis is not limited to this data. As long as there is dataset with two levels of text complexity it is possible to implement a custom parser, which transforms texts from data to a applications database.

Although the aim of this work is not to develop any new state-of-the-art algorithms, which could solve some simplification problem, it will provide the framework for easier visualization, effortless editing and exporting of the annotated data.

2. Related Work

2.1. Background

2.1.1. XML

Extensible Markup Language (XML) [1] is a popular text-based format which is used to store some arbitrary data. To be considered a genuine XML file, the file has to follow some strict set of rules which were defined in specifications made by World Wide Web Consortium. For example, there should only be one root element, all opening tags must have a closing tags and should be nested correctly etc. If the XML processor finds any irregularities with the file syntax, it is automatically flagged as erroneous and the processing of the XML file is stopped. This might seem really brutal, but this means that computer can read it reliably.

2.1.2. XML DOM Parsing

Even though there is a strict specification about how a XML file should look like, there is no official specification as to how to actually access the data. One of interfaces which is used is Document Object Model (DOM) [2]. It is an interface which is not bound to any specific language and is just an idea on how data from the XML file could be accessed. DOM represents data in a tree structure and each node of that tree is a one piece of that document. This is really useful, not only for navigation, but also for visualization of HTML or XML documents, as looking at

[1] https://www.w3.org/standards/xml/core#uses (Accessed on: 2020-11-30)
[2] https://en.wikipedia.org/wiki/DocumentObjectModel (Accessed on: 2020-11-30)

the source code of these documents can be overwhelming, especially when there are a lot of tags in the document.

XML DOM allows users to access elements and navigate the XML like a tree in all directions (parent, siblings, children). The Listing 2.1 shows a XML file and Figure 2.1 its DOM tree representation. This file is taken from a dataset used in this thesis. Some parts of the file are removed and some parts cleaned up to make the code more readable and to help with understanding. More about how useful it could be to represent XML file like a tree is discussed in Chapter 4.2.2.

```
<?xml version="1.0" encoding="UTF-8"?>
<Documents>
  <DOC TYPE="INSERT" NAME="APA_20200615_APA0361" SRC="APA" RECDATE
    ="20200615" LANG="GER">
    <HEAD>
      <FELD NAME="TITEL">
        <P ID="1">Nachrichten leicht verstaendlich (in der
    Sprachstufe B1) - Audio =
        </P>
      </FELD>
      <FELD NAME="INHALT">
        <P ID="1">Seit Montag gelten Lockerungen bei der
    Maskenpflicht
        </P>
        <P ID="2">Wien - Ab Montag gelten die neuen Lockerungen
    ...
        </P>
        <P ID="3">Die Masken-Pflicht in Oesterreich hatte am 30.
    Maerz begonnen. ...
        </P>
    </HEAD>
    <PART ID="1">
      <FELD NAME="MELDUNGSTYP">
        <P ID="1">Ueberblick</P>
      </FELD>
    </PART>
  </DOC>
  <DOC>
    <HEAD>
      <FELD>
        <P>...</P>
      </FELD>
      <FELD>
        <P>...</P>
```

```
            <P> . . . </P>
        </HEAD>
        <PART>
            <FELD>
                <P> . . . </P>
            </FELD>
        </PART>
    </DOC>
</Documents>
```

Listing 2.1: Cleaned up XML file taken from the data set used to develop the application described in this work

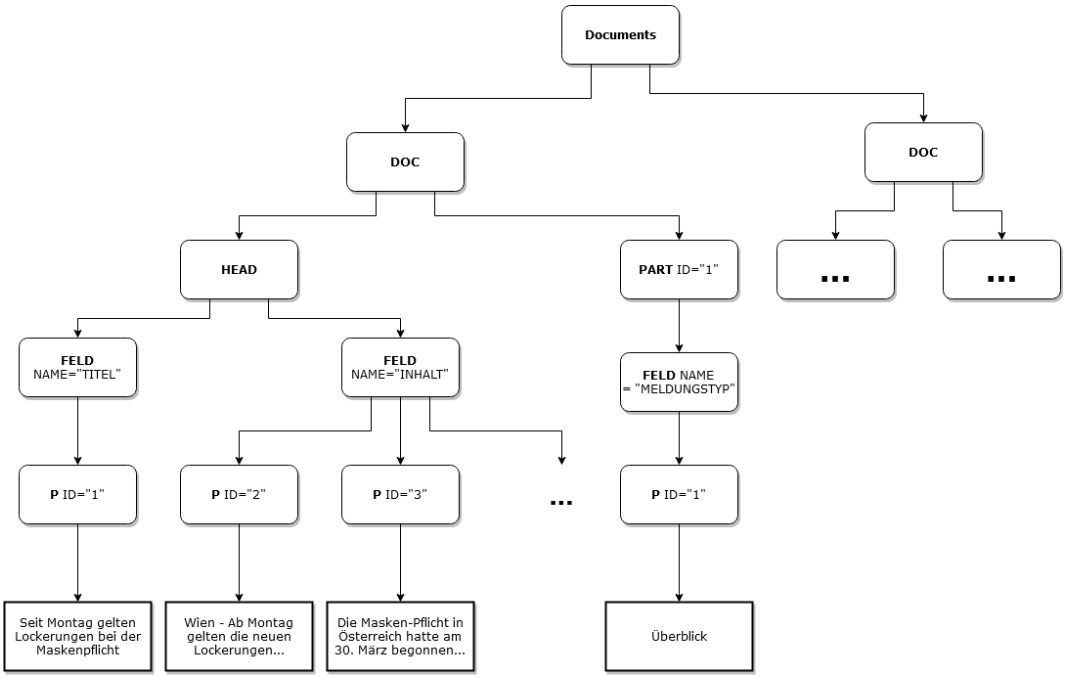


Figure 2.1.: XML tree representation of the Listing 2.1. The rectangles with rounded edges represent XML tags, while rectangles without rounded edges represent values of XML tags. There is one root element with two DOC children nodes. Every DOC node represents one language level of the article. DOC node consists of HEAD and PART nodes. Both of these nodes have FELD nodes as children, the only difference is the attribute name. Actual contents of the articles are stored in children nodes of FELD node with the name "INHALT". It is possible to access the article texts by using XML DOM parser and going down the tree.

2.1.3. JavaScript

JavaScript (JS) is a high level programming language, which is, along side HTML and CSS, one of the core technologies of the World Wide Web [3]. Building websites with only HTML and CSS is possible, but these websites will be static, which means that the data, which is displayed on the website, is hard-coded somewhere in the files. Including JavaScript into HTML makes the website dynamic, which means that the website can show some interactive content, communicate with servers, react to user input without reloading the website and a lot more.

2.1.4. React

React (React.js or ReactJS) [4] is an open-source JavaScript framework used for building graphical user interfaces for web applications. React is component-based [5], meaning everything that is shown on screen, is some kind of component. Components can be really primitive like Buttons, Lists, Forms, but these components can then be combined into more complex ones, which can also have some state. Components can also have really complex render methods, which means that depending on the state of that component, different elements can be displayed.

Further benefit of using components is that it can easily be reused, which makes programming in React really fast and clutter free. If two components look the same, but only have to display different information, the same component can be reused just with different data, which is passed onto the component. Another benefit of using React as a tool for building user interfaces is that when some state of the application changes, only components, which rely on it, are updated and rendered to the screen again. This provides a huge performance boost.

[3] https://en.wikipedia.org/wiki/JavaScript (Accessed on: 2020-11-30)
[4] https://en.wikipedia.org/wiki/React(webframework) (Accessed on: 2020-11-30)
[5] https://reactjs.org/docs/components-and-props.html

2.1.5. JSON

JavaScript Object Notation (JSON) [6] is a text format based on JavaScript. Even though it is based on JavaScript, it is completely independent and consists of data structures like collections of name/value pairs and lists of values. Because a lot of programming languages support these structures, it is a perfect format for storing data. It is also easy for humans to read and write. Additionally, it is also easy for machines to parse and generate.

2.1.6. Metadata

Metadata is ”data that provides information about other data” [7]. In the application described in this thesis, every article has metadata about the sentence pairs which is then exported in a JSON file. Every sentence pair consists of matching sentences, but also a metadata field, which describes the score of sentence similarity, whether it was matched by an algorithm or by a human, and more. Further information about the JSON format, that this application uses is written in Chapter 4.

2.1.7. Docker

Docker is an open source application which manages Docker containers. These containers have all the dependencies, configuration files and more, which allow applications, that are stored in them, to run on wide variety of machines. This solves one huge problem in development and deployment process, which is making sure that the application works not only on developers local machine, but also on other machines, servers etc. Placing an application into Docker containers and configuring them properly solves this problem, because Docker guarantees that the environment which was set once, will always be the same, no matter the underlying system.

[6] https://www.json.org/json-en.html (Accessed on: 2020-11-30)
[7] https://www.merriam-webster.com/dictionary/metadata (Accessed on: 2020-11-30)

2.1.8. Spring Boot

Spring [8] is one of the most popular open source Java frameworks in the world, that provides Java with some additional features and modules, making programming in Spring easier and faster.

Spring Boot is an extension of the Spring framework, which further simplifies programming. Even though Spring solves some issues with overly complicated configuration of Java, Spring Boot takes this to another level by providing the default configurations and some 'starter' dependencies. This means that for many use cases, the developers do not need to fiddle with the setup, but can get up and running in a matter of minutes. Moreover a lot of stuff works "out of the box", without any redundant code or external libraries.

2.1.9. Promise based HTTP client

Promised based HTTP clients receive promises after the client sends requests. These promises [9] can have three states: Pending, Fulfilled and Rejected. Fulfilled promises means that the value that is promised is returned to the client, rejected means that the reason for failure is returned and pending means that the client has to wait for the answer.

2.2. State of the Art

2.2.1. Text simplification

Text simplification, as the name suggests, should simplify texts as a whole, but most of the research which was done focused on simplification of the individual sentences, which could in fact make the whole texts easier to understand. This

[8] https://www.baeldung.com/spring-vs-spring-boot (Accessed on: 2020-11-30)

[9] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/GlobalObjects/Promise (Accessed on: 2021-01-03)

approach allows for easier data acquiring, which is really important if automatic algorithms are to be developed.

Datasets Most popular datasets used to measure the performance of the text simplification are PWKP and Turk Corpus.

PWKP is a dataset collected by Zhu, Bernhard and Gurevych [23]. Data was acquired from more than 65 thousand articles from Wikipedia and Simple Wikipedia, which is aimed at the children, students, second language learners or people with disabilities. To align sentence pairs, monolingual sentence alignment, with TF*IDF sentence-level similarity measure, was applied to Wikipedia articles, which leads to a dataset consisting of 108 thousand sentence pairs. To also account for sentence splitting, the dataset does not consist of only 1:1 mappings of ”complex” and ”simple” sentences but also 1:N, meaning that one ”complex” sentence could have multiple simpler ones.

TurkCorpus is another dataset used for the evaluation of the of the text simplification. Xu et al. [21] took sentences from the Normal Wikipedia, a subset of PWKP, which they gave to the workers at Amazon Mechanical Turk [10], a crowdsourcing marketplace, to be rewritten manually. The result is a dataset consisting of 2350 sentences, where each sentence has 8 simplifications. The dataset is then further split into 2000 sentences used for training and 350 used for result evaluation.

It is important to mention the research done by Xu, Callison-Burch and Napoles [20], where it was argued that the PWKP dataset is not actually simple and that about 50% of the sentence pairs are not real simplifications. Xu, Callison-Burch and Napoles [20] also present a new dataset called Newsela. This dataset consists of 1,130 articles in 5 levels. These articles, and their respective simpler versions, were written by professional editors. The problem with this dataset is that, even though it is free, it is forbidden to redistribute, which means sentence alignments are not available to the public.

Metrics Most popular metrics used for measuring simplicity are SARI (System Output Against References And Input Sentence), developed by Xu et al. [21] (2016) and BLEU, developed by Papineni et al. [11] (2002).

[10] https://www.mturk.com/ Amazon Mechanical Turk

Text simplification approaches Earlier work on text simplification focused on handcrafted rules which did not perform really good and as field matured, more modern, data-driven, approaches came to light. The current state of the art approaches mainly consider text simplification as a machine translation or neural sequence-to-sequence problems.

Two machine translation approaches which deliver the best scores, on the PWKP dataset, are PMBT-R and Hybrid approach.

PMBT-R, developed by Wubben, Bosch and Krahmer [19], uses phrase based machine translation to simplify sentences. PMBT-R uses Moses [11] software and GIZA++ [12] aligner to train the model and align the phrase pairs into a phrase table. To further improve the simplification of the sentences, the results of the text simplification are re-ranked according to their dissimilarity to the source sentence. Out of N best results, one is chosen which represents the best, according to PMBT-R, simplification result. According to the survey done by Alva-Manchego, Scarton and Specia [1], this approach scored the best on TurkCorpus dataset according to the BLEU metric.

Hybrid approach, developed by Narayan and Gardent [10], combines deep semantics and monolingual machine translation to simplify sentences. What differentiates this approach to others is that deep semantic representation (DRS) is used as an input, instead of sentences or parse trees. Hybrid approach consists of three models which aim to simplify the texts; DRS simplification model, which is applied first and is responsible for splitting of the sentences and deletion of the words, PBMT (phrase based machine translation) model used for finding the suitable substitutions and finally language model (LM), which is there to keep the simplified sentences grammatically correct. According to the aforementioned survey [1], Hybrid approach delivers the best result, on the PWKP dataset, according to the SARI score, while having close second BLEU score on the same dataset.

Despite having high score on the PWKP dataset, Hybrid [1] approach did not perform as well on the TurkCorpus and was outperformed by all other approaches described in this work. Alva-Manchego, Scarton and Specia [1] point out that this might be because metrics, on the TurkCorpus dataset, reward paraphrasing more

[11] http://www.statmt.org/moses/ Moses Software
[12] http://www.statmt.org/moses/giza/GIZA++.html GIZA++

than deletion of words and sentence splitting, as these paraphrased sentences bear closer resemblance to the reference solutions from the TurkCorpus dataset.

The same survey shows that approaches, which use either Paraphrase Database (PPDB) or Simple PPDB, usually have the best SARI scores when TurkCorpus test set is used.

Paraphrase Database (PPDB) is database consisting of over 220 million paraphrase pairs collected by Ganitkevitch, Van Durme and Callison-Burch [8]. Simple PPDB, collected by Pavlick and Callison-Burch [12], is a subset of PPDB consisting of 4.5 million paraphrase pairs. These databases are used as an additional knowledge about paraphrasing complicated words and phrases.

Two highest scoring approaches, which utilize either PPDB or SPPDB, are DMASS-DCSS and SBSMT(PPDB+SARI).

DMASS-DCSS, developed by Zhao et al. [22], models text simplification as a neural sequence-to-sequence problem. It consists of two models Deep Critic Sentence Simplification (DCSS) and Deep Memory Augmented Sentence Simplification model (DMASS). DCSS's purpose is to choose the simplified words with the help of the loss function. On the other, DMASS is a simplification model with the augmented memory to store key-value pairs acquired from SPPDB. Even though they are presented as a two separate models, Zhao et al. [22] show that using DMASS and DCSS together leads to the best SARI score on the TurkCorpus.

Syntactic based machine translation (SBSMT), developed by Xu et al. [21], combines monolingual machine translation with paraphrase database. SBSMT (PPDB+SARI) is also tuned for a specific metric, in this case SARI. According to the survey [1], this approach results in second best SARI score on TurkCorpus.

2.2.2. Annotation tools for NLP

There are a lot of annotation tools for natural language processing. These tools are there to annotate the data set, called corpora, which is fed to the machine learning algorithms. These algorithms train the model, by looking at the input and output,

which, in the end, should make predictions about unseen data [13]. The better the data, which is used for model training, the better the model will be.

One of these tools is Doccano [9]. It provides features like text classification, sequence labeling and sequence to sequence labeling. Doccano is open source and has a MIT licence, which means that it can be used free of charge. The interface is clean and there are shortcuts for faster annotation. To accelerate this process even further, Doccano also supports team collaboration. It is possible to import plaintexts or JSON files. After the data has been annotated, it can be exported to JSON files, which can later be used to train models.

Brat [17] is another tool used for text annotation. It is web based and open source. Brat provides a really simple way of adding and connecting annotations, and these annotations are then visualized in a intuitive way based on the concept of "what you see is what you get". Annotations created can be exported as specific brat format which can then be converted to other formats. Visualizations can be exported as SVG-s, PNG-s, PDF-s or EPS-s.

While there are a lot more tools, which are used for annotation tasks, to the best of my knowledge, I was not able to find one with features, which are shown in this thesis. None of these tools provide a way to annotate two texts from different language levels, do automatic sentence matching and also allow manual intervention when matching these pairs. An honorable mention should go to Newsela[14]. This website allows users to read articles in multiple levels, but these levels are not shown side by side, and it also does not allow users to do any annotation tasks and export data.

[13] https://docs.microsoft.com/en-us/windows/ai/windows-ml/what-is-a-machine-learning-model (Accessed on: 2020-12-14)

[14] https://newsela.com/ (Accessed on: 2020-12-14)

3. Use Cases & Requirements

3.1. Personas

Figure 3.1.: Alfred

Alfred is a language expert. He spent his last 20 years reading books and doing the grammar exercises, which means he can find mistakes in nearly every text. Even in this one. With his special knowledge, he can make easy sentences really hard, by using less common words. He was hired for this project to correct texts that the automatic simplification made. He is really unhappy, because in a few years he might not have his job anymore and that is why he refused to use the computer for the past 20 years. The last computer he used was in the library, to search for a book and even though he did find it, he did not like that he had to use the computer to help him. Now he was hired to fix what the computer created and he is really eager to show that computers can't do his job. This now leads to a problem, because he does not know how to use the computer so this application has to be made user friendly. The UI has to be clutter-free and he would like to just click when he wants to edit the sentence or match sentences from the original and simplified texts.

Alexander is a software developer and tester who will spend his entire day just watching Netflix. In the afternoon the guilt will kick in, and he will go to his computer to write code. This means that he will be looking at the black text on white background during the whole night. This hurts his eyes, so he demanded that dark mode is implemented, otherwise, he will not test the application.

Figure 3.2.: Alexander

Helga is a scientist, who is helping to develop the algorithms for automatic sentence matching. She downloaded the plugin for her favorite editor, which is Atom, that makes JSON files look beautiful. The plugin color codes the matching brackets, formats the JSON file, allows collapsing of the code inside brackets, and a lot more. She fell in love with this plugin and JSON files in general and she insists that the output of the final article with matched sentences will be in JSON format.

Figure 3.3.: Helga

Figure 3.4.: Manfred

Manfred is a programmer who finished his university in the 80s. Back in his days, you had to write the code on paper, go to the university, and then wait in line for the computer to be free. If the programmer made any mistakes on the paper or while he typed his code, he had to leave early and everybody in the ”line” knew that he did something wrong. This ultimately made him a better programmer, and he says that since then he did not make any errors while programming. This experience also made him resent all the new ”kids” who started programming and he thinks they do not know how to program. He is now responsible for running programs on their dedicated servers and does not allow anybody to touch it. Sadly for Manfred, the web application that is presented in this thesis will have to run on his server. Manfred does not want this app to destroy anything and that is why he demands that this app is run in Docker containers so when he inevitably stops liking it, he can just delete it without impacting his system.

Ingrid is a friend of Helga and her assignment is to search for any annotation mistakes that Helga might have missed. Her favorite toy as a child was her markers. Later in life, the only motivation she had for studying, was the thought of highlighting text. To help with this project, Ingrid was promised that she will be able to use virtual highlighters to mark the sentence pairs in her favorite colors.

Figure 3.5.: Ingrid

3.2. Use cases

Import text

Priority: High

Description:
The user should be able to navigate to Import page. There, user could import files either by dragging and dropping files, by typing the filename or using file picker to choose any file on the system.

Fix automatic sentence pairs

Priority: High

Description:
After the file is imported, application will try to construct automatic sentence pairs. The algorithm used is very simple so in some cases it might be necessary to either fix individual sentence pairs or delete all of them, so that the user can start from scratch. User should see changes that are made instantly. In the end the user should be able to save all sentence pairs for an article.

Add sentence pairs

Priority: High

Description:
The user should be able to enter editing mode and click sentences to delete or to match them. User is able to select multiple sentences and the system should show the user which sentences are selected. The user should also be able to clear current selection of sentences. In the end the user should be able to save all sentence pairs for an article.

Export sentence pairs

Priority: High

Description:
The user should be able to export sentence pairs as JSON file. Sentence pairs should also contain some metadata about the pairs.

Edit text content

Priority: Medium

Description:
After the file is imported, it might be needed to fix some mistakes which are caused by faulty file or mistakes in parsing. User should be able to add, edit, merge and remove sentences from an article. After editing, the user should be able to save these changes.

Approve text when imported

Priority: Low

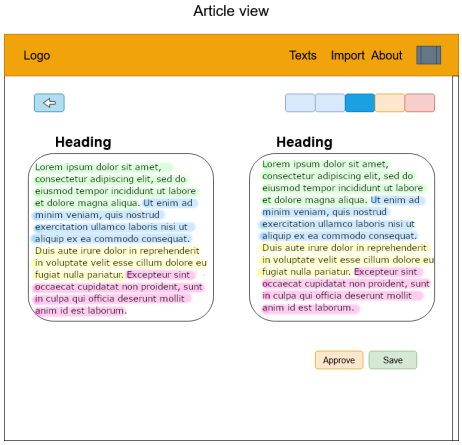
Description:
There should be separate view for articles which are approved. Approved articles are ones which are identical to the articles in file. After the files are imported, articles should be moved to not approved section and then user can, after reviewing the file, approve them and moved them to approved section.

3.3. Functional Requirements

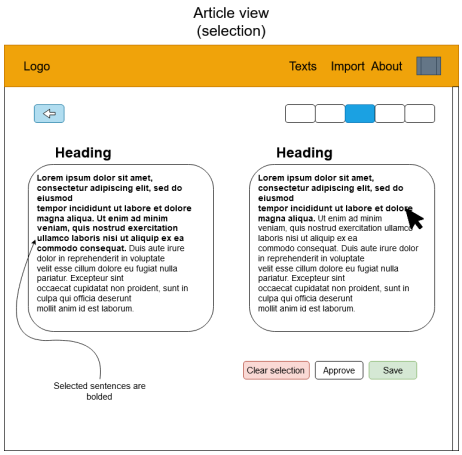
ID	Description	Priority
1	Parse only texts with both B1 and A2	High
2	Allow annotators to correct/approve ML proposed candidate pairs	High
3	Allow annotators to submit manually extracted candidate pairs	High
4	Save annotation results	High
5	Show B1 and A2 texts side-by-side	High
6	Show articles which are already annotated	Medium
7	Dockerize the application	Medium
8	Detect if text is body or title	Medium
9	Dark mode for website	Low
10	Allow users to correct algorithm generated sentence s2	Low
11	Color code the sentences which are paraphrases of each other	Low

3.4. Mockups

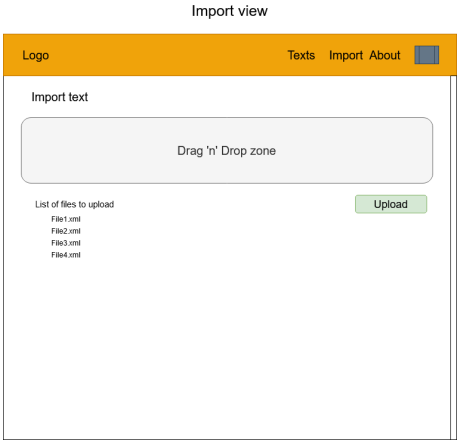
(a) Homepage - showing all articles as cards. Every card has a button which opens the article. There is also a sidebar on the left side, which allows users to filter the articles by text mode.



(b) Article View - with two side by side texts and toolbox in the top right corner. Matcing sentence pairs are highlighted with the same color. Toolbox has buttons which allow users to add and remove sentence pairs, edit text of the article and a button to export current sentence pairs as JSON file.

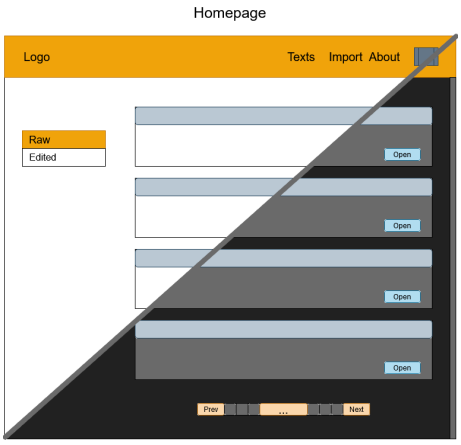


(c) Selection mode - allows users to click sentences to match them. Sentences which selected are bolded. Bottom bar consists of buttons to clear and save current selection.



(d) Import View - provides and interface which allows users to import data. The users can drag and drop files in the specific zone on the page. The filenames, of all files which are to be imported, are displayed below the drag'n'drop zone. There is also a button to upload the files which are listed below the drag'n'drop zone.

(e) Export View - shows sentence pairs for the current article as JSON. It is also possible to save this JSON to a local machine by clicking the button Save To.



(f) Dark mode - inverts the colors of the backgroud and the text to allow for easier reading in dark environments.

4. Method

4.1. Concept

4.1.1. Components

Application described in this thesis consists of three components: back-end (server side), front-end (client side) and the database.

Back-end

(server side) is responsible for communication between front-end and the database. The server receives requests from the client and then fetches the required data from the database. Depending on the type of request, the server can also do some processing of the data received from the database. Another responsibility of server is to parse files that are sent from the client. After the file has been received, back-end reads the file and sends the read data to the database according to the underlying model. Lastly, it is the task of back-end to prepare the data, which will be displayed in JSON file, which will then be exported in front-end.

Front-end

(client side) is responsible for presenting graphical user interface (GUI) elements and the data, which is provided by the back-end. Users are able to navigate the GUI and send requests to the server by interacting with the system. It is possible to consume XML files on the client side and back-end could be removed, which would mean that front-end could communicate with the database directly. The problem with this approach is scalability and performance. In the future, it might

be necessary to increase the number of clients, which are able to access the system and annotate files at the same time, which would lead to a number of problems with database access and data integrity. Furthermore, it is much faster to send files to the dedicated server and process them there.

Database

is a system component where data is stored according to a model. As explained previously, client cannot directly communicate with the database, but must do so via back-end.

4.1.2. XML importing

After the file is received from the front-end, XML DOM parsing is used and the XML file is transformed to a list of nodes, which can be traversed to access all texts from a file. These texts are then fed through a method, which separates them into a list of sentences.

4.1.3. Sentence matching

The next step of the import process is automatically matching the sentences from two different text levels. This is done by calculating Jaccard similarity between all the sentences and matching ones with highest score. It is important to note that this value still has to be larger than some threshold. Last step of the process of XML importing is saving everything to the database.

4.2. Implementation

4.2.1. Database

Database used in this project is PostgreSQL. Since each component of the application is Dockerized, the only thing that is needed for database setup is to configure Docker

to run a PostgreSQL container and tell Spring Boot (programming language used for back-end) how to connect to it (See 4.3 for configuration). All Docker configuration files are in the Appendix B.

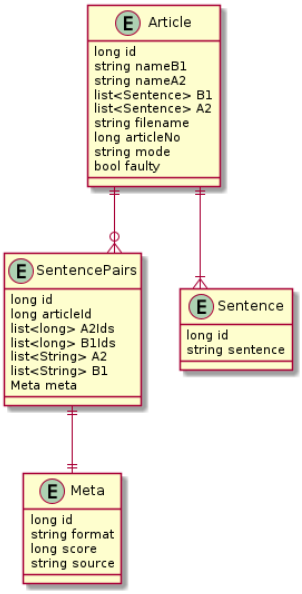


Figure 4.1.: Entity relationship diagram of the database. Main entities are represented in rectangles together with their respective properties. One Article consists of one or more Sentences and also zero or more Sentence Pairs. Every Sentence Pair has exactly one Meta entity.

4.2.2. Back-end

Overview and configuration

The back-end was implemented with Spring Boot. Spring Initializr [1] was used to quicken the setup process. The website provides an easy interface where programmers choose the dependencies needed and download a zip file, which contains all the essential files to get started. Spring Boot dependencies needed for this project are shown in Listing 4.1 and explained below the code snippet.

[1] https://start.spring.io/

```
implementation 'org.springframework.boot:spring-boot-starter-data-
    jpa'
implementation 'org.springframework.boot:spring-boot-starter-web'
implementation 'org.modelmapper:modelmapper:2.3.8'
implementation 'org.springframework.boot:spring-boot-starter-
    actuator'
compile group: 'org.apache.commons', name: 'commons-text', version
    : '1.8'
compileOnly 'org.projectlombok:lombok'
runtimeOnly 'org.postgresql:postgresql'
annotationProcessor 'org.projectlombok:lombok'
testImplementation('org.springframework.boot:spring-boot-starter-
    test') {
  exclude group: 'org.junit.vintage', module: 'junit-vintage-
    engine'
}
```

Listing 4.1: All Spring Boot dependencies from build.gradle

Lambok [2] is a Java library which helps to reduce the number of lines written. Instead of writing getters, setters and constructors for every class, the programmer only needs to annotate a class with flags. Lambok will then plug itself into the build process and generate Java bytecode for every annotation flag used in code.

Apache Commons Text is a library, which provides methods that make text handling easier. In the application described in this thesis, it is used to calculate similarity between sentences from different texts. More information about this process is written in 4.2.2.

Data mapper is a library, which is used to map from one object to another. In this case, it is used to build the objects, which will be sent from back-end to front-end. Front-end might not need all the properties of a class, and this is where data mapper is used to transform full objects to data transfer objects, which will be consumed by the front-end and be displayed to the end user. An example is shown in the Listing 4.2.

```
public class Article {
    private long id;
    private String nameB1;
    private String nameA2;
    private List<Sentence> B1;
```

[2] https://projectlombok.org/

```
    private List<Sentence> A2;
    private String filename;
    private int articleNo;
    private String mode;
    private boolean faulty;
}

public class ArticleNamesDto {
    public long Id;
    public String nameB1;
    public String nameA2;
    public String filename;
    public int articleNo;
}
```

Listing 4.2: Example of the full object and the respective data transfer object (DTO). ArticleNamesDto does not contain all the properties of the Article as they are not needed.

As already mentioned in Section 4.2.1, to be able to communicate with the database, Spring Boot needs to be configured. This configuration is stored in application.properties and is shown in Listing 4.3.

```
spring.jpa.database=POSTGRESQL
spring.datasource.platform=postgres
spring.datasource.url=jdbc:postgresql://database:5432/db
spring.datasource.driver-class-name=org.postgresql.Driver
spring.datasource.username=postgres
spring.datasource.password=postgres
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.
    PostgreSQLDialect
spring.jpa.hibernate.ddl-auto = update
```

Listing 4.3: Spring Boot configuration from application.properties

To actually access the database, except running application with configuration shown above, it is also needed to create repository interfaces, for every class, which represents model entity. Writing actual implementation of these interfaces is not needed, because they are automatically provided by Spring Boot Data JPA.

File importing

XML files are the primary and only source of data for the application described in this thesis. As a result, file importing is one of the most important functional

requirements. After the file has been received in the back-end, number of `javax.xml.parsers` methods are called to parse the XML file and transform it to DOM tree, which can be easily traversed.

One characteristic of the data set provided is, that every file consists of multiple articles and every set of articles for one language level (for example B1) is wrapped in DOC tag. This means that if we check our DOM tree, we can deduce how many levels there are in the file. This is done at the beginning to skip all the files with one or more than two levels, as these articles are either unusable (one level texts) or undesirable (more than two levels). All the files which pass this check, are then fed to the methods, which make articles out of children nodes.

As the data set does not provide any way of knowing which paragraph is the title and which one is the text, this part of the importing process is the most sensitive and usually the main culprit as to why the application fails to parse the given XML file. Importing is done based on assumptions, one being that when looking at the XML file, the title should not be longer than two lines, which, when translated to code, means that if the node has text inside and this text does not contain more than two characters representing end of line, this means that this text should be a title. However this rule does not work all the time because some texts might have a really small last text paragraph, which is not the title. Therefore adjustments were made, mainly that every time two texts are parsed one after the other which are classified as titles, the last one is used as a title and the previous one is appended to the previous article as a last paragraph.

After the first DOC is parsed, meaning the first level is parsed, the children of the second DOC are sent to a different function, which does not create new articles, but adds the sentences from the second text to the fitting articles from the first level. Similar to the first method, this method also works on some assumptions. The assumption here is that every file should have the same number of articles of multiple levels, meaning that the first article from the first level, and the first article from the second level should be about the same topic. However, during the development and testing phases, it was shown that a lot of articles have a mismatched number of paragraphs or articles. The main reason for the mismatched number of paragraphs is that sometimes articles have difficult words and explanations as a separate paragraph usually, but not always, at the end of the article. This is easy to fix as all of these explanations contain the word Erklärung(explanation), so using Regex to find these paragraphs and ignore them solves the problem. On the

other hand, if, after the parsing, it is shown that a number of articles from the first level and from the second level does not match, it is safe to assume that the file either has more or less paragraphs, which do not conform to the format previously described. This file is then rejected and the user is then notified to fix it, or import it manually.

Sentence matching algorithm

After the file has been stored to the database for the first time or after every change to the sentences of the text, the sentence matching algorithm is called. This algorithm is really simple and is used to show the functionality of the front-end but also to show where in the process of importing, new algorithm could be called. Even though it is basic, depending on the text complexity, it did deliver surprisingly good results, but the majority of users of the system will probably want to implement something better. The method, which does the matching first, goes through all the sentences, compares the Jaccard similarity with the sentences from another language level and stores them in a matrix. New sentence pairs are made with pairs of sentences where their similarity score, taken from the matrix, is bigger then some threshold, in this case 0.7. The similarity of the article titles is always calculated and stored as a new pair, so even if texts are completely different, there will be at least one sentence pair.

4.2.3. Front-end

Overview and configuration

Front-end was implemented in React and yarn is the package manager used. For all dependencies See 4.4.

```
"dependencies": {
    "axios": "^0.19.2",
    "bootstrap": "^4.5.2",
    "bootstrap-switch-button-react": "^1.2.0",
    "jquery": "^3.5.1",
    "react": "^16.13.1",
    "react-bootstrap": "^1.3.0",
    "react-dom": "^16.13.1",
```

```
    "react-dropzone": "^11.2.4",
    "react-icons": "^3.10.0",
    "react-paginate": "^6.3.2",
    "react-router-bootstrap": "^0.25.0",
    "react-router-dom": "^5.2.0",
    "react-scripts": "3.4.1",
    "react-toastify": "^6.0.8",
    "styled-components": "^5.1.1",
    "typescript": "^3.9.7"
  },
```

Listing 4.4: React dependencies from package.json

Bootstrap and react-icons are used for styling React components. Dark mode is made possible with stylized-components.

React router is a collection of components, which are used for navigation of the GUI.

Axios is a promise based HTTP client used for communication with back-end.

Importing just the test files leads to more than 900 articles and showing all of them on one page would make an app unusable. This is why pagination is implemented allowing only 5 articles per page to be shown. To make this possible package, react-paginate is used.

Some of the requests made by the front-end expect some kind of feedback. To be able to show the messages from the server in the GUI, package react-toastify is used. This packages shows a small popup with some information, based on the response code and message, to help the user check if the action succeeded or failed. This library is also used to notify users about the current mode they are in when they edit the text (sentence matching mode, sentence pairs delete mode, text editing mode etc.).

React-dropzone is a package used to make drag-'n'-drop zones where files could be dropped and from where they are sent to the back-end to be imported.

Implementation details

React is component based framework and props are used to pass the data from one component to another. Depending on the number of components, which are nested

inside each other, there might be a lot of data forwarding between components, which do not use them. This problem is called prop drilling. Context is used to solve this issue. To quote the official React documentation [3], "Context provides a way to share values like these between components without having to explicitly pass a prop through every level of the tree." The whole application is wrapped with GeneralState. This component has some initial state and it returns <GeneralContext.Provider value={...} > where prop value is this components state. This means that every component can access this global state and subscribe to the changes made to it. In other words, when some value of the global state changes, every component which uses it, will be rendered again. The most important states which are used by most of the components are shown in Listing 4.5.

```
const initialState = {
  mode: 'light',
  loading: false,
  text_mode: 'raw',
  articles: [],
  article: {},
  pairs: [],
  selection: [],
  offset: 0,
  perPage: 5,
  currentPage: 0,
  totalElements: 0,
  textEditing: '',
  unsavedChanges: false,
};
```

Listing 4.5: GeneralState.js contains all states and data which are accessible to the whole application.

GeneralState is also component where methods for communication with back-end are located. As already mentioned in the previous paragraph, axios methods are used to send requests to the server and when the server responds with some data, reducer[4] is used to change the values of the GeneralState. Functions can also be passed as prop value of the <GeneralContext.Provider>, which implies that every component can call these methods as needed and thus change the global state.

[3] https://reactjs.org/docs/context.html#when-to-use-context (Accessed on: 2020-12-06)

[4] https://reactjs.org/docs/hooks-reference.html#usereducer (Accessed on: 2020-12-06)

The front-end application is split into three main parts: Home page, Article View page and Import page.

The Homepage consists of Sidebar, where buttons for choosing text modes are placed, and a list of cards.

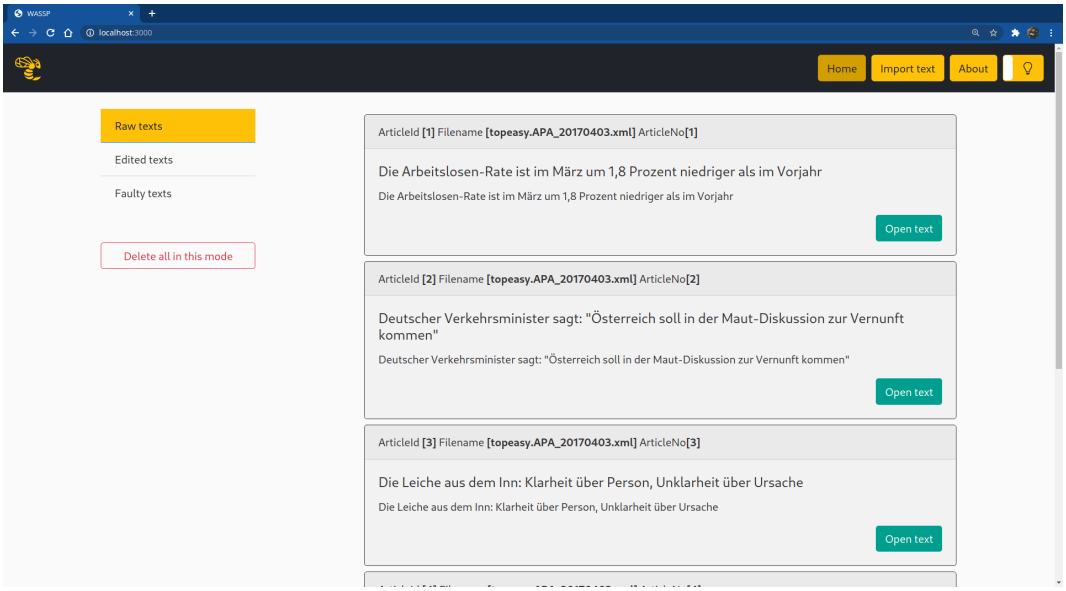


Figure 4.2.: Frontpage showing articles as cards with the sidebar on the left side to filter and delete articles by text mode. The cards contain titles of both language level texts together with some information about the origin of the article, like the filename and article number. The article number represents the location of the article in the file, eg. first article in the file will have a number one.

Text modes can be thought of as a representation of the individual steps of the process from importing files, to the final JSON file with sentence pairs. When articles are imported from files, they are automatically marked as raw. This means that these articles need to be checked by the user to guarantee the correctness of the data. This is an important step, because the data set format might not be consistent throughout the years and there are no definite markers, at least with the data set used to develop this application, which show where one article starts and another one finishes. Most of the data import is done based on the assumptions, which are collected after analyzing the data set. Articles are also considered raw because Regex was used to split sentences of the articles. When the article is checked by

the user and is proven to be correct, the user can then approve it, which marks the text as edited and this text is then hidden from the list of articles which are marked as raw. This means that when the article ”lands” in the edited pile of articles, all the content of the article is correct and sentences of the article are split correctly. Having said that, this text mode does not say anything about the correctness of the sentence pairs made by the automatic sentence matching algorithm. This is still to be verified by language experts, or any other user. Splitting the articles into raw and edited groups enables easier and faster revision process, because users, who are responsible for reviewing the sentence pairs, are guaranteed correct articles, and users, who are responsible for checking the correctness of the articles and article sentences, are not able to accidentally destroy already approved articles. The whole flowchart of the workflow is shown in Figure 4.3.

As already mentioned, next to Sidebar there is a list of cards. The cards represent one article with basic information about the article and the file from where the article was fetched. This is useful because it allows end users to check the imported article against the actual file and fix any mistakes that might happen to be there after importing. Additionally, there is also ArticleNo in the header of the cards, which represents the number of the article in the filename, and can be used to further speed up searching for the correct article in the file. Lastly, cards display titles of both articles and next to them there is a button, which opens the selected article. Clicking this leads to a second component of the application which is Article View page.

Article View page is where both language versions of the article are on display. The titles of individual articles are displayed at the top. Below every title there is a box with respective sentences from the article. Each sentence is displayed in a separate row to help with the readability and usability of the selection tool. Sentences, which the automatic algorithm or the user deemed to be paraphrases of each other, are highlighted in the same color. There are two bars, which provide additional functionality above and below the articles and they are named based on the location with the respect to the articles eg. Top bar and Bottom bar.

Top bar consists of buttons used for going back to article view, entering sentence matching mode, editing text content, exporting the JSON with current sentence pairs and sentence pair deleting mode, which allows users to click the sentence pair to be deleted when in that mode. One of the most important buttons in Top bar is the aforementioned button for entering sentence matching mode. When in

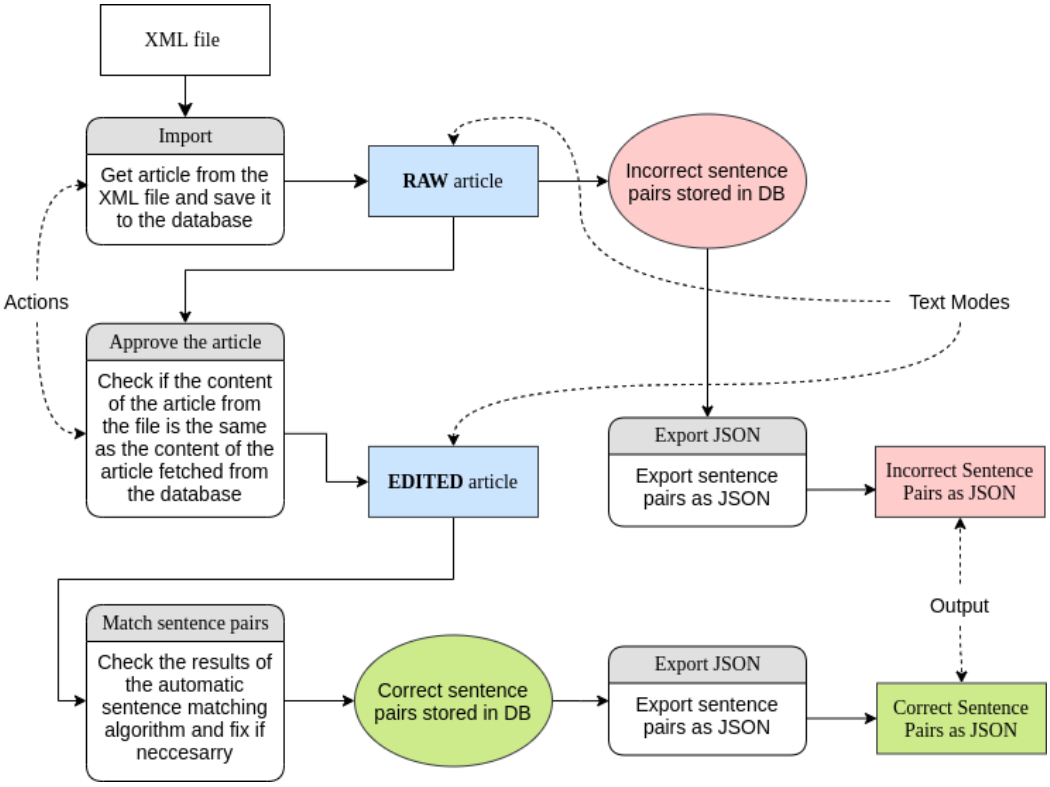


Figure 4.3.: The flowchart showing the full workflow, from the file importing to the exporting of the JSON file with sentence pairs. The rectangles with rounded edges represent actions available to the user. The rectangles with the blue background represent **text modes** of the article. The ovals represent what is stored in the database, while the rectangles with the background, other than blue, represent the output of the data.

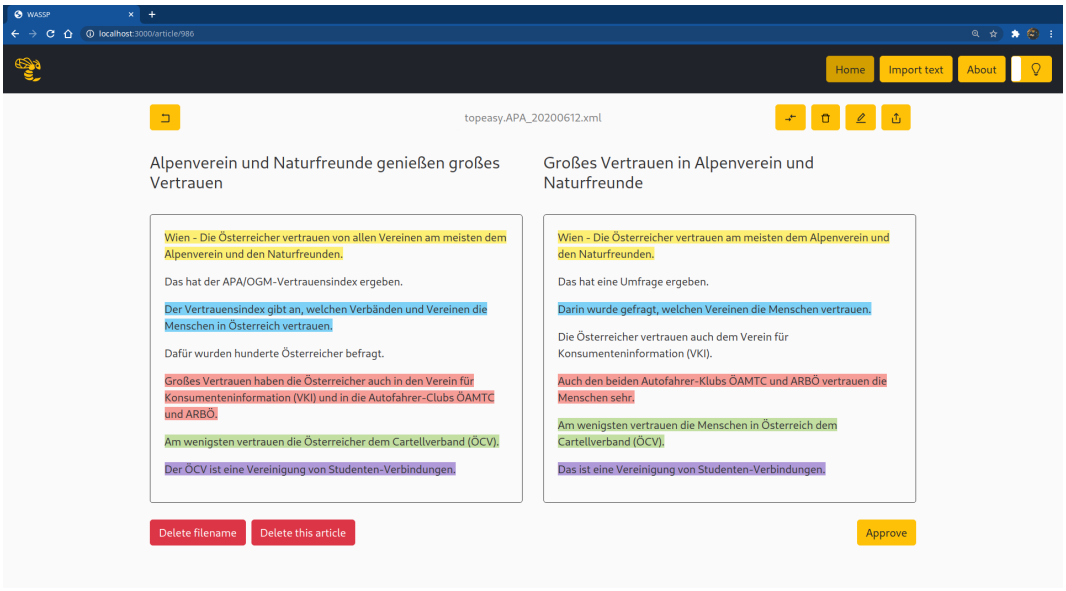


Figure 4.4.: Article View page showing side by side texts from the same article. The matching sentences are highlighted with the same color. The toolbox, on the right side below the navigation bar, contains buttons which, when clicked, enter modes which allow users to add and remove sentence pairs, edit the content of both texts and export current sentence pairs as JSON.

this mode, users are able click the sentences in boxes to match them with their paraphrases from another article. It is only possible to match sentences, which are not already matched, but it is possible to select multiple sentences from both texts. Sentences, which are selected are made bold to provide a visual feedback for the user. Current sentence pairs are compared with last saved sentence pairs to check if any changes are done. When changes are detected, two new buttons will be shown, one, which sends current sentence pairs to be saved to the database and another one, which can clear current changes. Furthermore, when there are unsaved changes, it is only possible to leave the current view by accepting that unsaved changes will be discarded, from the alert window, which will pop up on the screen. Once the user is satisfied with the matched sentences, clicking the button for exporting sentence pairs will open modal window where the JSON file is displayed.

Bottom bar consists of buttons to delete all articles, which have the same filename, delete the displayed article and approve the article, which, as already mentioned, marks it as edited. Out of all the buttons located in the bottom bar, the decision to include a button, which allows the deletion of all articles with the same filename, might seem ridiculous, but there is an explanation as to why this is included. Sometimes it might happen that when importing one file, some articles get paragraphs from another article, which will then offset all articles parsed from that file. Depending on the file it might then be easier to just import articles from that text manually or fix the file in an external editor and import it again. The reason to include this button to Bottom bar is to provide an action which makes sure that all the articles from that file name are deleted from the database so that the file can be imported once again.

The last main component is Import page, where tools for file import are placed and is shown in the figure below.

Import page consists of a drag'n'drop zone where users can just drag files which, after the user clicks submit button, will be sent to the back-end to be imported. It is also possible to click this zone, which opens a file manager where users can select any file on their computer. After the file has been added to the array, the filename will show beneath the zone with the size of the file. Underneath this list there are two buttons, one for clearing this array and another one, which sends them to the server.

Lastly, there is the Dismiss All Notifications button. To understand the functionality of this button it is important to remember that package react-toastify is used to

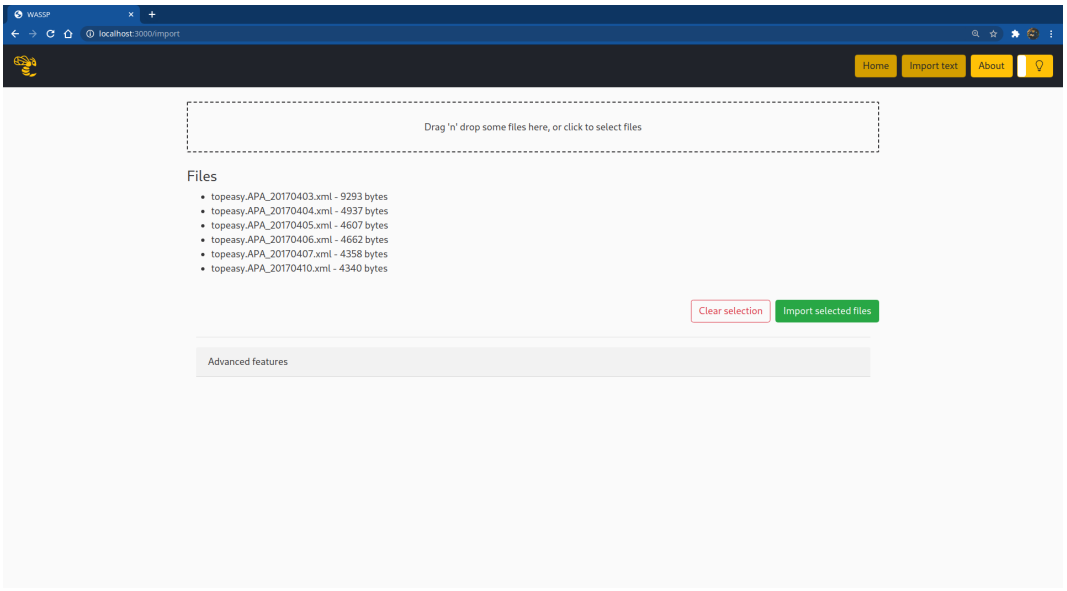


Figure 4.5.: Import page contains a special drag'n'drop zone which allows users to drag the files to this zone to import them. The filenames, of the files which are to be imported, are listed below the drag'n'drop zone.

display small notifications, so called toasts, with some information. These toasts are usually sent with the information whether the file has been successfully parsed or not. If the file is imported successfully, the toast will only show for 2 seconds after which it will be automatically dismissed. If the file was not imported, the notification will not be dismissed automatically to allow users to read the filename and fix it or ignore it.

The React Toast component is placed in the Main.js next to Navbar which means that it will be visible on all pages, even if the page changes. The Toast component is also configured to only show 6 toasts at the same time, and when one toast is dismissed, it will be replaced by the ones, which are were not displayed yet. All of these configurations can make toasts annoying, especially during testing and developing of the application. This is why the button Dismiss All Notifications was implemented. It provides a way to clear all toasts and to clear the waiting queue without restarting the application. It was proved to be really useful during the development process and this is why it also was not removed in the final product. This feature might not be needed for all users, so it was placed in Advanced features

drawer below drag'n'drop zone.

```
<div>
  <ThemeProvider theme={mode === 'light' ? lightTheme : darkTheme
    }>
    <GlobalStyles/>
    <Router>
      <MyNavbar />
      <ToastContainer
        position='bottom-right'
        autoClose={2000}
        hideProgressBar={false}
        newestOnTop={false}
        rtl={false}
        pauseOnFocusLoss
        pauseOnHover
        limit={6}
        draggable={false}
        closeOnClick={false}
      />
      <Switch>
        ...
      </Switch>
    </Router>
  </ThemeProvider>
</div>
```

Listing 4.6: Code snippet from Main.js showing Toast configuration

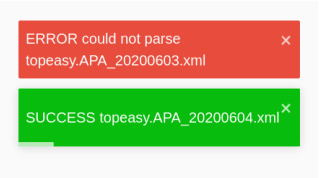


Figure 4.6.: Examples of fail and success toasts showing information about the import status of the files

The last and probably least useful feature of the front-end part of the application described in this thesis is dark mode. Dark mode inverts the colors of text and background, which makes it easier on the eyes in dark environments. It is also a personal preference for some users, mainly programmers, to use everything in

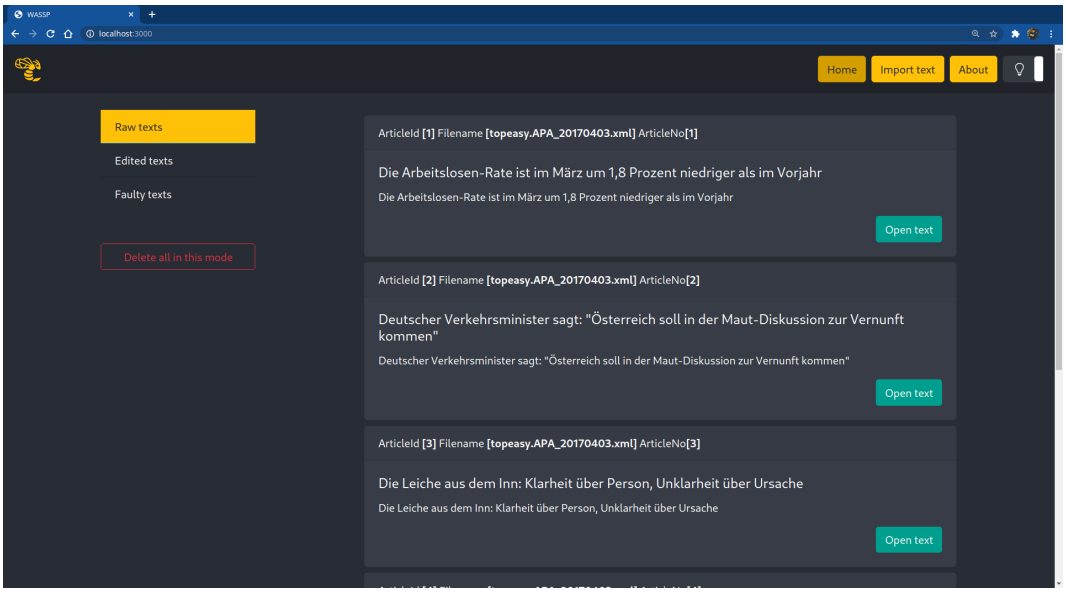


Figure 4.7.: Frontpage in dark mode. Colors of the background and texts are inverted to increase readability in the dark environments.

dark mode. Dark mode is implemented with the help of styled-components library. Wrapping everything in ThemeProvider component allows every component, which is below it, to have an access to the theme. With the help of GlobalStyle component it is possible to toggle theme and doing this will change the colors of GUI elements from light colors to dark ones and vice versa. To actually toggle the theme, a switch is added to the far right side of the navigation bar. The switch also changes the color based on the mode.

4.3. Back-end API routes

This section contains a table with all available routes, appropriate methods and a short description describing what each route does.

Path	Method	Description
/article	GET	Get all articles
/article	POST	Add new article

/article/fault	GET	Get names of faulty articles
/article/names	GET	Get all articles with DTO
/article/names	DELETE	Delete all articles which have specific mode
/article/{id}	GET	Get article with id
/article/{id}/approve	GET	Approve article with id
/article/{id}/filename	GET	Get filename of article with id
/article/{id}/pairdtos	GET	Get sentence pairs DTO of article with id
/article/{id}/pairs	GET	Get sentence pairs of article with id
/article/{id}/pairs	POST	New sentence pairs of article with id
/article/{id}/{level}	GET	Get text of article with id of specific level
/import/apa/all	GET	Import all files from data folder
/import/apa/move	GET	Move all files to data folder from full_dataset
/import/apa/multi	POST	Import multiple files
/import/apa/stat	GET	Get stats of all files from data folder
/import/apa/{filename}	GET	Import file from data folder with filename

5. Evaluation

The evaluation is split into two parts. The first part is an assessment of the system with respect to real numbers and the second part is the user evaluation. Users are first provided with a questionnaire where they needed to answer some questions about themselves and their computer usage habits and experience. Next, the users are given a number of tasks that they need to solve within a recorded time frame. Lastly, the users are given another set of questions to evaluate their experience while using the system. More about the full procedure and about the results of the user evaluation is written in Section 5.3.

5.1. Data set

Data set used to develop and test this application was provided by APA (Austrian Press Agency). The data set consists of 794 XML files containing news articles in German. Out of these files, only 261 of them had only two language levels which could be used as data. Statistics about the number of files and how many levels they have is shown in the table below.

Number of levels	Number of files
One	504
Two	261
Three	4
Four	26
Five	1

Table 5.1.: Statistics about the data set which is used to develop the application described in this work. Number of levels represent a number of language levels of the article.

Even though there are 261 files which satisfy the required number of levels, some of these files are either corrupted or badly formatted, so importing them without any human intervention is not possible. When imported, these 261 files, without any interventions, leads to 986 articles. It is important to note that some of these file have two levels but the content is the same. As a conclusion not all 986 articles are useful because exporting sentence pairs with a similarity score of 100 is not desirable behavior.

5.2. Raw numbers

Importing one file, which has two language levels and has a couple of sentence pairs, takes at most 0.3 seconds, while some articles are imported in less than 0.1 seconds. Spring Boot's stopwatch is used to measure time and this time is measured from the beginning until the end of the controller method called.

Running import on the whole data set, all 794 files, takes about 30 seconds. It is measured from the point when the user clicked the import button, to the point when back-end shows the last log output in the console, meaning that all the files are imported or at least the files that satisfy all requirements. Times for importing one file and all 794 do not match, but that is because for every file, before actually creating articles, sentences and sentence pairs, number of language levels is checked. If the file does not have exactly two levels, it is discarded immediately. This speeds up process of the import significantly.

Memory consumption is measured with the command `docker stat` as all components of the system are run in docker containers. Breakdown of average memory usage is shown in the table below.

Component	Memory usage
Back-end	1.5 GB
Front-end	500 MB
Database	20 MB

Table 5.2.: Memory usage of the individual Docker containers while the application is running.

5.3. User evaluation

The most important evaluation is done by the users. No matter how good or bad the raw numbers are, the users who will use this system will ultimately decide whether they like it and if they would like to use it.

5.3.1. Participants

User evaluation was done using usability testing. Testing was done with 8 users, 5 female and 3 male. All users were students and most of them had a secondary school diploma as their highest level of education. For age distribution see Figure 5.1. For main area of study distribution, see Figure 5.2.

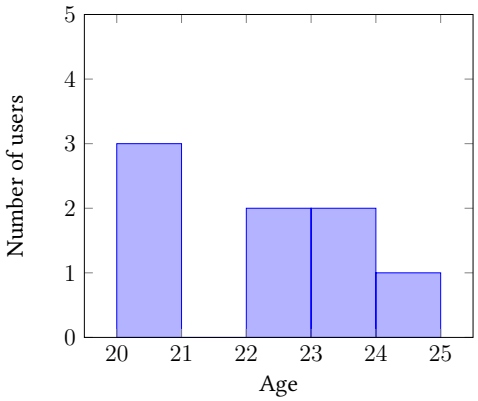


Figure 5.1.: Age distribution of participants who did user evaluation.

Test users spend on average around 6 hours ($\mu = 6.25$, $\sigma = 2.68$) at the computer, and during that time they usually watch Netflix, browse the Internet, or use the computer for studying. Lastly, none of the users had any previous experience with any annotation tools and were not familiar with anything related to text simplification or other NLP tasks.

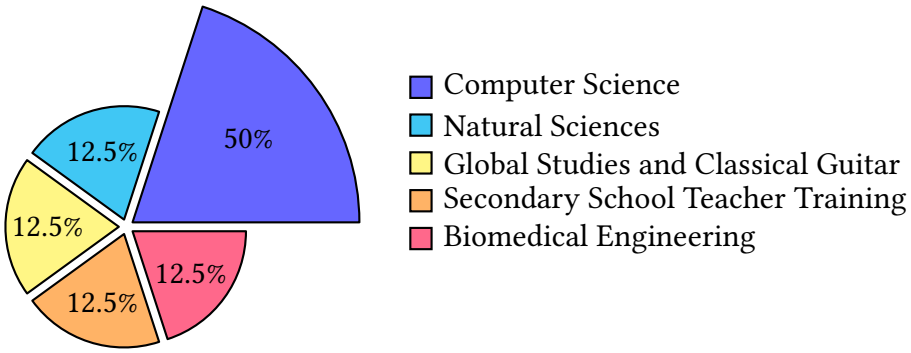


Figure 5.2.: Main fields of study of participants who did user evaluation.

5.3.2. Methodology

Users were first given a quick questionnaire with some basic questions about themselves and their computer usage. The last question is important because it is used to test the assumption that users who spend a lot of time in front of the computer usually get familiar with software easier. It is also important to ask this question to see if UI is user friendly and intuitive, even for people who do not spend a lot of time in front of computers, and who might not have seen a lot of similar design patterns.

After the users answer the first part of the questionnaire, they are introduced with the main concepts of the application they are going to test. They are also introduced to sentence simplification problems and how this application helps to solve the problem of gathering enough data, in this case sentence pairs. Furthermore, the basic workflow is explained; from file importing, approving the article after the contents of the file have been crosschecked with the sentences which are parsed and displayed in the application, and finally exporting the sentence pairs as JSON file. They are also shown a diagram with information about the text modes (raw and edited). Lastly, they were informed that it is not them who are getting tested, but the system they are using, and the more feedback they provide, the better it is, as this can later be used to improve the system.

Before providing the users with the list of tasks they needed to do, they were allowed to navigate the system and get to know all the functions that it provides. They were encouraged to try everything and they were also told that some buttons

in the GUI have a tool tip when hovered over, so it might be a good idea to try to read all of them before starting the testing process.

The main part of the usability testing is the thinking aloud test. In this step users are given tasks, one by one, and asked to perform the actions they think are necessary to solve the task. They are encouraged to talk about their thinking process while navigating the system. Users are only provided help, if the time limit is breached. This limit is set individually for every task. Time is measured and results are shown in 5.3.3. There were 10 tasks, ranging from simple navigation, to the ones representing full workflow; from file import to sentence matching and lastly sentence exporting. For the full list of tasks, See Appendix C.

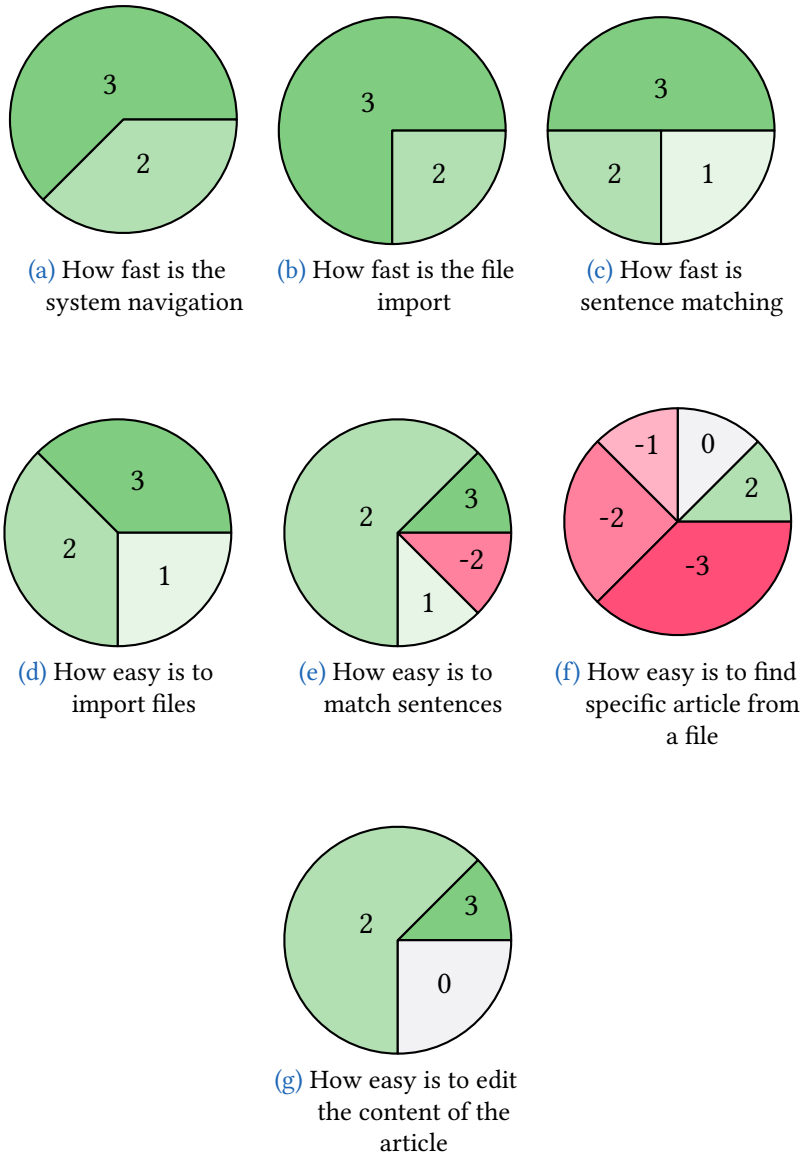
After a user finishes all the tasks, they are given a second part of the questionnaire were they are asked to write their general opinions, together with the positive and negative aspects of the application. The last section of the questionnaire consists of questions about the systems´ usability and performance together with the 7 point Likert scale. For the full questionnaire See Appendix D.

Lastly, users were asked to redo the task which represents the full workflow, in this case the task number 10, but this time they were given some hints and shortcuts which could help them finish the task faster.

5.3.3. Results

Results from the questionnaire and users testing are written in this section.

Users´ answers to the 7-point Likert scale are visualized below. Numbers inside of the individual pie chart elements represent the number which users choose, on a scale from -3 to 3. Participants were asked a series of questions relating to the performance, effectiveness and difficulty level of the application described in this thesis. For performance questions, -3 on a scale means slow, while 3 means fast and for ease of use questions, -3 means really hard to use, while 3 means quite easy to use. The size of the pie chart elements also plays a big role, because the bigger the element, the more users chose that number on the scale. Colors and their shades are also used to make the readability of the graph better, green was used for the positive answers, whereas red was used for negative. The darker the color, the more it represented each extreme.



(a) How fast is the system navigation

(b) How fast is the file import

(c) How fast is sentence matching

(d) How easy is to import files

(e) How easy is to match sentences

(f) How easy is to find specific article from a file

(g) How easy is to edit the content of the article

Figure 5.3.: Chart showing the answers of the participants to the questions about the application they tested. The questions asked are displayed below the individual charts. The participants answered these questions on a 7-point Likert scale.

From the charts it is easy to see that most of the users thought the performance of the whole system is good, in other words opening pages, clicking buttons, going

back, all felt really fast and smooth ($\mu = 2.625$, $\sigma = 0.517$). Participants also felt that importing of the files is immediate with a highly positive response ($\mu = 2.75$, $\sigma = 0.462$). Moreover, all of the participants also regarded the import feature as positive and had little difficulty navigating it ($\mu = 2.125$, $\sigma = 0.834$). Although the speed of sentence matching was graded positively ($\mu = 2$, $\sigma = 0.755$), the easiness of use was not received so well ($\mu = 1.5$, $\sigma = 1.511$). The user who gave the negative rating stated that the system feature was unnecessarily confusing. The lack of any search functionality was not received very well, especially with tasks which required users to first search specific articles from a file. Participants graded article searching as really bad ($\mu = -1.5$, $\sigma = 1.77$). It is important to notice that users which gave a neutral or positive grade used built in search functionality of the browsers, but they also expressed that dedicated search bar would be much more useful. Lastly participants rated easiness of use of tools, which help with editing the content of the articles, as mostly positive ($\mu = 1.625$, $\sigma = 1.060$).

Table 5.3 shows the time it took for a participant to solve a task. Additionally, there are also two more columns; one showing the average time it took to solve the task and one showing the time limit set for each task. The time limit is used to prevent the test from taking too long, and if users breach them, the task is marked as failed and this means that the functionality, which is tested with that task, should be looked at more carefully, to try to make it better. It is worth mentioning that not a single participant exceeded the time limit.

When asked about the positive aspects of the application, participants stated that they liked the responsiveness of the system and the design. Test users also liked how easy the system is to use, since they were not provided with any tutorial, just with custom tooltips.

When asked about the negatives, almost all participants mentioned the lack of search functionality. Furthermore, the application described in this thesis had two separate modes, one for creating sentence pairs and one for deleting them, and some users thought that this is confusing and that it should be possible to do these tasks in one mode. Lastly, a couple of participants mentioned that the color scheme should be different for some buttons, mainly the buttons in the toolbox in Article View.

	P1	P2	P3	P4	P5	P6	P7	P8	Avg.	TL
T1	10s	3s	45s	31s	3s	5s	5s	10s	14s	2m
T2	14s	25s	11s	9s	30s	20s	15s	25s	19s	2m
T3	3m	4m 10s	2m 47s	4m 40s	2m 36s	3m 7s	4m 10s	2m 15s	3m 20s	6m
T4	1m	10s	40s	6s	5s	5s	24s	5s	19s	2m
T5	25s	50s	1m 6s	24s	1m 5s	18s	45s	30s	40s	4m
T6	20s	40s	38s	30s	17s	5s	37s	5s	24s	2m
T7	5s	35s	1m 40s	57 s	10s	25s	1m 4s	15s	39s	2m
T8	1m 15s	45s	1m 40s	57s	1m 21s	1m	1m 14s	30s	1m 25s	4m
T9	10s	5s	10s	5s	11s	4s	15s	5s	8s	2m
T10	1m 35s	1m 22s	1m 40s	1m 12s	1m 21s	1m 9s	2m	1m	1m 24s	6m
Sum	8m 14s	9m 5s	11m 17s	9m 31s	7m 39s	6m 38s	10m 49s	5m 20s	8m 52s	32m

P - participant
T - task
Avg. - average time it took to solve the task
TL - time limit set for a task

Table 5.3.: The table showing the time it took for each participant to complete the individual tasks, together with time limit which was set for every task.

5.4. Discussion

Focus of the application shown in this thesis is user friendliness for novice users and performance of the system as a whole.

The former is important because language experts, who will be employed to supervise automatic sentence matching algorithm and fix the mistakes, might not be familiar with complex software. As already mentioned, there were 8 participants, half of whom were computer science students, representing expert users due to their experience with using a lot of different software. This experience might help them to use the system in a more efficient way and also it might help them to understand steps in the tasks better. The other half of the users, non computer science students, represent the novice users. Results shown in the 5.3 show that both groups took the same amount of time to finish the provided tasks, meaning that even users not usually surrounded by similar systems, managed to do the tasks with the same efficiency as the expert users. As a result, it could be said that the application has a low learning curve. Further indicator that the application is novice friendly is that not a single user exceeded the time limit set for the tasks.

Rating the performance of the application can be done by looking at the raw numbers shown in section 5.2, or by looking at times measured in table 5.3 together with the user scores presented as graphs in 5.3.3. It can be concluded that the participants felt the application is very fast and this can be confirmed with the raw numbers.

The most frequently brought up system flaw is the lack of the search bar, which was not a specified requirement of the program. In regards to that, the user experience was probably worsened due to the fact that pagination was implemented with only five articles per page. In my opinion the user experience would not be as bad if there were more articles per page. The removal of the pagination could be a potential solution but we would have to tread lightly with that option, since it might lead to performance issues. In connection to the search issues, almost all users overlooked the fact that the article number and file name were displayed next to the title of the article, which could help them search for texts faster and make the whole search experience more enjoyable.

Lastly, the users said that they had little to no issues with manipulating the editing feature of the system. As can be seen in the provided graph, users regarded this

feature as mostly positive, with two neutral rankings. Although there is no objective metrics for measuring the correctness of data which is exported, providing users with the option to edit the content of the articles themselves and being able to clearly see, in real time, the selected sentence pairs should be enough to prove that the correctness of the output data is dependent more on the users than the system.

6. Conclusions

The goal of this thesis was to develop the web app for easy visualization, effortless editing and exporting of the annotated data, without developing any new state-of-the-art algorithms, which could solve difficult simplification problem. The language experts who will be using this application do not necessarily have to be versed in any programming language or to be computer experts of any sort. In regards to that, one of the biggest concerns of the application, aside from its functionality, was the user friendliness.

All of the set requirements, written in Chapter 3, have been successfully and fully met, derived from the user testing. The acquired results, written in Chapter 5, adequately prove its functionality and simplicity, due to the fact all of the users successfully completed the tasks, without exceeding the time limit. In addition to the previous point, users had no official training, only a five minutes time frame where they could explore the application on their own accord.

Specially emphasized were performance and correctness of the sentence pairs. These were the most highly praised aspects of the application, which was described in the 5.3. Although users gave some constructive criticism on the existing application, none of it was related to its core functionalities and was more inclined to help the system to be better tailored to the users´ specific expectations.

The data, after parsing the files and checking whether sentence matching is correct, acquired with this application is a step in the right direction on acquiring even more useful data, which can be used to help machine learning algorithms train the model. The more data processed, the better the model will be, ultimately improving automatic sentence matching and overall text simplification in the future.

Lastly, this application could prove beneficial not only for language experts in their respective field, but also flatten the learning curve for new language learners by providing them with a side by side view of the same text written in different difficulty levels.

Appendix

Appendix A.

Milestones

A.1. Timeline

Timeline is presented with a GANTT[1] chart, please see A.1. For better readability, all milestones are also written in the next section A.2.

Figure A.1.: GANTT chart

[1] https://en.wikipedia.org/wiki/Ganttchart

A.2. Listed

- Exposé finished
  - Topic of thesis finalized and agreed with supervisor
  - Draft version of the chapters:
    - Introduction chapter
    - Background chapter
    - Use cases & Requirements chapter
  - Proof of concept for the back end and the parsing of the data
- Literature review finished
  - Final version of the chapters:
    - Introduction chapter
    - Related Work chapter
    - Use cases & Requirements chapter
  - Proof of concept for the front end
- Evaluation pipeline established
  - Draft version of the chapters:
    - Method chapter
    - Evaluation chapter
    - Conclusions chapter
  - Usability testing
- Finalizing the thesis
  - Final versions of all chapters
  - Bug fixing and testing of the software
  - Finalizing the application

Appendix B.

Docker setup

```
version: '3.2'
services:
  app:
    container_name: backend
    image: backend
    build: ./backend
    ports:
      - '8080:8080'
    volumes:
      - type: bind
    source: ./data
    target: /data
    depends_on:
      - database
  database:
    container_name: database
    image: postgres
    ports:
      - '5432:5432'
    environment:
      - POSTGRES_PASSWORD=postgres
      - POSTGRES_USER=postgres
      - POSTGRES_DB=db
  frontend:
    container_name: frontend
    build: ./frontend
    restart: always
    ports:
      - '3000:3000'
    # To prevent the react from dying at the start
```

```
    stdin_open: true
    depends_on:
      - app
```

Listing B.1: docker-compose.yml

```
FROM openjdk:8-jdk-alpine
VOLUME /tmp
EXPOSE 8080
ADD build/libs/bachelor_app-0.1.jar app.jar
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-
  jar","/app.jar"]
```

Listing B.2: Dockerfile (back-end)

```
FROM node:13.12.0-alpine
WORKDIR /app
COPY package.json /app
COPY yarn.lock /app
RUN yarn install
COPY . /app
CMD yarn dev_start
EXPOSE 8000
```

Listing B.3: Dockerfile (front-end)

Appendix C.

Tasks

Tasks

1. Find all texts which are approved
2. Import file topeasy.APA_20200309.xml located on Desktop
3. Find article with title “Untersuchungs-Ausschuss zu "Ibiza-Video" gestartet” and approve it after checking the content of the file is correct (XML file was already open in another text editor)
4. Open any article which filename is topeasy.APA_20200604.xml and export current sentence pairs as JSON
5. Open article number 2 from articles where filename is topeasy.APA_20200604.xml, and match “Ohne den neuen Kollektiv-Vertrag wollte Ryanair den Laudamotion-Stützpunkt in Wien zusperren.” from B1 to “Ryanair will, dass die Mitarbeiter von Laudamotion viel weniger Geld verdienen als bisher.” and “Sonst sperrt Ryanair den Standort von Laudamotion in Wien zu.” Lastly save changes
6. Delete all articles where filename is topeasy.APA_20200615.xml
7. Delete article about Wiener Tiergarten
8. Open first article from topeasy.APA_20200309.xml , delete all sentence pairs and save changes
9. Delete all “raw” articles
10. Find article about Wiener Tiergarten (filename is topeasy.APA_20200604.xml), fix the mistake where first sentence in both language levels is not split correcty, fix mistake by automatic sentence matching algorithm (only first sentences should be matched, not first from B1 and first and second from A2), save changes and call export current sentence pairs

Appendix D.

Questionnaire

Questionnaire

Part 1

General questions

1. Name

2. Age

3. Sex

4. Occupation

Education

1. Highest level of education

2. If student write your main area of study

Computer use

1. How many hours do You usually spend at the computer?

2. What do You usually do at the computer?

3. Are You familiar with any annotation tasks?

4. Are You familiar with any annotation tools?

Part 2

System evaluation

Overall impressions

Positive aspects

Negative aspects

How fast is the system navigation

Really slow			Neutral			Really fast
-3	-2	-1	0	1	2	3

How fast is the importing of file

Really slow			Neutral			Really fast
-3	-2	-1	0	1	2	3

How fast is sentence matching

Really slow			Neutral			Really fast
-3	-2	-1	0	1	2	3

How easy is to import file

Really hard			Neutral			Really hard
-3	-2	-1	0	1	2	3

How easy to match sentences

Really hard			Neutral			Really hard
-3	-2	-1	0	1	2	3

How easy to find specific article from a file

Really hard			Neutral			Really hard
-3	-2	-1	0	1	2	3

How easy is to edit the article

Really hard			Neutral			Really hard
-3	-2	-1	0	1	2	3

Bibliography

[1] Fernando Alva-Manchego, Carolina Scarton and Lucia Specia. 'Data-Driven Sentence Simplification: Survey and Benchmark'. In: *Computational Linguistics* 46.1 (2nd Jan. 2020). Publisher: MIT Press, pp. 135–187. ISSN: 0891-2017. DOI: 10.1162/coli_a_00370. URL: https://doi.org/10.1162/coli_a_00370 (visited on 02/01/2021) (cit. on pp. 10, 11).

[2] Arnaldo Candido et al. 'Supporting the Adaptation of Texts for Poor Literacy Readers: a Text Simplification Editor for Brazilian Portuguese'. In: *Proceedings of the Fourth Workshop on Innovative Use of NLP for Building Educational Applications*. Boulder, Colorado: Association for Computational Linguistics, June 2009, pp. 34–42. URL: https://www.aclweb.org/anthology/W09-2105 (visited on 30/07/2020) (cit. on p. 1).

[3] Yixin Cao et al. 'Expertise Style Transfer: A New Task Towards Better Communication between Experts and Laymen'. In: *arXiv:2005.00701 [cs]* (May 2020). arXiv: 2005.00701. URL: http://arxiv.org/abs/2005.00701 (visited on 03/07/2020) (cit. on p. 1).

[4] John Carroll et al. 'Practical Simplification of English Newspaper Text to Assist Aphasic Readers'. In: *In Proc. of AAAI-98 Workshop on Integrating Artificial Intelligence and Assistive Technology*. 1998, pp. 7–10 (cit. on p. 1).

[5] R. Chandrasekar, Christine Doran and B. Srinivas. 'Motivations and Methods for Text Simplification'. In: *COLING 1996 Volume 2: The 16th International Conference on Computational Linguistics*. 1996. URL: https://www.aclweb.org/anthology/C96-2183 (visited on 29/07/2020) (cit. on p. 1).

[6] Jan De Belder and Marie-Francine Moens. 'Text simplification for children'. In: (Jan. 2010) (cit. on p. 1).

[7] Richard Evans, Constantin Orasan and Iustin Dornescu. 'An evaluation of syntactic simplification rules for people with autism'. en. In: (2014). Accepted: 2017-11-21T13:49:11Z Publisher: Association for Computational Linguistics. DOI: 10.3115/v1/W14-1215. URL: https://wlv.openrepository.com/handle/2436/620875 (visited on 28/07/2020) (cit. on p. 1).

[8] Juri Ganitkevitch, Benjamin Van Durme and Chris Callison-Burch. 'PPDB: The Paraphrase Database'. In: *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. NAACL-HLT 2013. Atlanta, Georgia: Association for Computational Linguistics, June 2013, pp. 758–764. URL: https://www.aclweb.org/anthology/N13-1092 (visited on 15/01/2021) (cit. on p. 11).

[9] Hiroki Nakayama et al. *doccano: Text Annotation Tool for Human*. Software available from https://github.com/doccano/doccano. 2018. URL: https://github.com/doccano/doccano (cit. on p. 12).

[10] Shashi Narayan and Claire Gardent. 'Hybrid Simplification using Deep Semantics and Machine Translation'. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Baltimore, Maryland: Association for Computational Linguistics, 2014, pp. 435–445. DOI: 10.3115/v1/P14-1041. URL: http://aclweb.org/anthology/P14-1041 (visited on 03/01/2021) (cit. on p. 10).

[11] Kishore Papineni et al. 'Bleu: a Method for Automatic Evaluation of Machine Translation'. In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, July 2002, pp. 311–318. DOI: 10.3115/1073083.1073135. URL: https://www.aclweb.org/anthology/P02-1040 (cit. on p. 9).

[12] Ellie Pavlick and Chris Callison-Burch. 'Simple PPDB: A Paraphrase Database for Simplification'. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. ACL 2016. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 143–148.

DOI: 10.18653/v1/P16-2024. URL: https://www.aclweb.org/anthology/P16-2024 (visited on 15/01/2021) (cit. on p. 11).

[13] Luz Rello et al. 'Frequent Words Improve Readability and Short Words Improve Understandability for People with Dyslexia'. en. In: *Human-Computer Interaction – INTERACT 2013*. Ed. by Paula Kotzé et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2013, pp. 203–219. ISBN: 978-3-642-40498-6. DOI: 10.1007/978-3-642-40498-615 (cit. on p. 1).

[14] Matthew Shardlow. 'A Survey of Automated Text Simplification'. In: *International Journal of Advanced Computer Science and Applications* 4 (1st Jan. 2014). DOI: 10.14569/SpecialIssue.2014.040109 (cit. on p. 1).

[15] Advaith Siddharthan. 'A survey of research on text simplification'. In: (2014). DOI: 10.1075/ITL.165.2.06SID (cit. on p. 1).

[16] Advaith Siddharthan. 'Syntactic Simplification and Text Cohesion'. en. In: *Research on Language and Computation* 4.1 (May 2006), pp. 77–109. ISSN: 1570-7075, 1572-8706. DOI: 10.1007/s11168-006-9011-1. URL: http://link.springer.com/10.1007/s11168-006-9011-1 (visited on 28/07/2020) (cit. on p. 1).

[17] Pontus Stenetorp et al. *BRAT: a web-based tool for NLP-assisted text annotation*. 2012. URL: http://brat.nlplab.org/index.html (cit. on p. 12).

[18] Willian Massami Watanabe et al. 'Facilita: reading assistance for low-literacy readers'. In: *Proceedings of the 27th ACM international conference on Design of communication*. SIGDOC '09. Bloomington, Indiana, USA: Association for Computing Machinery, 5th Oct. 2009, pp. 29–36. ISBN: 978-1-60558-559-8. DOI: 10.1145/1621995.1622002. URL: https://doi.org/10.1145/1621995.1622002 (visited on 29/07/2020) (cit. on p. 1).

[19] Sander Wubben, Antal van den Bosch and Emiel Krahmer. 'Sentence Simplification by Monolingual Machine Translation'. In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. ACL 2012. Jeju Island, Korea: Association for Computational Linguistics, July 2012, pp. 1015–1024. URL: https://www.aclweb.org/anthology/P12-1107 (visited on 03/01/2021) (cit. on p. 10).

[20] Wei Xu, Chris Callison-Burch and Courtney Napoles. 'Problems in Current Text Simplification Research: New Data Can Help'. In: *Transactions of the Association for Computational Linguistics* 3 (1st Dec. 2015). Publisher: MIT Press, pp. 283–297. DOI: 10.1162/tacla00139. URL: https://doi.org/10.1162/tacla00139 (visited on 23/07/2020) (cit. on p. 9).

[21] Wei Xu et al. 'Optimizing Statistical Machine Translation for Text Simplification'. In: *Transactions of the Association for Computational Linguistics* 4 (2016), pp. 401–415. DOI: 10.1162/tacl a 00107. URL: https://www.aclweb.org/anthology/Q16-1029 (cit. on pp. 9, 11).

[22] Sanqiang Zhao et al. 'Integrating Transformer and Paraphrase Rules for Sentence Simplification'. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. EMNLP 2018. Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, pp. 3164–3173. DOI: 10.18653/v1/D18-1355. URL: https://www.aclweb.org/anthology/D18-1355 (visited on 03/01/2021) (cit. on p. 11).

[23] Zhemin Zhu, Delphine Bernhard and Iryna Gurevych. 'A Monolingual Tree-based Translation Model for Sentence Simplification'. In: *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*. COLING 2010. Beijing, China: Coling 2010 Organizing Committee, Aug. 2010, pp. 1353–1361. URL: https://www.aclweb.org/anthology/C10-1152 (visited on 15/01/2021) (cit. on p. 9).