KING'S College LONDON

University of London

6CCS3PRJ Final Year
Automated Timeline Extraction

Final Project Report

Author: Oliver Philip Höhn

Supervisor: Dr Jeroen Keppens

Student ID: 1426248

April 7, 2017

Abstract

The abstract is a very brief summary of the report's contents. It should be about half-a-page long. Somebody unfamiliar with your project should have a good idea of what your work is about by reading the abstract alone. -Summary of Project When legal and related professionals examine a case, they receive a substantial number of documents. These documents need to be examined in a useful manner to understand the events occurred. One useful perspective to understand what happened is a timeline of events. However, reading a large collection of documents and producing a timeline can be cumbersome. The aim is to ease this task by producing a system that shows the events depicted in the documents through a timeline.

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary. I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words.

Oliver Philip Höhn

April 7, 2017

Acknowledgements

It is usual to thank those individuals who have provided particularly useful assistance, technical or otherwise, during
your project. Your supervisor will obviously be pleased to be acknowledged as he or she will have invested quite a lot of time overseeing your progress.
-Acknowledge Supervisor, Friends & Family I would like to thank my supervisor, Dr. Jeroen Keppens. The supervision and support he provided was extremely helpful throughout the progression of the project. Also I would like to thank my family and friends for the continued support and encouragement throughout the project. I could have not done it without them.

Contents

Chapter 1

Introduction

This is one of the most important components of the report. It should begin with a clear statement of what the project is about so that the nature and scope of the project can be understood by a lay reader. It should summarise everything that you set out to achieve, provide a clear summary of the project's background and relevance to other work, and give pointers to the remaining sections of the report, which will contain the bulk of the technical material. -What is Project About (incl scope)? What is Aim? Bakground? Relevance to other works? (Pointers to other sections?)

The project aims to facilitate the understandment of a substantial number of documents (espeically law related) through a graphical representation. When a law professional is tasked with a law case, they are expected to fully understand the overall structure and occurrence of the events described in the documents. However, when a large collection of documents are involved, this task can be both cumbersome for the employees and financially expensive for the employer. As the documents tend to be extensive and complex, requiring many working hours to understand the underlying story, thus not allowing the professional to progress to the next stage of their work.

1.1 Project Scope

As the system requires analyzing, processing and graphically representing the information in the documents, the main areas of the project are Natural Language Processing (NLP), Data Processing and Data Representation. There relevance to other works will be discussed in the Background Chapter.

1.2 Objectives

//what is an event (summary, size, subjects, date, etc)

The Objectives are to produce a system that is effective, efficient and simple to use. The system should require as input a selection of documents written in correct English (i.e. in natural language) and produce a timeline with the events described in the documents.

Since the user does not have to be computer scientists or have any technological knowledge, but rather are expected to be law professionals, the system should be approachable for them. Therefore the visibility of what the system is doing, and what the users options are is important.

The timeline should be self-expanatory. The user should understands which events happened during which time periods, and know from the information provided what occurred during an event.

The system should produce responses in an appropriate time based on the input, and allow the user to rectify where the system has made mistakes.

The system should allow users to change the input settings, such as the length of the summary, reference points or how many processors can the program use in parallel (to limit/improve the performance), to allow the user to see how the timeline changes with different input parameters.

The user may consider the need to save the timeline for later use, or further analyzation. Therefore, providing tools to save the data of the timeline as a PDF or as a JSON in the user's system would benefit them. Producing a JSON output would allow the system to be used by 3rd-parties, that may provide their own graphical user interface (UI) using the system to only process the documents; or further process the events identified by the system.

1.3 Report Structure

Following this chapter is a discussion of the background related to this project. This is then followed by a formal presentation of the requirements and specification of the project. Which is then followed by the design architecture and patterns used in the system. Followed by the implementation, testing and evaluation of the system. During the evaluation, it will be discussed to what extend the requirements have been met. In the final chapter, the project will be concluded and improvements for future work will be discussed.

Chapter 2

Background

The background should set the project into context by motivating the subject matter and relating it to existing published work. The background will include a critical evaluation of the existing literature in the area in which your project work is based and should lead the reader to understand how your work is motivated by and related to existing work.

//explain what an event (from what is it built off) //defintion of an event

The resulting system should aim to produce a timeline of events based on the input text. An event is given by its date(s), subjects and a short summary of the text that produced it.

For this system, the text of documents will be split into their sentences, which are analyzed separately. An event is produced when a given sentence contains a date (or temporal expression). Events may contain a range of dates. A range is given by a start and end date. The range describes that the event occurred during that time period. For example, an event that occurred in the 1980s would have two dates, one for the start date: 1980-01-01, and one for the end date: 1989-12-31. While an event that happened on one specific day would only have one date associated with it.

The subjects of an event are given by the "person, place, thing, or idea that is doing or being something"[1]. These are key words that help convey the meaning of an event. They aid the summary of the event, i.e. summary of the sentence that was identified as this event, by providing additional information used to understand what happened.

2.1 Natural Language Processing

-explain what NLP is

The projects primarily enters the field of Natural Language Processing (NLP). NLP is the field in Computer Science that focuses on producing systems that understand human readable and spoken language (cite). While there is ground breaking research in the field, many of the systems require the human input to be subject to constraints. As many of the systems use rule sets that are applied to the input to translate it to a useful input for the system. For example, removing ambiguities in text.

For this system it is expected that the documents are written in correct English. This is because every language has their own grammar. The grammar can be used as a rule set for the language. Thereby using them to identify key words such as people, locations, and companies; and also using them to produce summaries in some case [4]. For example, it is to be expected that documents are written in correct English. To expand the system to different languages would require different rule sets to be applied depending on the language to process the text. Different languages would require different algorithms, models and rule sets used.

An issue that may occur in the input documents is that they are grammatically disorganised and incorrect. Many NLP software tools (incluiding StanfordCoreNLP which is used in this project) perform extremely poorly with such input. One example is the performance of NLP tools with Tweets, short sentences used in the social media platform Twitter [1], that use informal abbrevations due to a character limit. In the paper Named Entity Recognition in Tweets: An Experimental Study [8], they looked at the performance of popular NLP tools on "Tweets". Many of the different NLP tools such as Apache OpenNLP and Stanford CoreNLP performed poorly. This is due to the NLP tools and their algorithms not being able to apply their rules on to the text to identify its different components. Hence, for this project it will be assumed that the input documents will be written in correct English. This is to be expected as the primary use of this tool is for formal documents, such as law documents. In the future this can be further expanded to other languages. -what parts of NLP are involved

NLP is a broad area of study. For this project the focus is on Automatic Summarization, Named-Entity Recognition (NER), and Sentence Breaking.

2.1.1 Automatic Summarization

In Automatic Summarization the aim is to produce a shorter version (a summary) of a given input text. The summary should still hold the same meaning of the original input. The summary can be built directly from the words in the input, or it can built using a dictionary.

[1] https://twitter.com/

Since in this project an event is built from a sentence that contains a date, the Headline Generation area of Aumatic Summarization is focused on. In Headline Generation, a summary is built based on the data provided in the input text (its grammar), where a threshold value is given such that the aim is to provide a summary of that size. For Headline Generation there are two main implementations: statistic based and decision (trimming) based [3].

In the statistic based model, Noisy-Channel models are the most prominent, as shown by the multitude of publications [2, 3, 9]. In noisy-channel models, the belief is that the summary of the given input lies within the text but it is surrounded by unwanted noise (text). These systems require a large collection of annotated data (pairs of input text and their summary), which are used in the calculation of the statistical values used to determine whether a produced summary correctly represents its original text. Examples of these algorithms can be found in the works of [3, 5].

The decision based models, are older than the statistic based models and use the grammar of the input text to trim (remove) parts of the inputs until no more rules can be applied or the summary produced is below a given threshold [4]. This is done by tokenizing, breaking an input text into words, phrases, symbols and tagging them each by an identifier(cite). The tokenized text, which is usually represented as a tree where the leaves are the words in the input text and the inner children the identifiers, is passed through an algorithm which applies rules. These rules remove branches of the tree until no more rules can be applied or the summary text falls below a given threshold. The leaves of the trimmed tree is then used to produce the summary. //compare statistics to decision

The trimming based models do not tend to produce as good of summaries as the statistic based model, due to them producing, usually, only one summary while the statistic based models produce a selection to choose from. However, as can be seen from the works of Knight and Marcu [5], the trimming based models can produce better summaries than the statistic models in some occassions. For example, stastic based models are domain dependent as their data set is for a specific domain. Thereby placing statistic models in another domain will cause them to perform poorly compared to the decision models that are domain-independent.

The main advantage of the trimming model is their speed, and not requiring a large corpus of data (like the statistic models) by relying on the grammar to build the summary. In newer works of text summarization, neural network models are used. These fall under the statistical based models. They produce extremely accurate results, but as most statistical models they require a large corpus of data. Requiring a large data set causes algorithms to be processor-

heavy because the annotated data needs to be read, and the stastistical computations need to be carried out. This is not an issue when it is done for a large set of text, but if it is done for every sentence identified with a temporal expressions, then the computation-work is much larger than the input. Thereby, more work is being done for the computation than required.

Due to the time-constraints, it was decided to use the trimming approach, in specific the algorithm provided by Dorr, Zajic, and Schwartz [4] (Figures 1 and 2). Where the given input text is turned into a tree based on its grammatical structure, with the words in the text as the leaves, and the inner nodes being the identifiers (given by the POS Treebank[2]), which is then trimmed. Note that removing a parent node, includes removing its children in the tree. -cite

Algorithm 1: Dorr, B., Zajic, D. and Schwartzm R, (2003). Hedge Trimmer: A Parse-and-Trim Approach to Headline Generation

Input : A Grammatical Tree T of the Sentence to summarize

Input : threshold: Threshold value

Output: A Summary of the given sentence

```
get the leftmost-lower subtree with root S;
remove time expressions;
remove determiners (e.g. 'a', 'the');
while the number of leaves in tree > threshold and there are subtrees to remove with this rule do
    remove all the children except the first, where the rightmost-lowest subtree with root XP and its first child also being an identifier XP (where XP can be NP, VP, or S);
end
while the number of leaves in tree > threshold and there are subtrees to remove with this rule do
    remove any XP (where XP can be NP,VP,PP) before the first NP found;
end
get Tree T' from lastRule;
from T' create a sentence S by reading of the leaves in pre-order;
return S;
```

[2] https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

Algorithm 2: Last Rule

Input : A Grammatical Tree T of the Sentence to summarize

Input : threshold: Threshold value

Output: A Grammatical Tree of the Summary of the input after the last rule has been applied

```
1  if the number of leaves in tree > threshold then
2      make a copy of the tree T';
3      while the number of leaves in tree > threshold and there are subtrees to remove with
         this rule do
4          remove any trailling PP nodes (and their children);
5      end
6      if the number of leaves in tree > threshold then
7          while the number of leaves in tree > threshold and there are subtrees to remove
             with this rule do
8              remove any trailling SBARs (and their children);
9          end
10         while the number of leaves in tree > threshold and there are subtrees to remove
             with this rule do
11             remove any trailling PPs (and their children);
12         end
13     end
14     return T';
15 end
16 return T;
```

2.2 Data Processing and Representation

In the grammatical tree formed, the inner leaf identifiers are given by the P.O.S Treebank[3]. Each word or set of words are given a part of speech tag identifying them. An example can be found below (see Figure 2.1).

On the grammatical tree shown in the figure, the algorithm proposed by Dorr, Zajic, and Schwartz [4] will be applied and the process will be graphically shown in the following figures (see Figures 2.2, 2.3, 2.3, 2.4 and 2.5). Note that the value of the threshold does not force the

[3] http://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

summary to be below it, but it is a length of the summary to which the algorithm is working towards. For the sentence: "On Friday the Washington Post came out with the latest from its long-running investigation into Trump's charitable donations.", the following gramamtical tree is produced 2.1.

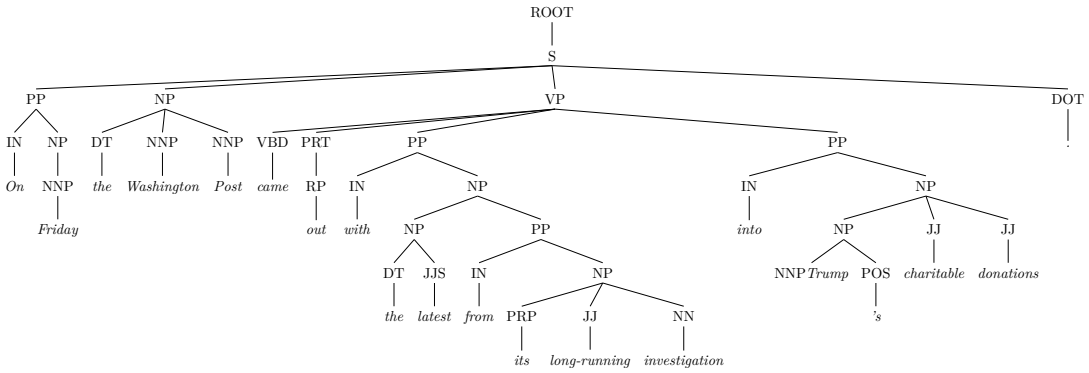


Figure 2.1: P.O.S/Grammatical Tree of: "On Friday the Washington Post came out with the latest from its long-running investigation into Trump's charitable donations."

The inner nodes in the tree are identifiers given by the P.O.S Treebank, and the leaves are the words in the sentence. A parent identifier can be a broader identifier of a collection of sub-identifiers or direct identifiers of a word. Using the Hedge-Trimmer algorithm [4], the tree is procesed as follows. First, the lowest-leftmost S must be identified, as can be seen from the figure 2.1 there is only one subtree with root S. This subtree is extracted, and the algorithm is continued. The next step is to remove time expressions, of which there is only one, "Friday". However, when removing it, the "NP" parent must be removed, else there is a grammatically incorrect summary (given by the Hedge-Trimmer algorithm [4]). The result is graphically shown in the Figure 2.2.

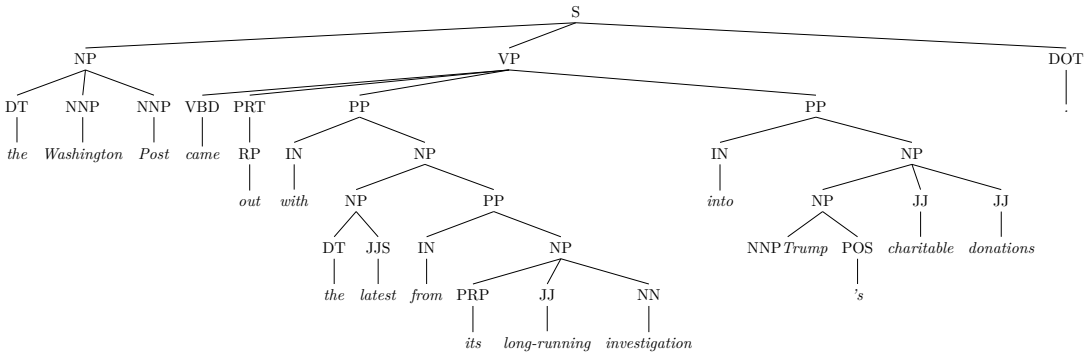


Figure 2.2: P.O.S/Grammatical Tree after removing time expressions

On the resulting tree, the algorithm dictates the removal of determiners. A determiner is identified by the "DT" parent tag. However, not all the parent identifiers with this tag are

removed, only ones that have a child labelled "the" or "a". The resulting tree after applying this rule is given by Figure 2.3.

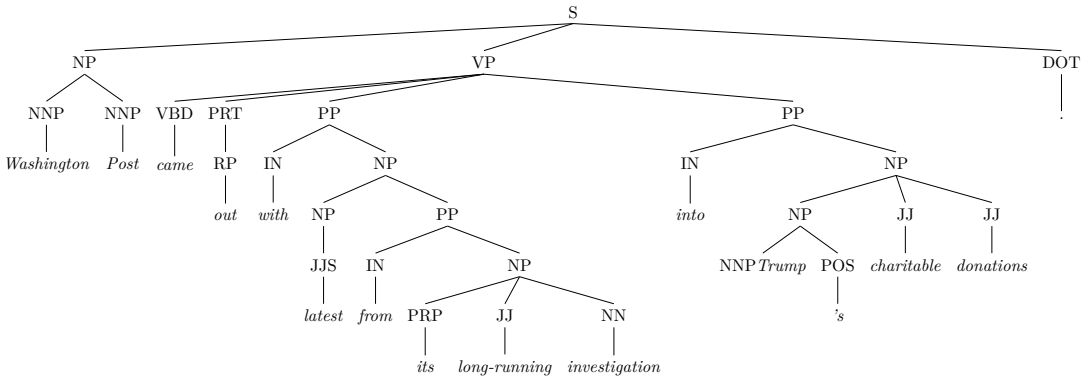


Figure 2.3: P.O.S/Grammatical Tree after removing determiners

The threshold value is used in the following rules. For this example, the threshold value is 10. From the two rules left, only one will be applied as the number of leaves in the resulting tree will fall below the threshold value. Thereby, the summary is below the threshold, so the last rule rule will not be applied. This avoids over-trimming the summary.

The threshold's value importance is of stopping the algorithm from over-trimming or under-trimming. This is done in the sense of there being an optimal summary where the core meaning is kept but if it is further reduced the meaning is lost. When the tree is over-trimmed there is a possibility that the resulting summary does not have the core meaning of the original sentence, or has no meaning at all. The advantage of under-trimming is that the meaning of the sentence is very likely kept, due to the summary sentence being of greater length than the optimal summary. Thereby, having more words and thus retaining the original meaning, as it is closer to the original sentence. However, the aim is to produce a summary, i.e. a short sentence with the same meaning as the original. Under-trimming can ensure the meaning is kept in the summary, but not that the resulting summary is optimal in size (thereby incrementing the time required by the user to view the entire timeline, thus increasing the time the to understand what occurred, which is what the user aimed to avoid with the tool).

When applying the XP-Over-XP rule, a parent identifier labelled XP (where XP can be NP,VP or S), has a first child identifier of type XP also, then all other children of the parent are removed. This rule is done iteratively until the resulting tree is below the threshold, or the rule cannot be applied further due to there not being anymore parent XP-first child XP pairs. The first iteration of the rule is shown in Figure 2.4, and the second and final iteration in Figure 2.5.

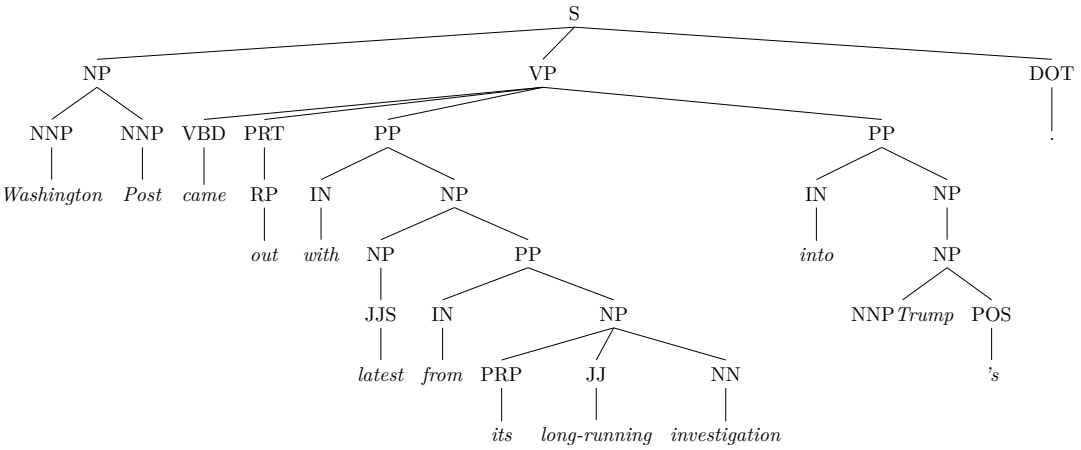


Figure 2.4: P.O.S/Grammatical Tree after XP-Over-XP first iteration

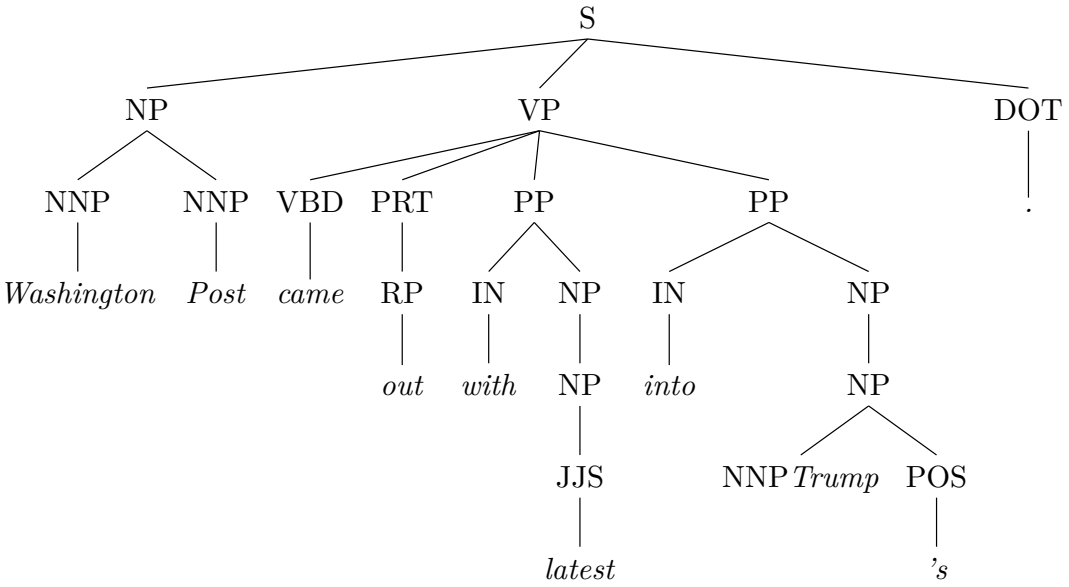


Figure 2.5: P.O.S/Grammatical Tree after XP-Over-XP second iteration

If the number of leaves did not fall under the threshold value, then the next rule to be applied is XP-Before-NP. In this rule, any XP before the NP of the sentence (the grammatical subject) is removed. This would be carried out until the number of leaves falls below the threshold, or the rule cannot be applied further. Finally, the last rule is applied. This rule consists of iteratively removing trailing PP and SBAR subtrees until the threshold value is reached, or the rule cannot be applied further.

The result of applying this algorithm (Hedge Trimmer [4]) on the input text: "On Friday the Washington Post came out with the latest from its long-running investigation into Trump's charitable donations.", produces a summary: "Washington Post came out with latest into Trump's". The original meaning of the sentence, which is that the Washington Post re-

leased a new article on Trump, is kept. However, the resulting summary is not grammatically correct as "Trump's" should be "Trump". Even though the summary is not grammatically correct the summary is significantly shorter than the input text, and does convey the meaning of the original sentence. This is a headline of the input, indicating what the input text was about, and thereby giving a general description of what occurred. -tags are POS (cite) //d -example of input text to tree //d -example of producing summary //d -algorithm //d -what are the options for the summary (Neural Networks vs Decision-Based) //d -give an algorithm for determining the summary, with an example //d

-explain the date problem, with example (determine that it uses an ISO standard)

2.3 Normalizing Dates

An issue when processing documents, and identifying events, is that sentences will not always include the exact date of when an event occurred. These temporal expressions are ambigious. For example, "Yesterday" or "Last week" have different meanings depending on the context in which they are in. To produce a timeline, exact dates are required to sort the events by their date. This is valuable to the user, as events that occur within the same time era will be grouped and separated from events of other time eras. To determine the exact date of an ambigious temporal expression, a reference point can be used. For example, if the text was written on the 11th of March 2017, then "Yesterday" in this context refers to the 10th of March 2017, but if the document was written on the 1th of February 1689, then it refers to the 31st of January 1689. Thereby, it is important to be able to infer reliably and accurately the date to which ambigious time expressions refer.

The reference point (or base date) acts as a context of when an event occurred. Reference points, and similar techniques have been used in other timeline works [6]. The reference point should be the context in which the document was written in, or was aimed to be written in. This can be the publishing date of an article, or the creation date of a document, or any date which allows the exact date of an ambigious temporal expression to be determined. For exact dates that are described in text, such as "On the 12th of December 1996...", the reference point has no value, as it can be determined without it that this text refers to an event that occurred on 12-12-1996. Therefore, the reference only helps the task of producing exact dates.

An issue that may arise is that temporal expressions point to a range of dates, i.e. "In the 1980s...". While it is not possible to determine the exact date, or dates, in which an event occurred it can be determined reasonably [6] that it was somewhere between the start of 1980s,

i.e. 01-01-1980, and the end of the 1980s, i.e. 31-12-1989. In an attempt to produce an exact date, a range of start and end date can be used for this event. This allows for it to be compared to other events, to then sort, and inform the user the event occurred somewhere within that time period.

Chapter 3

Requirements & Specification

3.1 Brief

-purpose of project -how established requirements

The system should take as input a set of documents, process them autonomously (i.e. without the users involvment), and produce a graphical representation of events in the documents. This requires the identification of sentences in the text that contain dates, providing an exact date (to compare to other events), identifying subjects and providing a summary of the sentence. Requirements are generated from this.

3.2 Requirements

3.2.1 Functional Requirements

The functional requirements of a system are behaviours a system should have[1]. In this project the functional requirements are as follows:

1. Process documents of different file types (e.g. .pdf, .txt, and .docx).
2. Identify dates and subjects in text.
3. Summarize sentences.
4. Produce a graphical timeline of the events in the input documents.
5. Modify/Delete events in the timeline.

[1] http://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/

6. Travel between timeline and relevant document.

7. Save timeline (as .pdf or .JSON).

As the software will require as input documents, the 3 most used document file types[2] should be allowable file types in the system. Since the aim is to identify events in the input text, and events are identified temporal expressions, then it is necessary to identify these. The subjects of a sentence are required to aid the description of an event. Subjects, in this case, include names of people, locations, and quantities of money (i.e. key words).

The resulting system should enable users to modify events as it can be the case that the summary, subjects, or date determined by the system are wrong. Providing the ability to edit the events would allow the user to correct these mistakes.

The final two requirements do not affect the processing of the system, but are advantageous to users. Being able to switch between timeline and document will provide the ability to go from a general description of an event to the actual, full-detail, and in context description (which is the original sentence of the event). Providing a save to PDF ability allows the timelines to be included in documents, as the PDF files can be merged. More interestingly, producing an intermediate JSON output makes the system compatible with 3rd-party applications that can provide other graphical representations and/or process further the data. Providing two graphical representations of a timeline (in the system and as a PDF) along with a JSON representation should allow the system to be integrated in documents, reports and other applications.

3.2.2 Non-Functional Requirements

Non-functional requirements of a project are descriptions of how the system must perform the functional requirements, and the qualities the system should have. In this project the non-functional requirements are as follows:

1. A responsive and intuitive UI (Visibility).

2. Reasonable output time (Efficient).

3. Identify the majority of events (Effective).

These three requirements will be evaluated to determine if these are met in the produced system.

[2] http://www.computerhope.com/issues/ch001789.htm

Since the system should be used by any kind of user, with no required technical knowledge, it should be intuitive for the user to know how to use it, i.e. the system should be usable. The user if the system will be discussed later. It would be unreasonable that the resulting system is extremely slow. Such would be the case if on an input n it would require a much larger (exponential) time to complete .

Efficiency relates to the task of identifying events. Identifying events is the most important non-functional requirement, and one of the most important general requirements of the system. The system should be able to extract simple events in text where the full date is mentioned, but also be able to extract more complicated events where the temporal expressions are ambigious.

Extraction and inference of dates on temporal expressions will be further discussed later on. This issue is relevant as in some cases it is known that an event occurred after another, but the specific date cannot be determined. Thereby, producing a timeline of linked events, where an event appears after another not because its exact date suggests so, but the context would allow for the event to be linked in such way. This requires changing models in established NLP tools, and thus will be discussed in the Future Works chapter.

3.3 Limitations

The greatest limitation of the project is time, both in its development and in the execution of the system. As the project time is limited, compromises have to be made. For example, a noisy-channel neural-network summary system would produce different plausible summaries for a given text, of which one should be a reasonable summary. However, as mentioned previously, noisy-channel models require a large amount of annotated data (and thus are domain-dependent). Thereby, providing this set of data would require more development time, which would not allow for the completion of the project. In addition, loading large models of data to summarize one sentence, would have a substantial impact in the running time of the system. From the users point of view, if the system is not significantly faster than producing timelines manually, then they would prefer to do them manually as they are more accurate.

A limitation in all NLP projects is the technology. Modelling context in systems is extremely difficult and non-trivial (cite). For this project it is a clear issue, that for some inputs, poor timelines will be produced as the context of the text was not fully modelled, and thus understood by the system. The issue can be applied to ambigious temporal expressions, where the exact date referenced in the sentence cannot be determined. Solutions to these problems proposed by NLP researchers include looking at sentences related to each other (instead of independently)

and thereby linking references. However, this still does not mimic the context understood by humans during read or spoken interactions.

3.4 Additional Aims

-open source (with documentation), to be used in 3rd parties and allow the project to be further developed

An aim of the project is to make it open-source by providing it on GitHub along with a license. The license would allow anyone to use the system. This is benefitial as the system can be integrated in other programs (directly, as a library, or indirectly, through JSON). All the libraries used allow for the system to be open-source.

Chapter 4

Design

-objectives -use cases (actors) -architecture of backend -design patterns -gui (with screenshots)

For the design, a clear set of objectives, use cases and architecture have been developed to aid the implementation.

4.1 Objectives

The Objectives of the system are for it to be visible, efficient, and effective.

Visible - It should be visible to the user what functions are available. This includes being able to distinguish actions from informative text. Thereby, providing an application that can be picked up and used with minimal to no training and benefiting the growth of users of the system. This can be achieved by providing a simple and intuitive User Interface (UI), where buttons are highlighted, and actions are not cluttered, instead only necessary actions are provided.

Efficient - The time spent to perform tasks should be reasonable in the context of its input. This can be further expressed in the mathematical notation of Big-Oh. If an algorithm takes an input of size n, and roughly performs n operations to produce a result, then the algorithm is said to have a runtime of Big-Oh of n (visually shown as $O(n)$). This allows the efficiency to be compared between algorithms, as the focus is on how well they scale with larger inputs. The objective is to not have an algorithm that processes the files with an exponential time complexity. With an exponential time complexity, a small n would lead to a large running time, therefore a larger n would result in an infeasible running time. Threads can be used to aid this task. A Thread is a lightweight processor computation unit. Using more than one-thread allows for tasks to be carried out in parallel. Therefore, if the input is n documents, and

there are n threads, then the running time of the system would be the greatest running time of all the documents being carried out. Since all documents are being processed in parallel, completely independently of each other, then the time of the document that takes the longest time to process would result in the time it takes to process the entire set of documents.

Effective - The system should meeet its purpose. Which is to require as input documents and process them to produce a timeline. The system should allow the user to perform these tasks through menu options, buttons, and, in addition, produce appropriate responses. When an error occurs the system should not attempt to process the documents idefinetely, and instead produce a timeline with the available events. Effectiveness also involves how correct are the timelines poduced by the system. Whether they provide the user the correct information, and if the events are being identified in the documents.

4.2 Use Cases

A use case is a task an actor in the system performs. An actor is any type of user of the system. In this case, the user can be a law professional that uses the system to have a general understanding of a set of documents. Therefore, it can be assumed that the user does not necessarily have experience with NLP. It should be transparent to the user how the documents are being parsed, and only if they are interested would they require to look at the available source-code. The technical skill of the user does not need to be of an expert, as the tasks required are to provide documents, and view and interact with the produced timeline. In some cases, the user may produce their own graphical representation of a timeline and just use the produced JSON of the system. This would be the case if they have develoed their own system to interact with this system.

The use cases of the system are given by the requirements, and they are presented below. //use case stick figure diagram

1. Load Documents
    (a) Primary Actor: User
    (b) Goal: load set of given documents, where the document file types can be .pdf, .docx, or .txt.
    (c) Main Sequence:
        i. User selects the "Load Documents" option.

ii. System prompts a File Selector.

iii. User selects set of documents and the base dates (or reference dates) to use with them.

iv. System responds with timeline of events.

2. Swap from Timeline to Document

   (a) Primary Actor: User

   (b) Goal: show the sentence, in context, that produced the given event.

   (c) Main Sequence:

      i. User selects event.

      ii. System responds with dialog to "Edit Event" or "Go to Document".

      iii. User selects "Go to Document" option.

      iv. System opens new window with the text of the document where the event originates from, with the sentence that produced it highglighted.

3. Edit Event

   (a) Primary Actor: User

   (b) Goal: modify the data of an event.

   (c) Main Sequence:

      i. User selects event.

      ii. System responds with dialog to "Edit Event" or "Go to Document".

      iii. User selects "Edit Event" option.

      iv. System responds with dialog with the data of the event set in fields.

      v. User edits the data as needed.

      vi. System validates the entered data, and saves.

4. Save Timeline

   (a) Primary Actor: User

   (b) Goal: save the produced timeline as a PDF or JSON.

   (c) Main Sequence:

      i. User selects "Save To..." option.

ii. System responds with option dialog to select the file format to save.

iii. User selects the needed file format.

iv. System responds with File Selector.

v. User selects the location to save the timeline.

vi. System generates the required data to save the timeline in the desired format and attempts to save it in the system.

//note load documents use case includes adding to an existing timeline, user is the person using the system, edit event incl delete, checks for invalid data, file not available

The main sequence are the steps of the interaction in the use case to reach the goal. An error can occur during the interaction, it should be the systems responsibility to deal with the error appropriately, and not end the execution of the program. The primary actor, is the agent or entity that initiates the use case (cite or footnote).

Note that loading documents includes both when it is the first set of documents to be loaded, i.e. the timeline is empty, and when there is already a populated timeline. In the latter case, it would be beneficial to discard duplicated events. A duplicated event is one where it is produced from the same file, using the same reference date, and the data is equal. This is done to not clutter the timeline with events that are repeated, as the timeline should be efficient in the data it provides, i.e. describe the events in the document with as little as possible of additional data. It can occur that two documents produce the same event, these should not count as duplicated, as the event may have a different context depending on what file they originate from. Reference dates are included in the check of duplicate events as it may be the case that different events are produced for the same document when different reference points are used, and the user is interested in investigating this.

When an event is being edited, the user should have the option to delete it, as it may be that the system produced an erroneous event or the event is not relevant to the user. Events may not be relevant to the user if the same event occurs in two separate documents. This can occur when the events described in the documents overlap. The duplicate check would not identify them as duplicates as they originate from different files. To ensure the date entered by the user is correct, it is validated. The validation checks are carried out before the changes are saved.An example of invalid data is when the date of an event is modified but instead of a new date being set, other text data is entered. In the case of the event occuring during a range of dates, i.e. it has a start date and end date, a validation check should be that the second date does not occur before the first. The checks should be carried out before the data is saved.

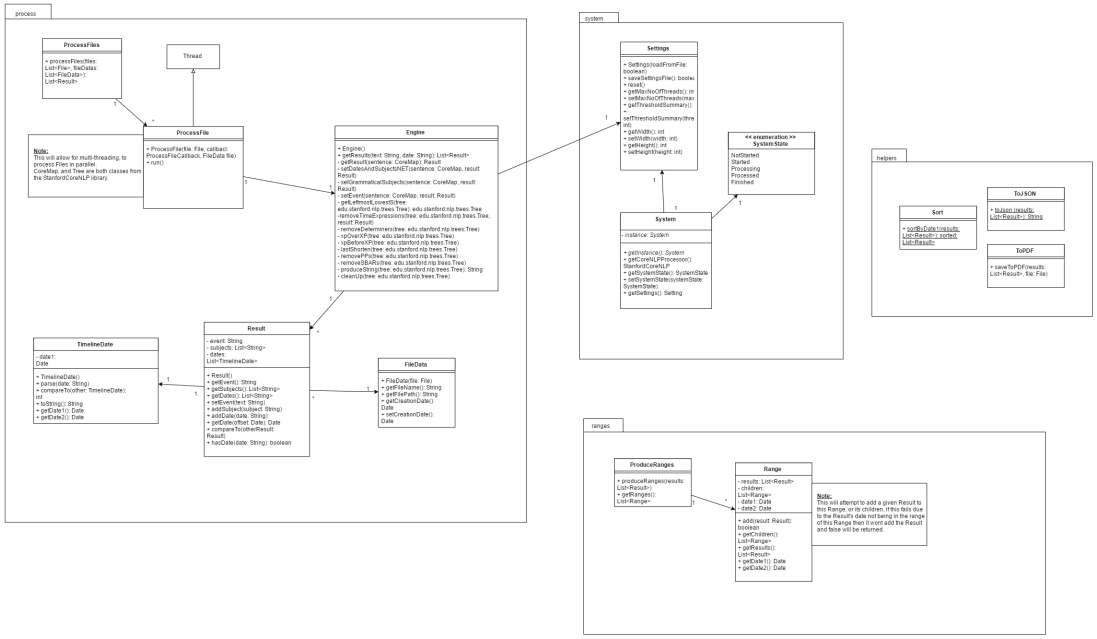
in the case the validation fails, the changes should not be changed, instead the user should be prompted to correct their input.

During the saving use case, the desired location can be unvailable. This can be either because the Operating System does not allow the alpplication to write to that directory, or because a file that is in use is being overwritten (i.e. a lock has been placed on it). Therefore, the user should be prompted to save in another location.

4.3 Architecture

The architecture of a system, is the structure of the components in the system[1]. The focus is on the back-end, or logic, of the system as opposed to the graphical, or front-end. A good software architecture is one where the components of the system are encapsulated with other related components. For this project, the software architecture has been produced using Unified-Modelling Language (UML). In UML, components or classes are represented by a rectangle, with their available functions listed. The architecture of the whole system is presented in the figure below (Figure 4.1). Each package will be looked at individually.

Figure 4.1: Full UML Architecture of Automated Timeline Extraction system.

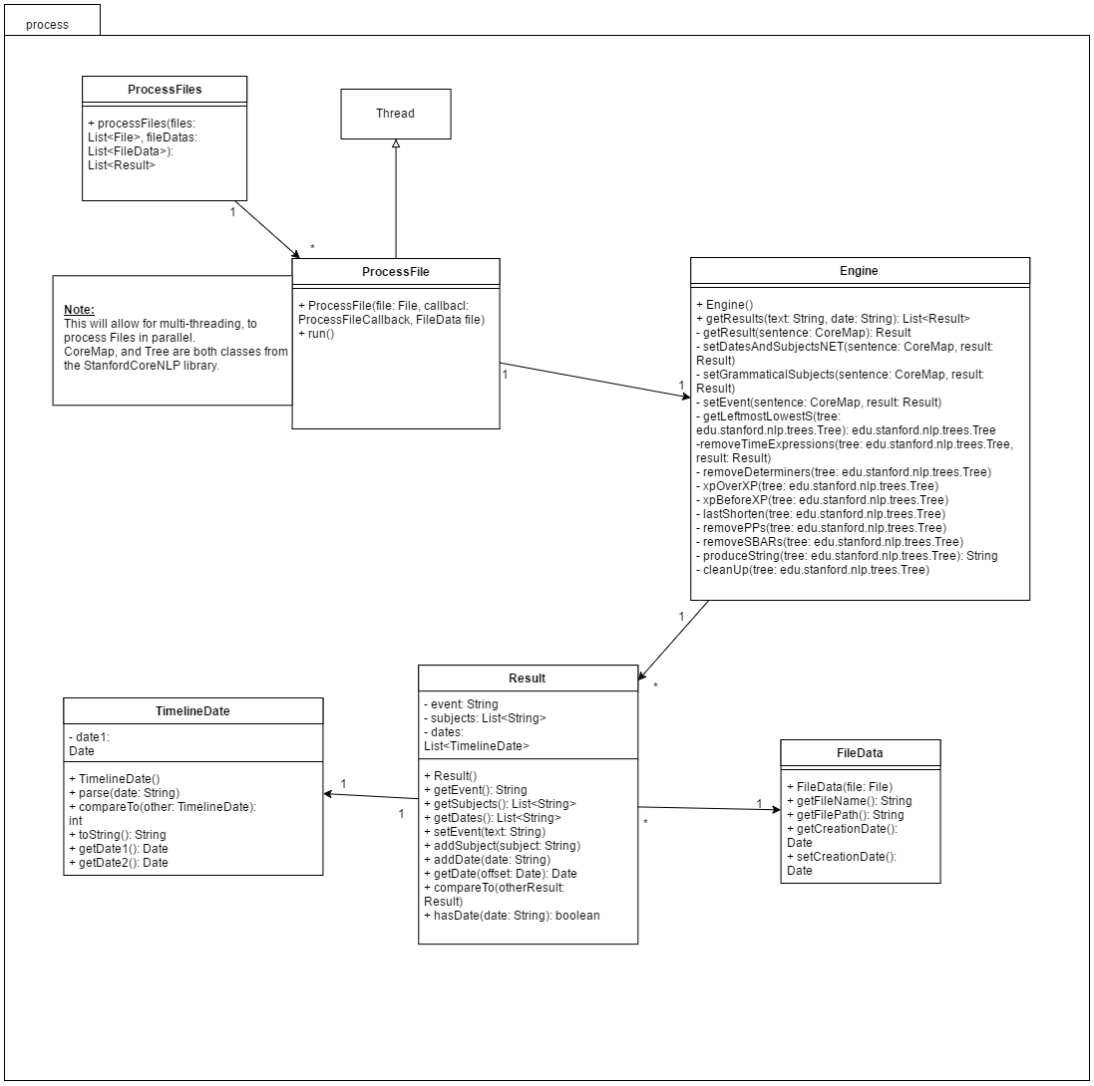


[1] https://msdn.microsoft.com/en-gb/library/ee658098.aspx

4.3.1 Process Package

The core of the system is the process package (see Figure 4.2). The architecture used here is Business Delegate, where all the interaction to the package is through one component, ProcessFiles, which delegates the work to the other components in the package. A list of files is passed in, along with their data (such as the file name, its path, or the creation/reference date used). Each file is processed in parallel. The maximum number of parallel processing allowed is given by the settings of the system (in the settings package). This is the maximum number of threads that can be ran at any given point. However, in the implementation one more thread should be added to the count, as the graphical user interface always runs on a separate thread. The belief is that with a maximum setting of n threads, then at any given point at most n files are being processed. Whenever one file finishes processing, another begins to be processed. As mentioned before, if n files are allowed to process at any given point, and the input size of documents is n, then the time it takes for the system to process all the documents is given by the greates maximum time to process one of the n files. The pseudo-code for this is given below (see Algorithm 3), in the implementation semaphores can be used to aid this task. A semaphore is a data structure that limit how many threads run by requiring threads to acquire a lock before performing their parallel processing. The lock is released afterwards by the thread. If no locks are available, threads wait until a lock is available.

Figure 4.2: UML of the Processing Package



Algorithm 3: Algorithm for processing a list of Files

Input : A list of Files to Process
Output: A list of Results

```
foreach File in the input list do
    wait until can run;
    /* if the maximum number of threads running in parallel has not been
       reached then stop waiting, else wait                              */
    process the file;
    add the produced Result to the list of Results to return;
end
return list of Results;
```

For this system, an event is described as a Result that contains date information (represented by TimelineDate), a set of subjects (or key words), and the summary of the sentence that

produced the event.

The TimelineDate component processes temporal expressions identified by the Engine, these are parsed and an exact date is produced which is then used to compare and sort the Results(events).

In this system, the StanfordCoreNLP suite [2] is used. It is a well-known and tested tool for NLP. Other tools exist, such as ApacheNLP, however Stanford's tool has a larger set of documentation and support, as well as being thread-safe and efficient. Thread-safe refers to the ability to share this tool between separate processes without having to worry about concurrency issues.

When a file is processed, its text is extracted, which is then parsed through the Engine. The Engine is responsible for producing the Results for a file. It identifies sentences with dates, and for those extracts subjects such as people, locations, money, etc. As well as using the Hedge-Trimmer algorithm, discussed in the Background Chapter, to produce a summary.

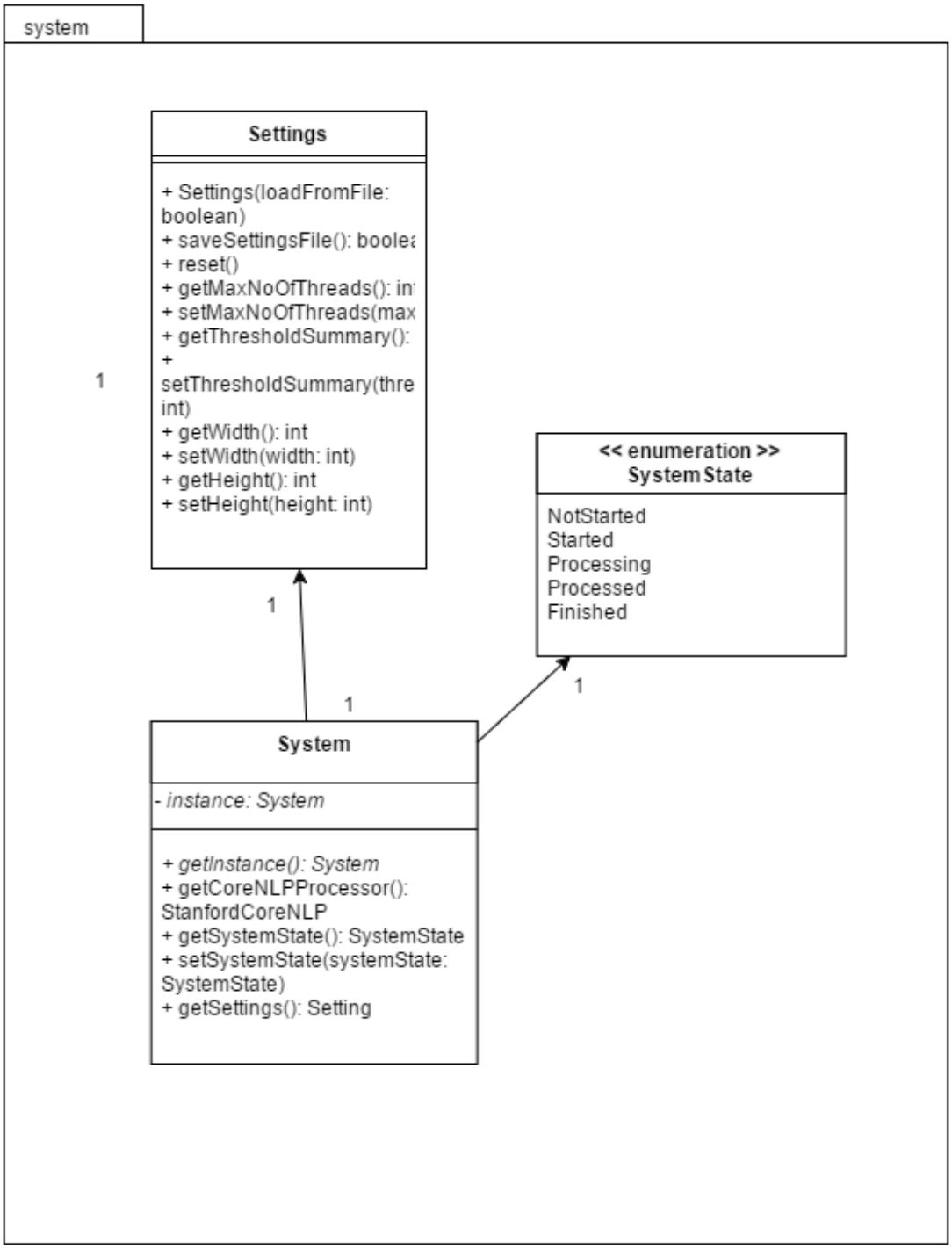
4.3.2 System Package

//talk about system to be shared throughout the system (same data shared)

The system package holds the system and settings components (see Figure 4.3). It is responsible for providing global settings, such as the maximum number of threads to be ran in parallel, the threshold value used in the summary algorithm, and graphical settings. The SystemState attribute is used to identify at which stage the system is in when it is processing files. This allows for the logic of the system to be decoupled from the graphical representation, as it can use the system state to determine which actions are available, when a loading bar needs to be shown, and when the timeline is ready to be displayed. The component is shared throughout the system, as it contains data required in different components (both logical and graphical components).

[2] http://stanfordnlp.github.io/CoreNLP/

Figure 4.3: UML of the System Package



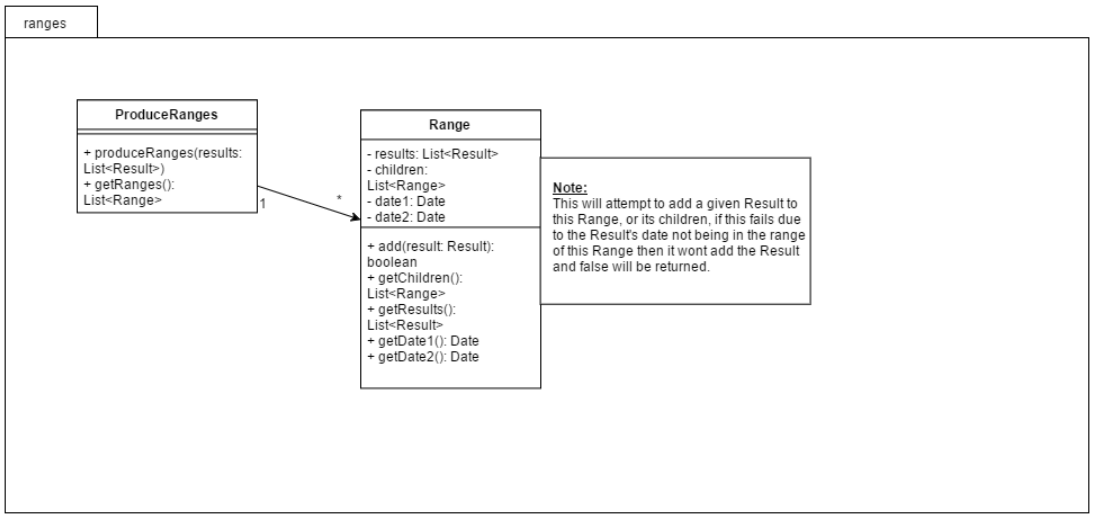
4.3.3 Ranges Package

//trees of ranges with results, algorithm, example, running time

The ranges package was the last package developed(see Figure 4.4). It focuses on placing Results into ranges (see the Algorithm 4), hence the name. A Range is defined by a start and

end date. A Result is placed within this Range if it has the exact same start and end date, if this is not the case then it is attempted to place the Result in one of the children of this Range. If this is not possible, then another Range root is checked. Since Ranges are similar to Trees (as they recursively include other Ranges), there may be a fores of Ranges in the system. The roots is a Range with the largest range of dates that encupsalets its child Ranges (and their Results). Thereby allowing to group related events together, and encapsulating them by their dates.

Figure 4.4: UML of the Ranges Package



Algorithm 4: Algorithm for placing Results in Ranges

Input : A list of Results
Output: A list of Range roots, i.e. a forest of Ranges

```
1 sort the list of Results by the number of days in between the start and end date in
    descending order;
2 foreach Result in the sorted list do
3 |   attempt to add it to one of the existing Range roots;
4 |   if failed to add to existing Range then
5 |   |   make a new Range using the data of the Result;
6 |   |   add the new Range to the list of Range roots;
7 |   end
8 end
9 return list of Range roots;
```

The algorithm proceeds as follows. The Results are sorted by the range, the number of days between their start and end date. A Result that only has one date, has a range of 0. The Results with the largest ranges are added first as it is more likely that they encapsulate other Results (does not apply the other way round). This leads to the production of a tree, shown below. The root is a Range with a start and end date that encapsulates all the dates of its

child Ranges. It may be necessary to expand the dates of a Range if it is the case that the dates of a Result partially overlap a Range. An expanded Range has no results, but has two children: the newly made Range for the Result that was being added, and the Range that the Result was previously partially overlapping.

To demonstrate the algoritm an example will be presented with the two Results presented in the table 4.5 and the pre-existing tree in Figure 4.6. First, the Results would be sorted by their range, such that the second result is the first to be added. This produces the tree in Figure 4.7. Then the next Result in the table is added, producing the tree in Figure 4.8. For the resulting tree, it was determined that the Result being added overlapped an existing Range, threby a new Range was formed that would encapsulate the previous Range and the result being added. This is done by extending the start and end dates appropriately. To place the Result in the tree after the Range was expanded, a new Range was created to hold it, and the previous Range that was partially being overlapped is now a child of the expanded Range.

As can be seen from the tree produced, the largest start-end date pair encapsulates the smaller start-end date pair, this can be done recursively. This allows for a graphical representation where the events are encapsulated by their dates, providing an alternative view to the traditional top-down timeline.

Result	Start Date	End Date
1	12-12-2016	18-12-2016
2	13-12-2016	20-12-2016

Figure 4.5: Example Results Start and End Date

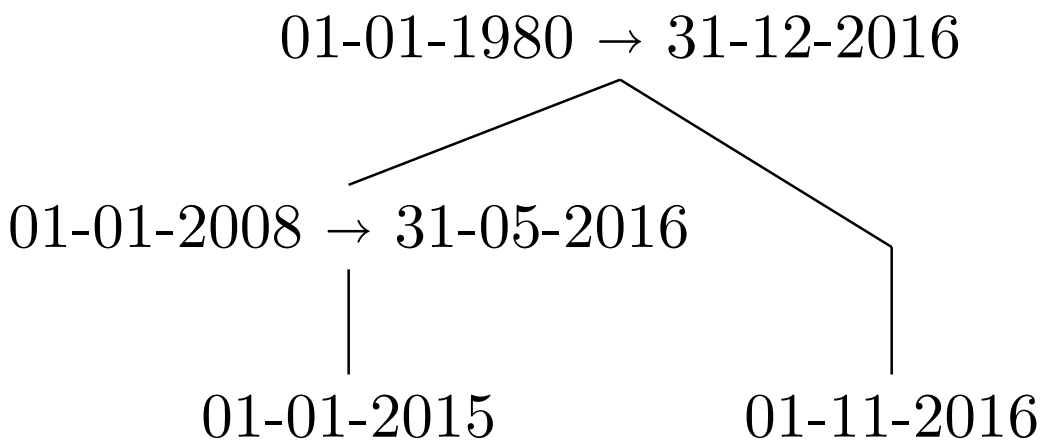


Figure 4.6: Pre-existing Range Tree

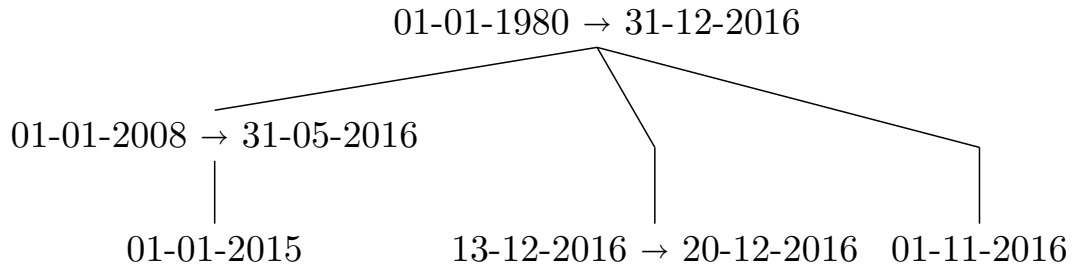


Figure 4.7: Resulting Range Tree after adding the Result: 13-12-2016 → 20-12-2016

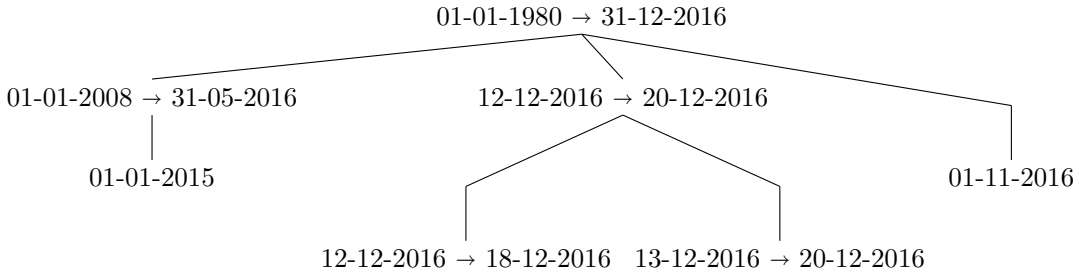
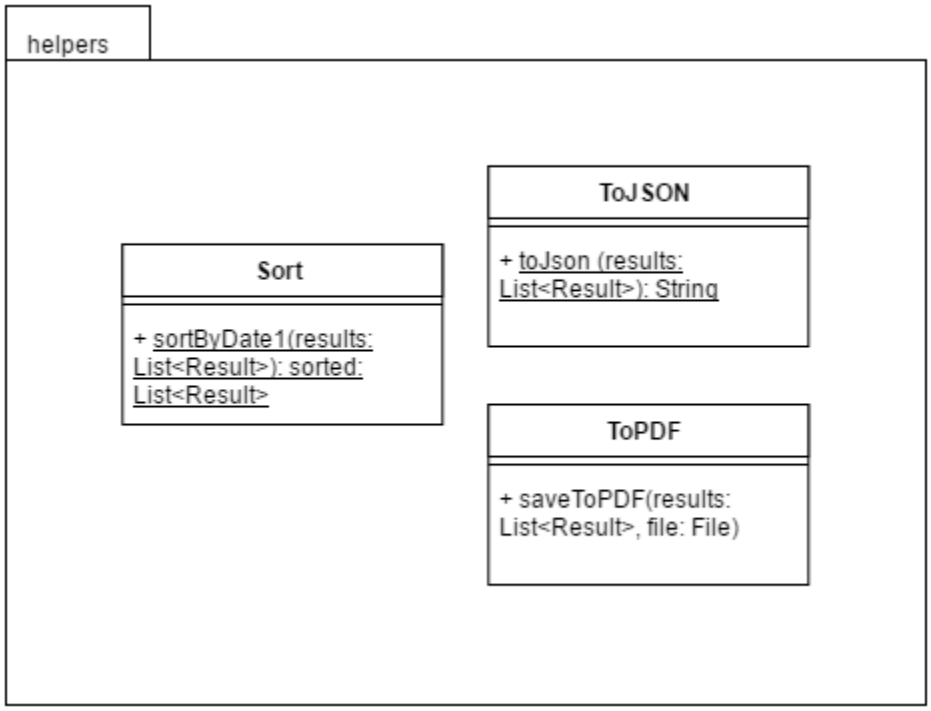


Figure 4.8: Resulting Range Tree after adding the Result: 12-12-2016 → 18-12-2016

4.3.4 Helpers Package

The helper package utility components required throughout the system (see Figure 4.9). For example, it provides sorting functions (sorting by the start date or by the range of days in between the start and end date) and saving the timeline in PDF or JSON format. It is a refactored package of reused logic in the system. Refactoring is done to remove repeated components and operations. Thereby having them in one place only instead of being duplicated around the system. The main advantage is that when a change needs to be made, it is made in one place, but if the operations are duplicated throughout the system then that change needs to be carried out at each duplicated occurrence.

Figure 4.9: UML of the Helpers Package



4.4 Design Patterns

Design Patters are general solutions to common problems in software development[3]. The solutions usually include a system architecture to follow. For this project, the main design patterns are Singleton and Observer.

Singleton - This design pattern ensures that only a certain number of instances of a component are available[4]. In most cases the number of components is restricted to one. The advantage of this pattern is to provide a universal access for global data throughout the system. For example, in the System component the NLP processing tool is held, along with the system state. These are both used throughout the system, hence it would reasonable to use a Singleton pattern to ensure their availability, and not reload them into memory each time.

Observer - This design pattern allows an observee to notify a list of its observers. The observee can be a model, and the the observers its view. This is used to separate the logic of the system from its user interface. It allows for the back-end of the system to be developed completely independently from the front-end. In this system, the front-end will be notified by the back-end. This is then strengthened further through the system states available. As the

[3] https://sourcemaking.com/design_patterns
[4] http://www.journaldev.com/1377/java-singleton-design-pattern-best-practices-examples

back-end begins processing documents, its state changes. The front-end should then change appropriately in such case, by retrieving the relevant data and showing the relevant for that state. If the system were to be developed further, with other graphical interfaces used or added to the pre-existing UI, then these could be added effortlessly as the logic of the system is independent of its view.

Model-View Controller - It is mentioned as a main design pattern of the system, because it is similar to the Observer design pattern, in that it separates the view from the data (which is a model) through a controller. The controller manipulates the data, and applies the appropriate changes in the view. It separates the logic of the system with its view, like an Observer pattern. This design pattern is enforced by many Graphical User Interface (GUI) frameworks, incluiding the one used in this project (JavaFX[5]).

Other design patterns are available, however these are the most relevant ones for this project. While the solution to ensure safe multi-threading is not a design pattern, it is worth mentioning as it allows for safe independent processing of documents through the use of semaphores (mentioned previously). //singleton, model-view-controller, describe, explain why, and what other options were available

4.5 UI

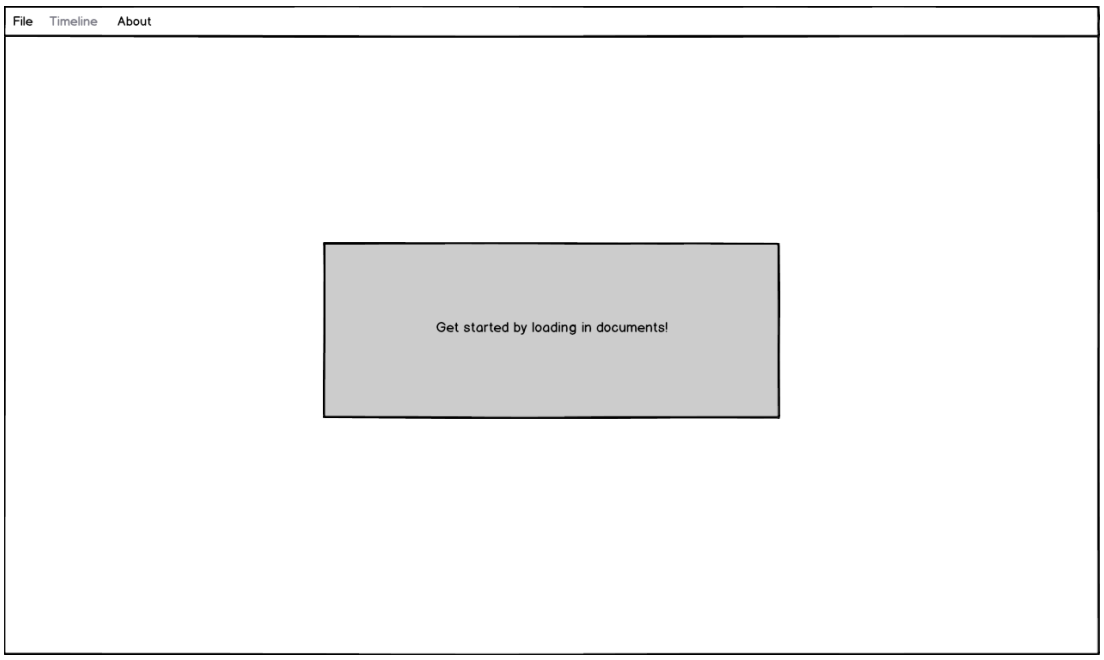
The system is intended to be used by law professionals. However, it can be used by anyone that requires its services of producing timelines, as it will be open-source. These considerations are taken into account during the development of the User-Interface. For example, as the main data representation of the system is the timeline of events, it is clear that it should occupy the majority of the screen. Options that are not relevant to the current state of the system should not be available, while options that are most likely to be used should be highlighted. The aim is to build a UI that is visible and easy to use for the user.

Actions that are made unavailable ensure that the user is not overloaded with buttons and menu options that do not make sense within the current state of processing the system is in. For example, when the system is initially launched, no timeline can be presented as no documents are available to be processed. Instead the user is invited to load documents, as can be seen by the Wirefram Figure 4.10. Wireframes are colourless screenshots of what the UI of a system should aim to look like, it is used by the front-end designers.

From the start up screen, the user can load documents and the reference points used for

[5] http://docs.oracle.com/javase/8/javase-clienttechnologies.htm

Figure 4.10: Wireframe of System at Startup



the documents through a dialog displayed after the files have been selected. As this is being done, the system can load any resources it requires (such as models) before the files are to be processed. After the documents have been processed, the timeline is displayed (see Figure4.11).

For each event produced by the system, a timeline row will be produced. With each event, options to edit (which will open a dialog that allows also for the deletion of events, as seen in Figure 4.12), and view the sentence that produced the event in its own context (see Figure 4.13). The user is also provided with additional information of which documents have been loaded, along with the ability to load more or remove some of the documents, and save the timeline.Note that the timeline menu item is unavailable when there is no timeline displayed (see Figure 4.10), but when one is displayed its available (see Figure 4.11).

In the Document Viewer (see Figure 4.13), note that the relevant sentence that produced the event, which the user interacted with to view this, is highlighted. This allows the user to see in context the event, as they can read the text surrounding it, giving them a better understanding of what occurred. This also allows the user to swap between the system and the documents. In addition it allows the user to view how each event has been produced and judge for themselves how effective the system is. The user can also jump through the document using the timeline by selecting the view funtion from an event. Therefore, the user is no longer required to read the entire document or scroll through it if they are only interested in certain

Figure 4.11: Timeline of System

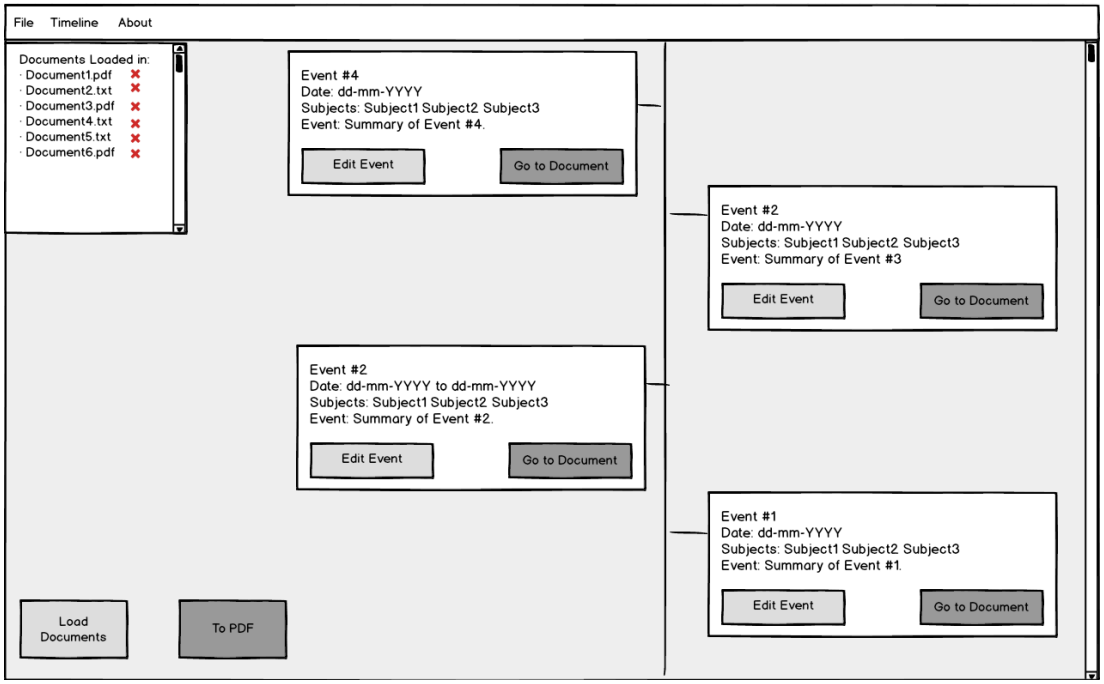
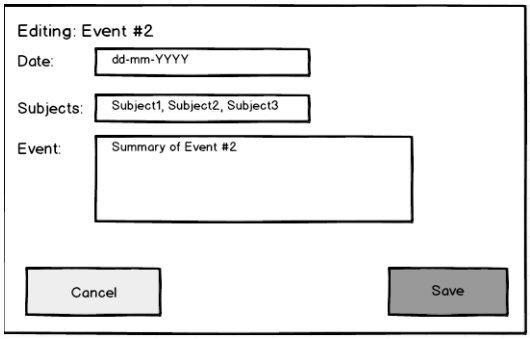
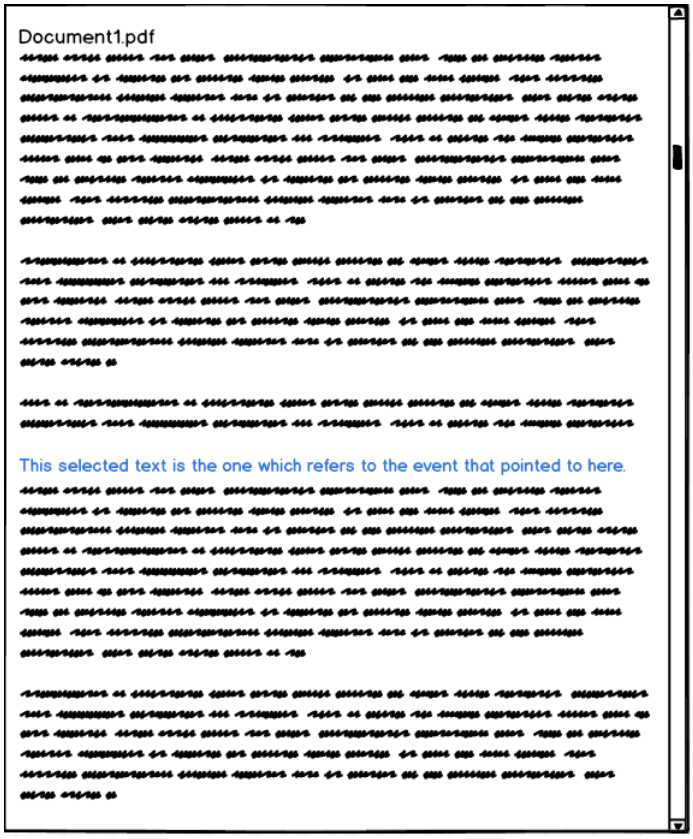


Figure 4.12: Edit Event Dialog of System



areas of it. This is extremely benefitial for large documents which are cumbersome to read, as the user can focus on specific areas of the document (i.e. the text surrounding the sentence that produced the event) instead of having to re-read the chapter, or even the whole document. Note the tool should not replace the reading of highly sensative and critical law documents (or any other kind of documents), as the user may require the full context of the document, but it can serve as a time-effective tool to skip through the document event-by-event, focusing on certain aspects of the documents being revisited. //wireframes, reason for this, why no color, why no other layout?, update to new system

Figure 4.13: Document Viewer of System



Chapter 5

Implementation

-How implemented

5.1 Approach

The development approach focused on the business-logic, or back-end, of the system. From the two main development methodologies, Waterfall and Agile, the latter was used. In Waterfall, the development process is sequential. Development is a sequence of phases that are completed one after the other. When a stage is completed it is not returned to, as like with a Waterfall, it is not possible to go up to the previous stage (cite). Agile development focuses on adaptive planning, evolutionary development, and continous improvement. The advantage of using an Agile approach over a Waterfall approach is that unpredicted features and issues are delt with when they arise. This does make Agile difficult to manage, as there is no clear indication of when the project will be completed.

Scrum is an Agile approach that uses Sprint cycles, which are short development periods. In the development cycles, a set of features must be implemented and tested. Testing allows to ensure that when new features are developed, the older features continue to work as expected. At the end of each cycle, meetings are held discussing what are the next steps, and what issues arose during the previous cycle. For this project, the meetings were with the supervisor.

The Agile approach proved to be useful when a new timeline view had to be implemented. A clear example of the advantage of this system was when the a new timeline view was suggested. In this new view, events are grouped by their start and end dates, such that events that occur within the time period of other events are encapsulated by the longer event. This could be

implemented in the system, due to the separation of the business logic and the view, and the development approach used. In a Waterfall model, the development is more structured, and thereby it is extremely useful for static requirements, i.e. requirements that will not change. However, in this case it would have caused issues in implementing the new view as it would require going up the Waterfall if the view of the system had already been implemented, or waiting until that step of the waterfall had been reached.

Note that the Agile methodology is used in software development teams. However it can be applied to individual developments, since the structure of the project allows for sprint cycles and review meetings with the supervisor, as well the possibility of new requirements.

5.2 Tools & Software Libraries

-development environment -why used that environment -software libraries (include an example use) -why

The development environment of the project includes a 64-bit Windows 10 machine, with an Intel Core i7-6700HQ CPU at 2.60GHz and 16.0GB of Random Access Memory (RAM). It includes a Java Intelligent Development Environment (IDE), with Git for version-control, and Gradle for dependency management.

The use of version-control allows development of features to be separated from the current working version of the code through the usage of branches. New features are added to the working version of the code, only if the required tests pass. Note this follows a Git flow development. In Git flow, a develop branch is made with the newest working features of the system, and new features are developed on separate branches. A master branch will only contain the lastest fully implemented working version of the product. Using branches allows for bugs and development mistakes to be pin-pointed to the development of a specific feature, and allows the rollback to a previous, bug-free, state.

Gradle allows for libraries to be regarded as dependencies of the project. When the system is ran on a separate machine, Gradle will retrieve the missing libaries used in the project before compiling and running the program. This allows for the system to be shared to other users, without having to include the libraries with the distribution of the code, as the required libraries and the version will be downloaded to the users system when they run the command:

gradlew run.

Where gradlew is a wrapper for Gradle, such that the user does not even have to have Gradle installed in their system to run the application. Thereby, providing advantages in portability

and general use.

As mentioned in the Background chapter, multiple libraries exist to aid the task of NLP. These are especially needed for the Named Entity Recognition(NER) and Text Summary tasks. An NER annotator will tag certain words, or collection of words into predefined categories such as People, Companies, Locations, and Money. These tools also aid in tagging words using the POS Treebank, which is a requirement to use the Hedge-Trimmer algorithm [4] to produce summaries. Using an NLP library speeds up the production of the system, as manually building an annotator requires multiple developers and years of work. As can be seen from the release history of the StanfordCoreNLP, where development began in 2010 and the latest release was in 2016[1]. The main two NLP tools available are Apache's OpenNLP[2] and Stanford's CoreNLP[3]. For this project the Stanford's tool was used in the implementation, as it provides an extensive documentation and examples, along with specific sections describing their annotators in detail.

The Stanford tool is the main library used throughout the project, as the project is reliant on its NER and POS annotators (cite the stanford annotators). It comes with models, that are loaded during the initialisation of the system. These models are used in the stasticial calculations in the annotators to determine whether certain words fall in predefined categories, or which POS tag should be given to them.

Due to the two main NLP libraries available being Java implementations, the decision was made to build the system in that language. It would be problematic to build the system in a different language to its libraries, as it would require to make the two programming languages communicate with each other, which can cause unpredictable problems in the development and execution of the system.

Additional libraries in the development include JUnit for Unit testing. This allows for features to be tested. Testing is used in Git flow, to ensure that new features developed do no affect the correctness of older features, and are only merged together if the test succeed. Thereby, ensuring the system functions as expected.

Libraries for text extraction of .pdf and .docx file types are required, as the enconding of these files is not in plain text . The Apache POI[4] and the Apache PDFBox[5] are used. In addition to text extraction, the PDFBox library along with the Apache Commons library allows the creation of PDFs (with text wrapping). This allows to save timelines as PDF documents.

[1] http://stanfordnlp.github.io/CoreNLP/history.html
[2] https://opennlp.apache.org/
[3] http://stanfordnlp.github.io/CoreNLP/index.html
[4] https://poi.apache.org/
[5] https://pdfbox.apache.org/

The Google GSON[6] library is used for the creation of JSONs, to save the timeline in a JSON file. The RichTextFX[7] library along with JavaFX library are used to build the Graphical User Interface (GUI) of the system. It was ensured that the libraries used have licenses that allow their use in this project, and its later relase to open-source.

5.3 Issues

Two main issues occurred during the implementation of the system: the creation of exact dates from named entity dates, and the creation of an encapsulated timeline. A minor issue is also explained, which is based on the input documents being grammatically incorrect.

5.3.1 Named Entity Recognition (NER) of Dates

The StanfordCoreNLP tool, allows the resolution of temporal expressions. To explain this, an example is presented. The Stanford tool allows for reference dates to be used when a document is processed. When the tool tags a temporal expression as a DATE, it attempts to normalize it. Note the annotator treats each word in the sentence as a mention. To identify its named entity recognition tag, the following is done on the mention:

mention.get(CoreAnnotations.NamedEntityTagAnnotation.class).

If it is a temporal expression that was tagged, the result of the operation is a String DATE (to identify it as a date). Thus, from the mention, it can be normalized using:

mention.get(CoreAnnotations.NormalizedNamedEntityTagAnnotation.class).

The Stanford tool will attempt to produce a date in the ISO 8601[8] format. As can be seen from the format, it can produce exact dates of the type dd-MM-yyyy, where dd is an integer value from 1 to 31, MM is the month as an integer value from 1 to 12, and yyyy the year. If BC dates are used, at the start of the normalized result a '-' is added (indicating a minus date). Using the ISO standard, an algorithm was written to process these dates. Note that, in addition to the possible dates given by the ISO standard, the Stanford tool produces 3 additional possible normalization outputs.

Normalizations refer to the attempt to produce a date from a temporal expression. It may be that, even with a reference point, the tool does not produce an exact date. For example, the temporal expression "now" would produce a normalized NER: "PRESENT_REF", i.e. a reference to the present moment. For a temporal expression that refers to the past, e.g. "...they

[6] https://github.com/google/gson
[7] https://github.com/TomasMikula/RichTextFX
[8] https://www.cl.cam.ac.uk/~mgk25/iso-time.html

once used to...", "once" would be normalized to "PAST_REF", i.e. a reference to the past of this moment. For a temporal expression that refers to the future, e.g. "In the future..., "future" would be normalized to "FUTURE_REF", i.e. a reference to the future after this moment.

The issue is that these normalizations do not allow for the comparison of events, which is required to sort them (as the start and end dates are not comparable). To allow for comparison, the most possible general dates for these references are derived. Since a "PRESENT_REF" refers to the present moment, it can be deducted that it represents the moment in which the text was written in, as that is the time context in which the author wrote it. Thereby, the decision was made to produce as a general date for "PRESENT_REF" a date that is the reference point provided by the user. As the reference point is supposed to be the date in which the text was written in. Note that the user can change the reference point to when-ever they would like, not just the assumed written date of the document.

For "PAST_REF" and "FUTURE_REF" a range of dates is used, i.e. a start and end date. For the former, the start date of the era is used, i.e. 01-01-0001, and an end date to the reference point is used. This is because it can be determined that an ambigious mention of the past would fall anywhere within that time period, however an exact determination cannot be made due to the temporal expression not being precise enough. For the latter, the start date is the present moment, i.e. the base date, and the end date is the last possible allowable date in the system, i.e. 31-12-9999. This is because it would be appropriate for a mention of the future to refer to a moment between now (i.e. the present moment in which the text is presumed to be written in), and until the end of times. However as a limitation to our system, the end of times is considered the date 31-12-9999. A snippet of the implementation of the algorithm is presented in Figure 5.1.

Note that even though the StanfordCoreNLP library is well documented, it was difficult to find all the different outputs for the normalization of NER dates. It was after the discovery of Stanfords TimeML usage that a document describing the possible outputs of normalized NER dates was found[9].

[9] `http://www.timeml.org/timeMLdocs/TimeML.xsd`

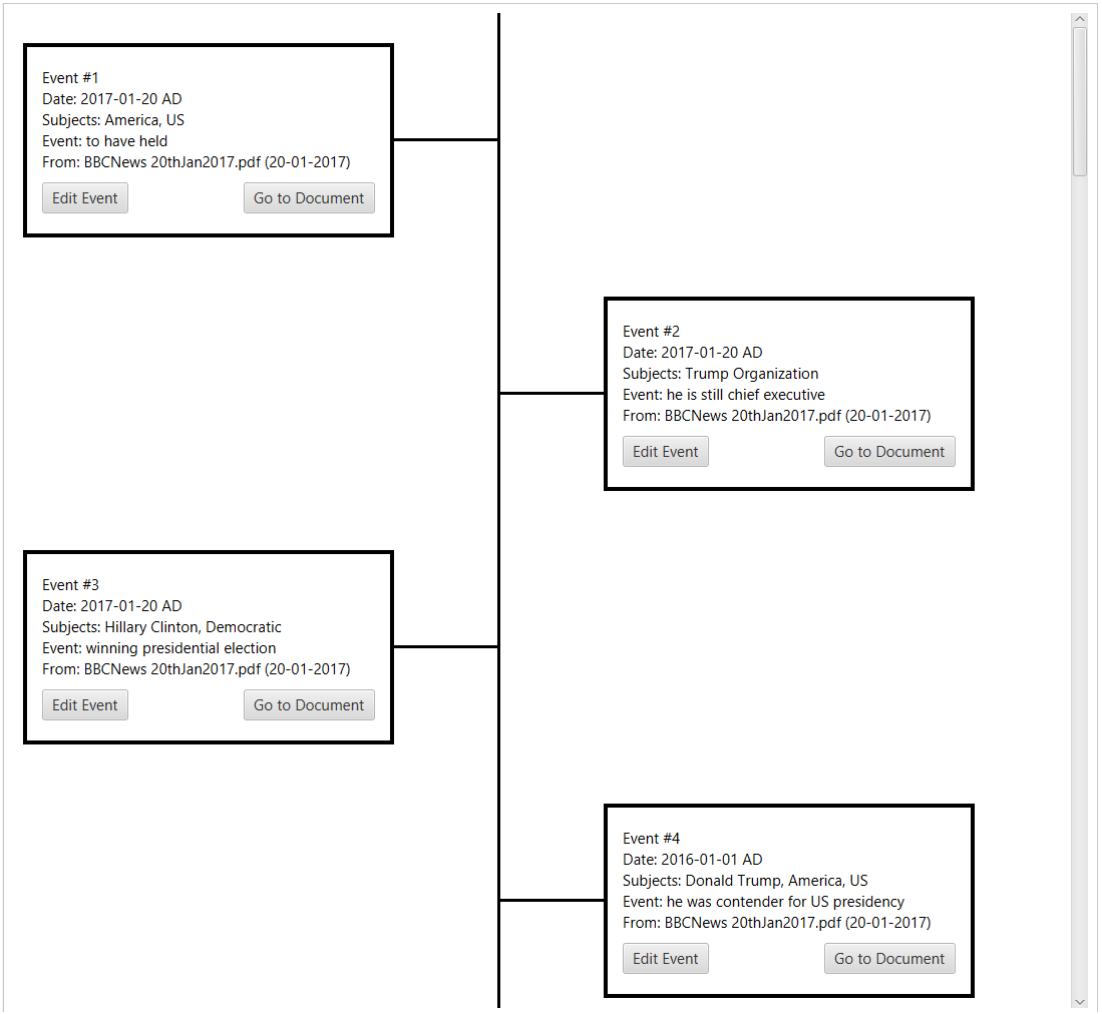
```
private ArrayList<Date> getDate(String date) {
   ...
   /* Set up variables for processing */
   if (onlyPastRefPattern.matcher(date).matches()) {
           //past so make range from 0001-01-01 -> base date (range)
           if (yearMonthDayPattern.matcher(baseDate).matches()) {
               //base date has the format yyyy-MM-dd
              /*split it and set the values in date1,month1, year1 of
      the start year 01-01-0001*/
      /*and the end date values of date2, month2, year2
      to the base date data*/
      ...
           }
       } else if (onlyPresentRefPattern.matcher(date).matches()) {
           if (yearMonthDayPattern.matcher(baseDate).matches()) {
               //base date has the format yyyy-MM-dd
      /*set day1,month1,year1 to the data in the base
       date data*/
      ...
           }
       } else if (onlyFutureRefPattern.matcher(date).matches()) {
           if (yearMonthDayPattern.matcher(baseDate).matches()) {
      //set the day1,month1,year1 to the base date data
           }
          //set year2,month2,day2 to the end year 31-12-9999
       }
   ...
   /* date1,date2,month1,month2,year1,year2 are
   then used appropriately to generate dates used
   for the event that holds this Timeline Date */
}
```

Figure 5.1: Part of the Implementation of Resolution of Normalized NER Dates

5.3.2 Encapsulated Timeline View

The initial representation of the events was a traditional timeline (see Figure 5.2). This view is effective when the events are on separate time periods, as it presents them one after the other in a sequence. However, when there are multiple events that occur during the same time period, they still appear one after the other. The issue is that unless the user specifically looks at the dates associated to the event, it is not possible for them to determine that the events occurred in the same time period, but instead that one occurred after the other. This clearly violates the visibility objective of the system. A solution to this issue is to provide two views: the traditional timeline view which is effective at displaying events that have disjoint dates, and another view where the dates encapsulate the events. For example, if there is more than one event that occurs on the "25-01-2017", then instead of listing them both, one after the other, they are grouped visually by their date. However, visually grouping the events may lead to a "bin-packing" problem. The "bin-packing" problem consists of a set of bins, in this case a set of graphical views, which need to be fit in a finite space, in this case a box of fixed width and height. To deal with this issue, scrollbars can be used in the containers that hold the event layouts. Scrollbars allow for a container to be of infinite height (and/or width). Thus, being able to place the bins (graphical representations of events) in the container without considering the space limit. This view has been given the name "Range View". It requires placing the Results (the events of a set of documents) in Ranges (a data structure that can have one date, or a start and end date). The algorithm was discussed in the Design Chapter.

Figure 5.2: Screenshot of the Traditional View of the Timeline of Events



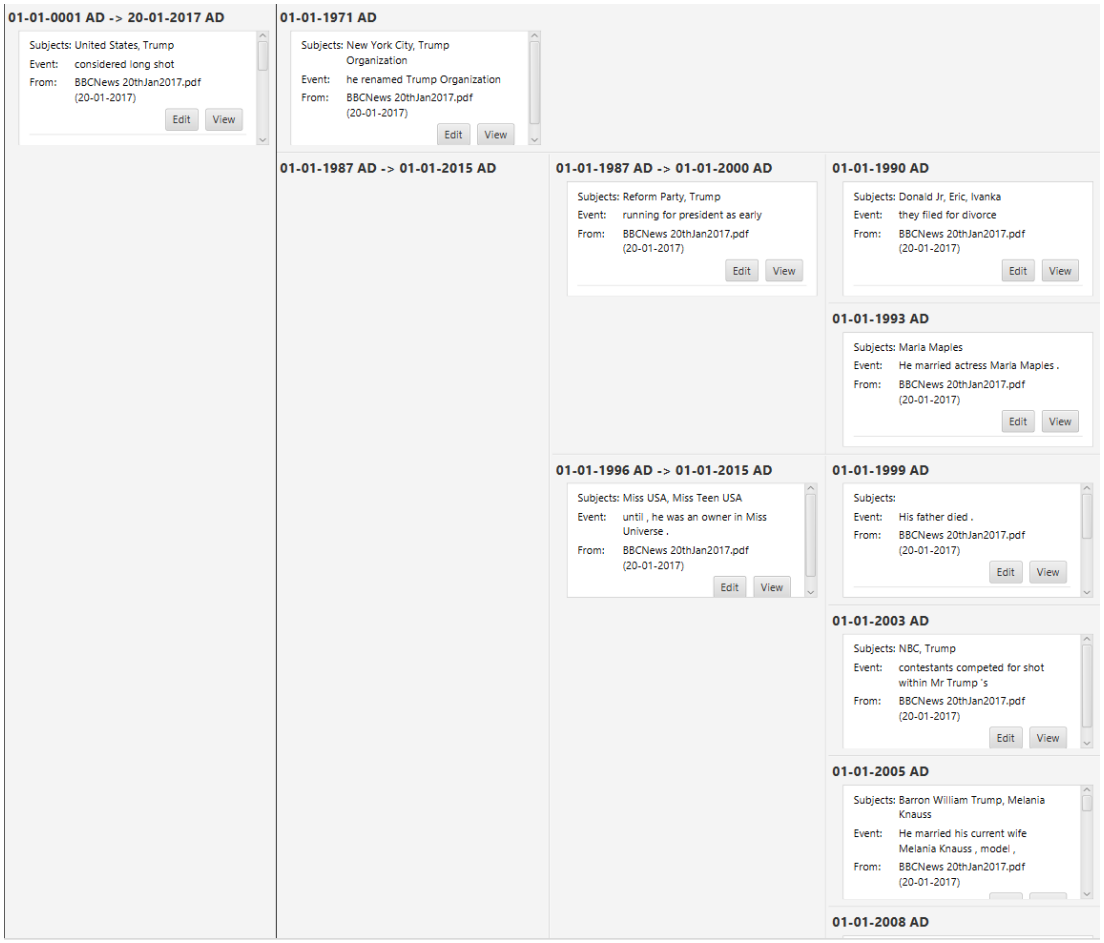
The algorithm behind the Range View, is having a list of Ranges, where each Range is a root node of a Tree of Ranges. Each Range may hold zero or more Results (i.e. events), and a set of children Ranges (zero or more). When the timeline needs to be produced, the list of Range roots is iterated over, assuming the list is of Ranges has been sorted in the order of the start date. For each a GridPane is made. A GridPane is a layout which consists of rows and columns that can contain subviews (where a subview is a view, i.e. a graphical component). In the first column, the current Range's data is placed (i.e. the date(s) and the Results held), and in the second column the layouts of the child Ranges are recursively set. The algorithm is presented in Figure 5. It is carried out for each Range root in the forest of Ranges.

Algorithm 5: Pseudo-Code of the Recursive Production of the Range Layout

```
1  function getRangeLayout(list l);
   Input  : A list of Ranges, of size n, to add in the first column
   Output: a layout that encapsulates the Ranges passed in the input, and their child
           Ranges
2  GridPane toReturn;
3  toReturn set the number of rows to the size of the input;
4  toReturn set the number of columns := 2;
5  for i := 0 → n do
6      Range := input list at i;
7      set up layout for this Range, and set it in toReturn at position (i, 0);
8      set its column span at (i, 0) to remaining;
9      get layout for the children of this Range := getRangeLayout(range.children);
10     set this layout in toReturn at position (i, 1);
11 end
12 return toReturn;
```

In the implementation, it was decided to include 3 columns, so as to have a separator between a Range and its children. Thereby, improving the visibility of the timeline, as the user can differentiate between the Results of a range of dates, and the Results of a sub-range. The individual layout of a Range is listing the Results it holds, and the start and end date for this Range. An example look of the Range View can be found in Figure 5.3.

Figure 5.3: Screenshot of the Range View of the Timeline of Events



5.3.3 Incorrect Input Documents

An important issue relating to the input documents is their format and grammar. NLP tools such as StanfordCoreNLP require certain constraints to be applied onto the input documents to be able to annotate them. These tools apply grammatical rules, thereby it is important that documents are grammatically correct to be able to correctly annotate them. This was discussed in the Background Chapter with a discussion on NLP tools being applied to Tweets. For this implementation, the documents are required to be in English as only the English models are loaded for the Stanford tool. Other languages are supported by the tool[10] (such as German, and Spanish), however these require their own models. In addition, the algorithm applied to produce the summaries only works on English written text, as it uses the English grammar. In the future, the system can be extended to other languages.

An issue discovered, was that sentences that are seperated by a period, but are not followed

[10] `http://stanfordnlp.github.io/CoreNLP/human-languages.html`

by a space, are treated as a single sentence. Thereby, it may be possible that only half the events are detected, if every two sentences are only separated by a period (and not a space). In addition, the summary would be applied to second sentence, as the rule of the algorithm details that the lowest-leftmost "S" subtree (i.e. sub-sentence) is picked. The start and end date, would therefore be the lowest date of the two events, and the highest date of the two events, due to how the dates of events are encapsulated in a TimelineDate object (that parses Normalized NER Dates, and updates the start and end dates it holds if a new minimum or maximum is found).

This issue cannot be prevented, as it would require manipulation of the input documents, and it can be the case that the user does not want the system to manipulate the input, but rather process it. Hence, it is advised to users to ensure the documents are in grammatically correct English, for the best results possible. Since the system will be used by law professionals, that are handling formal documents, it can be assumed that these documents would follow this format. -ner dates (explain, then present how to solve it through examples of code) -new timeline view (present how to solve it through examples of code) -minor issue of incorrect text

5.4 Testing

Unit Tests were the testing focus in this project. A Unit Test is when individual units (or pairs) of source code of a system are tested to determine whether they are working correctly (cite). As the system was implemented in Java, the library used to aid this is JUnit[11]. Unit tests are primarily done on the back-end of a system, as the focus is on correctness in the logic, not in the UI. Automated tests carried out on the UI are called Instrumentation Tests. They involve emulating the users interaction with the system, i.e. pressing a button or filling in a text field. Both Unit and Instrumentation tests are automated, such that they are carried out one after the other without the involvement of the developer.

The reason as to why only Unit Tests were used, is that the systems primary focus is its processing of documents, and not its graphical representation. The representation is used to display the results, however, the system could also be used to process texts. Thus, the resulting intermediate JSON would be used in a third-party visual representation. Thefore, Instrumentation tests were not carried out.

A total of 35 tests were developed. The main advantage of these, is that when new features are developed, tests are ran to ensure the rest of the system functions as expected. If the test

[11] http://junit.org/junit4/

cases are extensive and appropriate, then it can be assumed that the system will work correctly with the new features. In addition, this is aided through the use of Git (in Git flow), where the features are developed on separate branches, and are only merged to the current working version of code if all the tests pass.

The focus was on the Engine (the component that takes as input text, and produces lists of Results), the processing of Files (test files were used in this case), the production of Ranges, the changing of the systems states throughout the processing task, the parsing of Normalized NER dates, and the production of JSON's from a list of Results (a timeline). Tests perform assertions. Assertions are when an actual output of code is checked with an expected output.

Tests were divided into three categories: a simple test, an intermediate test, and a complex test (where all possibilities of input are tested). For example, for the production of Ranges, the complex test case expects multiple Range trees of different heights to be produced. Due to the benefit of Gradle, the tests can be ran using the command:

gradlew test[12].

In the build command, Gradle will not only run the test cases, but also produce the executable JAR which can be used to distribute the system as an executable to its non-developer users.

5.5 UI

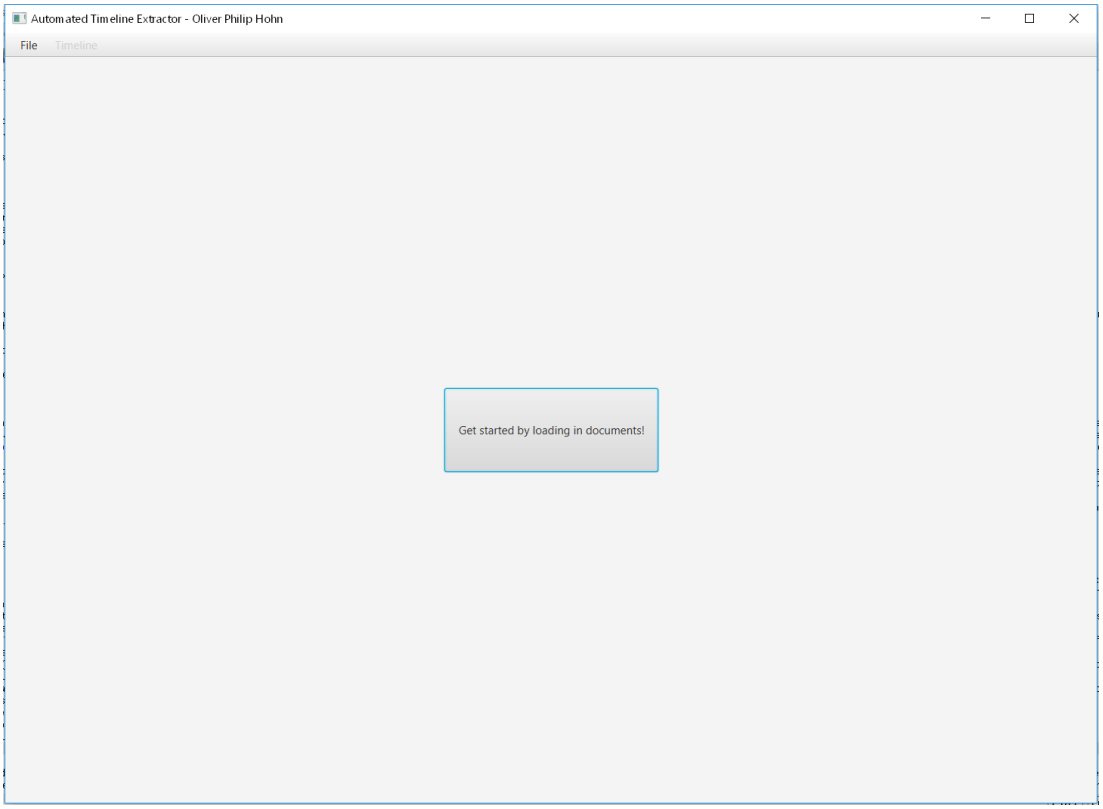
//implementation ofthe UI.

From the wireframes presented in the Design Chapter, the actual User Interfaces(UI) was developed. The screenshots of the different windows are presented in the Figures below (see Figures 5.4, 5.2, 5.3, 5.5, and 5.6). Note that the interfaces provide the requested functionality through the use of buttons and menus. Color is missing as the focus was on functionality, visibility and simplicity, instead of the UI being aesthetically pleasing. The system is aimed to be used for task completion, not enjoyment. The UI was, thus, developed to accomplish these tasks.

Note that the Range view (see Figure 5.3) is zoomable (i.e. can zoom in/out). This allows the user to have a broader image of the Range View, if it is required. Therefore, allowing the user to obtain a general picture of what occurred.

The data presented of an event includes its subjects (key words) and summary. As the events are encapsulated in Ranges, it is sufficient to show the date(s) for the Range at the top. This demonstrating to the user that the following events occurred during the given time period. This separates meta data (e.g. when an event occurred), from the actual event, which

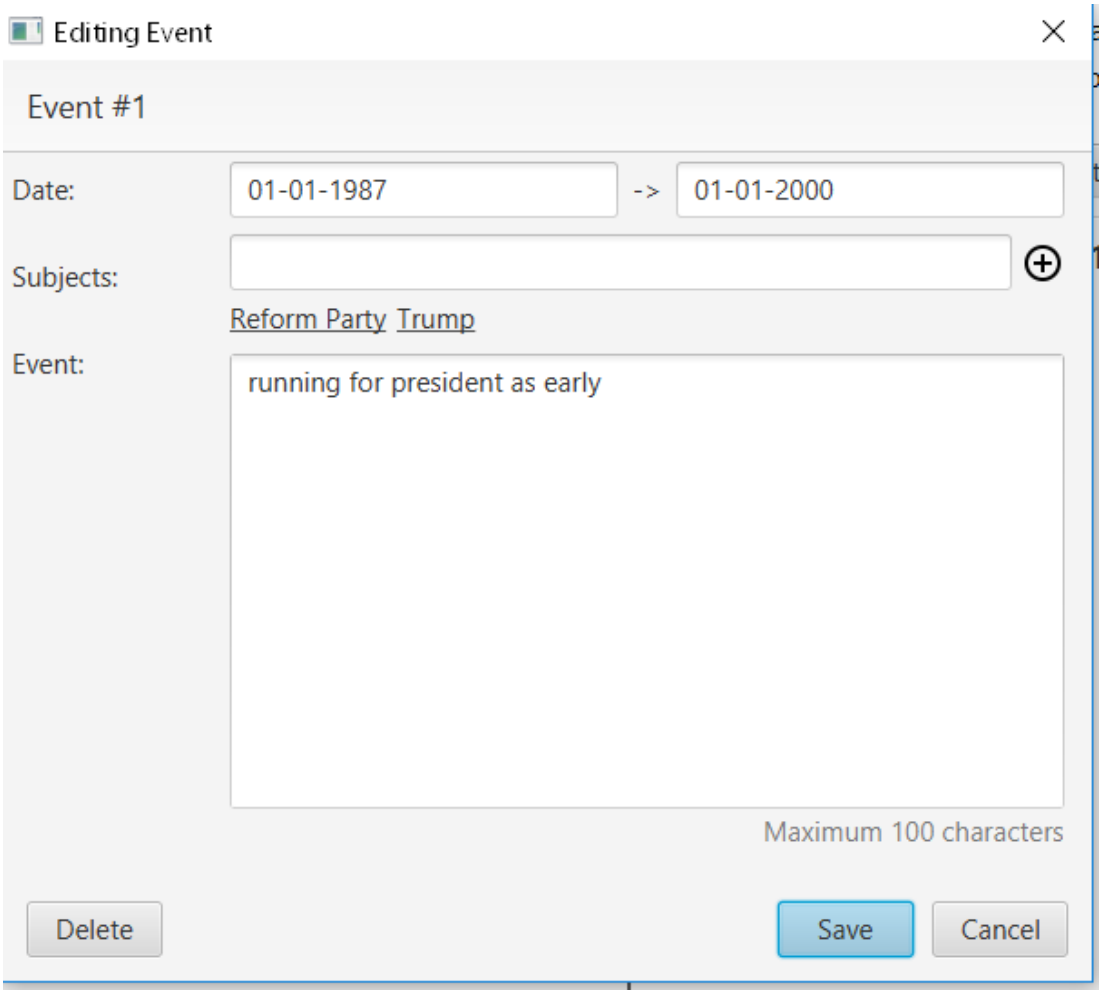
Figure 5.4: Screenshot of the Start Up View



is what occurred in the event. Each event is accompanied by two buttons, the operations are: to edit the event (which includes deleting it, as can be seen from Figure 5.5), and viewing its document.

An advantage of the Traditional View over the Range view, is memory efficiency. Since the Range view has the function of being zoomable, due to programming language constraints, it cannot be implemented through a ListView. A ListView is a layout data structure where objects of a list are placed row by row. The ListView is memory efficient. It only produces the layout for a row when it needs to be shown. For example, for a list of 10000 items, it is very expensive to hold in memory the individual layouts of 10000 rows, especially if only 5 are being shown. Instead, only the 5 that are being shown are held in memory, and the rest are generated when needed. This is not the case with the Range View, due to it having the zoom function, and thus not being a ListView. As the memory capacity of personal computing systems (where the tool is intended to be used, but is not limited to) is large, this would not be an issue for the processing of tens or hundreds of events, however for larger sets this would cause memory issues. For large sets of events, the traditional view is recommended.

Figure 5.5: Screenshot of the Edit Dialog View



5.6 Important Algorithms

//highlight algorithm implementation: processing of files through semaphores, adding to ranges, pdf save and json save

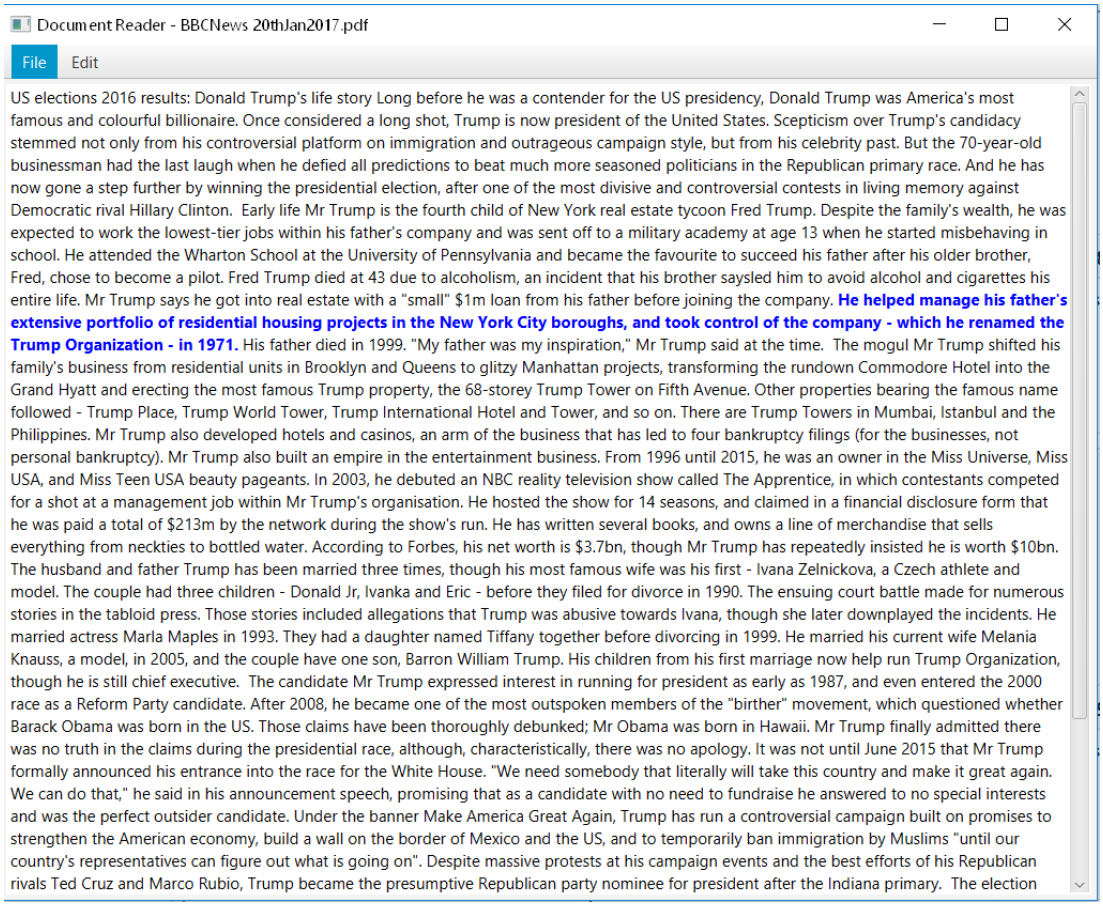
This section presents note-worthy algorithms, in specific their implementation.

5.6.1 Processing Files

The pseudo-code of processing files was presented in the Design Chapter. The use of semaphores was also mentioned. Semaphores enforce that only n processes can acquire their lock, with the $n+1$ process having to wait until another process releases its lock. However, the implementation is not trivial.

The system must allow a certain maximum number of threads to be ran in parallel to process documents. Processes must not fall in deadlock, which is when the processes are indefinitely

Figure 5.6: Screenshot of the Document Viewer



allowing other processes to run, such that no run, or starve, which is when a process waits indefinetely to run. This can occur when semaphores are used, and they are not released when a process finishes its task. To ensure the semaphores are released, a callback is given to the threads, which is to be used at the end of the threads parallel processing. If an error occurs, the semaphore is still released, allowing the next thread to run. Multi-threading precautions must be considered. When the callback is used, data is also transferred (specifically the Results of processing the document) which can cause concurrency issues when two threads attempt to add a result list at the same time. Fortunetely, Java provides the **synchronized** keyword for methods (see Figure 5.7). This ensures that no two threads may run the method at the same time. One thread must finish its execution of the method before another one can begin its execution.

```
public synchronized void callBack(ArrayList<Result> results, FileData fileData) {
    //we finished processing a file
    filesToGo--;//one less to look at
    //add the results to the list held
    this.results.addAll(results);
    //release semaphore
    semaphore.release();
    //check if we have processed everything,
    //if so release the finished semaphore
    if (filesToGo == 0) {
        //has processed
        BackEndSystem.getInstance().setSystemState(SystemState.PROCESSED);
        //has returned the results so we finished
        BackEndSystem.getInstance().setSystemState(SystemState.FINISHED);
        semaphoreFinished.release();
       //value is now 1, so the thread that was acquiring can continue
    }
}
```

Figure 5.7: Implementation of Callback after Files have been processed

Threading precautions have been taken throughout the project. Where sets of data are processed by threads, it was ensured that the data items in the set can be processed independently of each other. This cannot be possible in all situations, in such cases synchronization and semaphores are used to ensure concurrency.

5.6.2 Building Ranges

The algorithm for building Ranges out of Results was presented in the Design Chapter. However, it was not described how it can is checked whether or not a Result can be added to an existing Range. When a Result is attempted to be added to a pre-existing Range, a **add()** (see Figure 5.8) method is called. In this method, it is initally checked whether the Result should be added to this Range. This check involves looking at the dates of the Result and checking whether they are fully or partially encapsulated by the Range's dates. If this is not the case, then the attempt of adding the Result fails. If the Result's dates are encapsulated by the Range then it will be checked whether the Result can be added to any of the nodes of the Range tree.

This would occur if there is any node in the tree that holds the same exact dates as the Result. If this is the case then the Result will be added to that node. If this is not the case, then it can be determined that the Result belongs to this Range, however its position in the tree needs to be created. It can be the case that there is a Range that partially encapsulates the Result, if so the Range would need to be expanded. In both cases, the tree needs to be traversed to find the optimal position for the Result.

```
public boolean add(Result result) {
    TimelineDate timelineDate = result.getTimelineDate();
    //check constraints
    if (!shouldAdd(timelineDate)) {
   //check constraints if we can even add to this range
        return false;
    }
    //attempt to add through an existing range
    Range toAdd = checkCanAdd(result);
    if (toAdd != null) {
   //add to the results of the given range
        toAdd.results.add(result);
        return true;
    }
    //now we try to extend the range
    return createRangeAndAdd(result);
}
```

Figure 5.8: Implementation of Adding Results to a Range

5.6.3 Saving Results

Saving the results of the processed documents, is divided into two parts: saving them as a PDF, or as a JSON.

In the PDF, the file that is saved, is a graphical representation of the events, using the traditional timeline view. It is aided through the use of the Apache PDFBox and Commons library. The libraries work like a painting tool. The pages are canvases, where lines and text can be drawn on at specific positions. These positions are given by coordinates (x, y), where the top left of the page is the origin, i.e. $(0, 0)$, and the x values increment towards the right

of the page, and the y values towards the bottom of the page. To build a dynamic system, that can create a pdf for any number of Results, certain constraints are required. For example the maximum number of events to be displayed on each page,and the size of the text of the summaries and subjects (i.e. how many characters). Due to tool limits, it is not possible to continue on the next page of the PDF document, hence events cannot overflow onto the next page. However, since the text that is used in the summary should be short, assumptions can be made. For example, the maximum size of a container that holds an event. Knowing this, it allows the drawing of containers of maximum size on the document. Thus, ensuring that all the data for each event will fit in a container. Therefore, the system can be considered as drawing containers on the page and filling them with their event data. This is repeated for all the events, and where necessary creating a new PDF page to fill more events. Hence the task, has been broken down to moving the x and y positions, writting text (the event data), and drawing the rectangle (container). Since in the traditional view, the layout of the events are one on the left of the page and the other on the right of the page, the tasks have to be changed minimally to allow for this. In Figure 5.9, the implementation, to draw an event on the right of the page, is provided.

```
private void drawOddEvent(Result result, PDPageContentStream contentStream, int position)
    throws IOException {
  //initially y is the top right where this needs to
  //be shown, x starts from the middle
  currentY -= padding; //add some padding to y
  currentX = (int) widthOfPage / 2;
  contentStream.moveTo(currentX, currentY);
  //write the text for the Event
  int lengthOfHorLine = (int) ((widthOfPage / 2) - (padding + widthOfRectangle));
  currentX += lengthOfHorLine;
  writeText(result, contentStream, currentX, position);
  //draw the rectangle to surround the text
  drawRectangle(contentStream, currentX, currentY - heightOfRectangle);
  //draw the horizontal line connecting event timeline
  currentY -= heightOfRectangle / 2;
  contentStream.moveTo(currentX, currentY);
  contentStream.lineTo(widthOfPage / 2, currentY);
  contentStream.stroke();
  currentY -= (heightOfRectangle / 2) + padding;
}
```

Figure 5.9: Implementation of drawing for a Result in the PDF timeline

The JSON implementation, is aided by the use of the Google GSON library. JSON provides a clearly defined key-value data format that can be interpreted by any system. The advantage of this library is that it allows for a flexible creation of JSONs. Specifically, it allows the developer to specify how an object of a given type should be serialized (see Figure 5.11), i.e. how its data should be extracted into JSON format. In this case, the objects to be serialized are Result objects. The data extracted from a Result includes its start and end date, its list of subjects, its summary, and its file data. This is done through an adapter. The adapter defines how the data of the Result object is extracted, and how a JSON Object is created. For a list of Results, this will be carried out for each Result, and thereby, a resulting list of JsonObjects will be produced, i.e. a JsonArray. This can then be saved by the user in a file, and can then be interpreted by any third-party system. The format of a JSON of a Result is given in Figure 5.10. Where G, in the dates, is the ERA (i.e. BC or AD) of the date.

```
{
"date1":dd-MM-yyyy G,
"date2":dd-MM-yyyy G,
"subjects":[],
"event":String,
"from":{
   "filename":String,
   "baseDate":dd-MM-yyyy
   }
}
```

Figure 5.10: Structure of Result (event) JSON

```
public JsonElement serialize(Result src, Type typeOfSrc, JsonSerializationContext context)
    {
   //json object for this result
      JsonObject jsonObject = new JsonObject();
   //adding the range dates (which can be null)
   /* whenever a value is null, the key-pair will not be included in the final JSON */
   jsonObject.addProperty("date1", src.getTimelineDate().getDate1FormattedDayMonthYear());
   jsonObject.addProperty("date2", src.getTimelineDate().getDate2FormattedDayMonthYear());
   //adding the subjects
   JsonArray subjectJsonArray = new JsonArray();
   for (String subject : src.getSubjects()) {
      subjectJsonArray.add(subject);
   }
   jsonObject.add("subjects", subjectJsonArray);
   //adding the event
   jsonObject.addProperty("event", src.getEvent());
   /*adding the file data (excluding the path, since this can be used on other system
       where files are elsewhere)*/
   JsonObject fromJsonObject = new JsonObject();
   FileData fileData = src.getFileData();
   if (fileData != null) {
      fromJsonObject.addProperty("filename", fileData.getFileName());
      fromJsonObject.addProperty("baseDate",
          fileData.getCreationDateFormattedDayMonthYear());
   }
   jsonObject.add("from", fromJsonObject);
   return jsonObject;
}
```


Figure 5.11: Implementation of the Serializer used in the Adapter to generate JSON for a list of Results

-main algorithms: for building events, building ranges, pdf save, json save, processing files

Chapter 6

Professional and Ethical Issues

Either in a seperate section or throughout the report demonstrate that you are aware of the **Code of Conduct & Code of Good Practice** issued by the British Computer Society and have applied their principles, where appropriate, as you carried out your project. -no user data is collected (public interest) -project will be open source so the licenses applied can be used, however commerical would require a specific license -how dealt with ethical approval? (newspapers used, etc.) -in analysis kept testers anonymous -if used on cloud and use innapropriate setting smay use a lot of resources

During the development and evaluation of the system, great care has been taken to follow the Code of Conduct[1] and Code of Good Practice[2] issued by the British Computer Society. This must be done to avoid serious legal and ethical problems.

Importance has been given to state explicitly which software libraries and academic papers have been used throughout the development. The information and services of these have been used and modified to meet the requirements of the project. All the software produced consists of my own work, except where it is explicitly said otherwise.

As the system is intended to be released as open-source, it has been confirmed that the libraries allow this. If the system were to be used commercially, a license would have to be produced that would encapsulate the project and its libraries.

The developed system does not collect user data, thereby abiding to the public interest of the Code of Conduct. In addition, for the evaluation phase it was checked whether ethical approval would be required. It is not. Since the test participants are kept completely anonymous, and the data sources used, are publicly available.

[1] http://www.bcs.org/category/6030
[2] http://www.bcs.org/upload/pdf/cop.pdf

It should be noted that if the system is moved onto a server to produce timelines from large sets of documents, the processing power used would increase substantially, and may affect other systems running on the same machine. This can be avoided by modifying the settings of the system, e.g. the number of threads used, to reduce the number of processes running.

Chapter 7

Evaluation

-present how did analysis, why?, other options (relate to work) -results of analysis -conclusion

As presented in the Design chapter, the objectives of the system are: visibility, efficiency, and effectiveness. To evaluate each of these, the chapter is split into three sections. Each addressing an objective.

7.1 Visibility

-explain expert heuristic -knowledge of expert -that this is the least important of the three as the ui can be built by third party and only processing is used.

To evaluate the visibility of the system, an expert heuristic using the Nielsen Usability Heuristic [7] set was used. This involves an expert evaluating the UI of the system with a set of principles, which have been tested, and then discussing the system for each principle. The evaluation can be formal or informal. The former was choosen, as it was carried out at the end of the development. Informal evaluations are done when analysis needs to be done during the iterations of design, since it provides feedback quicker and allows designers to discuss the problems when producing the next design. Formal evaluation is done, usually, at the end of the development cycle when the system is being evaluated. In this case, the focus is on evaluating visibility. Heuristics do not only focus on visibility, since some of the principles focus on error prevention. However, these are also important for a system that is to be simple to use.

Other heuristic sets exist, however the Nielsen set is the most known set[1]. Another prominent design evaluation is cognitive walkthroughs. They consist of giving experts, scenarios to

[1] https://www.usability.gov/how-to-and-tools/methods/heuristic-evaluation.html

go through, and during their progression of the scenarios, they answer three questions related to design issues (cite).

Due to time constraint, only one in-depth expert heuristic could be carried out. Normally, there would be multiple experts, each carrying out their own evaluation, and then they would discuss between themselves to produce a single document detailing the usability problems. In this case, the expert has experience in applying the principles from previous projects. Note that they do have technical knowledge in Computer Science, hence they are not considering the system from a novice users perspective, but rather in identifying where the system satisfies the principles. Experts may be biased in how they believe a design should be, however this is expeced as designs are a subjective matter.

The data collected from an expert review is qualitative data. Qualitative data is statements, observations and subjective judgements. In this case, they are issues in the system with possible solutions, which aim to satisfy the principles they violate. It is different to the quantitative (numerical) data collected in the Effectiveness section.

The main principles evaulated are visibility, match with the real world, consistency and error prevention. With a main focus on visibility..

For the visibility principle, where the focus is on the user knowing the systems state, the design succeeds according to the expert. The user is aware of the actions available to them as the system progresses through its states of processing files. This begins from the system only allowing the user to load documents on startup, to the loading dialog when the documents are being processed, and to the subsequent viewing of the produced timeline. To ensure error prevention, where data validation is done on user input, red-borders are shown in the input fields when the data is invalid and the options to save the data are disabled. This allows the user to determine that a mistake was made with the input, and specifies where the mistake occurred. To ensure the user knows what the expected input format is, the input fields are aided with hints. For example, the format of dates is shown when the user must enter the base date for a document, or a character limit is shown when the user is editing the summary of an event. Clearly marked exit, "Cancel" buttons, are provided to allow the user to not commit their changes. Confirmation dialogs are also used to prompt the user to confirm an action that may be costly, e.g. deleting the subjects of an event, or deleting the event itself. All the previously mentioned design choices, accompanied with the non-technical language used in the system, ensure that the system should be accessible to any user.

To improve visibility, the expert suggested improvements. For example, hinting the user on

how to delete subjects, or providing an indicator for the zooming in/put function in the "Range View". To aid the user recognition of the function of buttons, it was suggested to use icons that visually describe the function. The expert suggested filling up a progress bar, instead of the circular progress bar in the system, when documents are processed. However, this is not a clear indication of the progress as some documents require more time to process than others, but in the progress bar they would have equal impact.

From this evaluation it can be determined that the system fulfilled its aim of producing a system that is visible to the user. The user is informed of the states the system is in as it processes documents. Clearly marked exists, and error prevention with indicators of what the error is, are provided. Note that visibility is not as important as the other objectives, as the system could be used as a library (directly, or indirectly through JSON outputs) where another developer produces their own UI. However, in the case that the system is used by a user (not a developer), the UI produced is accessible for them, and also provides useful information.

7.2 Efficiency

Efficiency relates to the time it takes for the system to produce an answer. A system is efficient if it produces a result in an appropriate amount of time based on its input.

A way to evaluate this is to run the system on different inputs and record the time taken to produce a timeline. However, the time recorded is specific not only to the machine that the system is being run on, but also how the CPU scheduled the work at that point in time (i.e. how much other work was being carried out on the machine). If the system is ran with other tasks running in the background and the CPU gives the application less priority, i.e. other tasks run before this one, then the system will require more time to produce a result. Note that depending on the hardware used when running the system, results may be produced quicker or slower (cite). In addition, not all input sizes can be tested, as the document set size can be infinite. The solution to this is to consider the time complexity of the main algorithms used in the system.

The time complexity of an algorithm is, the number of operations, and time required to produce an output for an input of a given size (cite). This allows to determine how well the system scales with larger inputs, by only considering how many operations are performed as a function of the input.

Time complexity focuses on the worst case scenario, and does not consider low-order operations. For example, if for an input n an algorithm performs $3n + 2$ operations, the low-order

term 2 and the co-efficient 3 are omitted. This would result in a time complexity $O(n)$ (where O is called big-Oh). In general the highest order-term is taken, and all the rest are omitted. For example, $4n^2 + 10n + 2$ would have time complexity $O(n^2)$. Low-order terms are omitted since with a very large input, or as $n \to \infty$ (n tends to infinity), the time required for the low-oorder operations becomes irrelevant. The co-efficient of the high-order term is omitted, as a machine can be 4 times faster, or 4 times slower, so it does not affect, in general, the number of operations performed.

To measure the efficiency of the system, the time complexity of the front-end (graphical part) and back-end (logical part) of the system have been computed separately, as the system can be used standalone, or as a library in other systems that produce their own UI. The complexity of each algorithm is presented in the table below (see Figure 7.1).

Algorithm	Information	Time Complexity
Document Processing	n - number of documents s - number of sentences w - number of words	$O(nsw^2)$
Range Production	n - number of Results	$O(n^2)$
Range Timeline View	n - number of Results	$O(n)$
Traditional Timeline View	n - number of Results	$O(n)$

Figure 7.1: Time complexity of main Algorithms in System

7.2.1 Back-End

Processing Documents

The two main algorithms are processing of documents, and the production of Ranges. The complexity of each is presented and discussed. It is assumed that the time complexity of the StanfordCoreNLP to annotate a document is $O(w)$ (for all w words in the document), and that the documents are annotated before being processed.

The algorithm used was previously presented (see Algorithm 3). For a list of n documents, each is processed. The algorithm can process at most x documents in parallel at any time. Depending on the users settings value x can be 1 or ∞ (infinity). If ∞, then it can be determined that the complexity of the algorithm is given by the complexity of processing the largest document, since all documents are being processed in parallel. As all documents are processed in parallel.

When a document is processed, each sentence is checked for a temporal expression, before a summary or any other processing is done. In the worst case, all sentences s in a document

have to be fully processed. In such a case, the dates, the subjects, and the summary need to be produced. These are all done after each other, so it can be determined that the computation complexity of processing one sentence is given by $max(getDate, getSubjects, getSummary)$. Where max will return the greates time complexity of the three operations.

To get the date of a sentence, with w words, each temporal expression needs to be processed. The processing of a temporal expression is linear (see the getDate implementation in Figure 5.1), as it does not depend on the input directly. It performs at most 3 loops to produce an exact date. Thus processing an NER date has a complexity of $O(1)$. In the worst case, every word in the sentence is a temporal expression (which does not happen normally in a sentence of well-written documents, but can still occur). Thereby, the time complexity for getDate is $O(w)$.

To select the subjects of a sentence, the NER(footnote) annotator is used to determine which words are of interest. In the worst case, all words can be of interest. When a word is identified it is placed in a list, for the subjects of that event. Placing a word in a list has time complexity $O(1)$, hence it being done for w words, in a sentence, leads to a complexity of $O(w)$.

To create the summary of a sentence, with w words, the hedge-trimmer algorithm is used (see the Algorithm presented in the Design Chapter). In the paper, the time complexity of the algorithm is not presented as the algorithm is not explicitly provided, but rather explained. Thereby, the algorithm that was implemented is analyzed. In the grammatical tree produced, each word is a leaf. The structure of the tree varies, however for this evaluation it is assumed that in the worst case a full-binary tree is produced. This may not be the case, as the trees produce vary in format, however it is required to proceed with the evaluation. If there are w leaves in the tree (i.e. each word in the sentence is a leaf), then there are $2w - 1$ nodes in the tree. The rules of the algorithm are applied one after the other. Thereby, the time complexity of the algorithm is given by the rule with the highest time complexity. The first two rules traverse the tree once, and have a complexity of $O(2w - 1) = O(w)$. In the last step, where the tree is iteratively shortened until it is below the threshold. In the worst case, the threshold can be 0 (not explicitly allowed in the system as the minimum value is 10). Note that the threshold value is met when the number of words in the summary falls below it, and that the size of the summary is given by the number of leaves in the current grammatical tree. At each step, at least one node needs to be removed for the algorithm to continue until the threshold is met. Note that if an inner node is removed, its children are also removed. Thus, the leafs are reduced, thereby getting closer to the threshold value. To identify the node to remove, the

tree must be traversed, and as presented earlier, this has complexity $O(w)$. In the worst case, only one leaf is removed each time. Thus the tree is traversed w times. Therefore the time complexity of the last rule is given by $O(w^2)$.

Therefore, the processing of a sentence with w words is given by $O(w)+O(w)+O(w^2)$, where each time complexity corresponds to a task (infering dates, getting subjects, and producing a summary). Thereby, the time complexity of processing a sentence is $O(w^2)$, as the low order terms are removed.

Assuming that the set of n documents is processed one after the other (i.e. at most 1 document can be produced in parallel, in the worst case), and that all the documents have s sentences (or less). Where each sentence has w words (or less). The running time of processing one document is $O(sw^2)$. Thus the running time of processing n documents is $O(nsw^2)$. Where it is not known which of n, s, or w is the high-order term. The time complexity of annotating a document is omitted as it can be considered that $sw^2 > w$.

If the input consists of many documents, with few short sentences (i.e. w and s are smaller than n), then the running time is given by $O(n)$. Which for processing n documents, is efficient since the system scales linearly with the input. However, this is not always the case, as it may be that there are many short sentences (i.e. s is greater than n and w, thus the time complexity is $O(s)$), or there are few long sentences (i.e. w is greater than n and s, thus the time complexity is $O(w)$).

Range Production

Ranges are produced with the algorithm presented in the Design Chapter. Based on an input of n Results, they have to be sorted, added to existing Range trees or else creating new Range trees, and then, finally, the trees must be sorted.

Sorting Results in Java has a complexity of $O(n \log n)$. This is due to Java using merge-sort when comparing the results (footnote). Merge-sort consits of recursively breaking down the problem space in half (which produces the $\log n$ part), and then building the sequence back up (which produces the n). However, it should be noted that with sequences that are almost sorted Java 8+, due to the use of TimSort[2], the time complexity is closer to$O(n)$.

Each Result needs to be added to a tree. Before the tree is traversed to find a location, a check is performed to determine whether it needs to be added to the tree. In the worst case, the Result cannot be added to any tree, thus all the trees are checked and a new Range tree needs

[2] https://bugs.openjdk.java.net/browse/JDK-6804124

to be amde to hold the Result. This can be the case when the dates of events are disjoint. In such cases, the amount of Trees that need to be checked increase by 1 for each Result added. Assuming that the time complexity to check whether a Result can be added to a tree is $O(1)$ (since the check is in constant time), then the time complexity is given by the sum of 0 to n. As in the first step, no Ranges exist so nothing needs to be checked. Then one tree needs to be checked, then two trees, and so on. Where at the nth Result, $n-1$ trees need to be checked, each with a complexity $O(1)$. Thereby, the total complexity is given by the sum of 0 to n, which is given by $n(n+1)/2$. This produces a time complexity $O(n^2 + n/2)$, which is $O(n^2)$.

After all the trees have been produced, the Ranges need to be sorted. In the worst case there is one full expanded tree. Which is a tree where for each Result, it had to be expanded, because the dates overlapped (but where never fully contained within each other). This leads to a full binary tree as when a Range is expanded it has two children set to it: one being a new Range holding the Result being added, and the other, the previous subtree that the Result's dates partially overlapped with. Which using the Java sorting algorithm leads to $O(n \log n)$, as the low-order terms are dismissed.

Thereby, the time complexity of the Range Production algorithm is given by the operation with the greatest complexity, which is adding Results to the tree. Thus, the time complexity of the algorithm is given by $O(n^2)$.

7.2.2 Front-End

In this section the focus is on the creation of the range and traditional timeline views.

Range View

As can be seen by the algorithm to produce the Range view (see the Figure 5 in the Implementation Chapter), each Range is considered separately when it is built. In the worst case, there is one Range with a fully expanded tree (i.e. a full binary Range tree). If the tree holds n results, which are held at the leaves, then it has a total of $2n-1$ nodes. For all nodes it must produce a layout, which includes the production of its children recursively. Therefore, $2n-1$ layouts are produced, so the time complexity is given by $O(n)$.

There can be performance issues, as this layout embeds layouts within each other and many views are held in memory at a time. However, the focus in this section is to consider the time for the system to produce the UI, not how heavy it is for the system's memory.

Timeline UI

The creation of the timeline view is trivial. For a list of n results, the layout for each row of the ListView is produced. Hence, n rows are produced. This leads to a time complexity of $O(n)$.

Note that the ListView used in the system does not hold all rows in memory at once. The layout of a row is only produced if it needs to be shown, i.e. that part of the ListView is visible to the user (cite). Hence, the time complexity to produce the timeline view is less than $O(n)$, as the system does not show all the Results. Only when the size of the ListView is smaller than the amount of allocated screen space it is given, will it have to produce n rows as all rows can be shown, and thus will have a time complexity of $O(n)$.

7.3 Effectiveness

-how did you test? -participants data? -safeguard participants -data set -why test this way

Effectiveness of a system is how correct it is at producing results, in this case timelines. To determine whether a timeline is correct, experimental testing was done. This allows to determine whether the produced timeline is similar to ones produced by people.

7.3.1 Testing Preparations

In order to test the effectiveness of the timelines, test participants were asked to build manual timelines given by a document. These were then compared to the produced timeline by the system for the same document.

Note that there is no one perfect timeline, but instead there can be many good timelines that express the documents. This is demonstrated by the different timelines produced by the test participants for the same document. The aim is to compare the produced timelines to the manual timelines, and see what are the differences and similarities.

Participants were kept anonymous during the entire process, and will be kept anonymous during the presentation of data. The participants came from different academic backgrounds, were of different ages, and had different levels of exposure to the document data sets. A clear limitation to the experiment was the availability of participants. As such, limited data was gathered.

The data sets consisted of publicly available newspaper articles of different domains: politics, criminal cases, and short bibliographys. To ensure that a substantial amount of data could be gathered, articles that were rich in temporal expressions were picked. This allows for more

timeline data points to be compared between a manual timeline and an automated produced timeline.

Using publicly available data sets (e.g. newspapers), and keeping the participants anonymous complies with the guidelines of the BCS, and does not require an ethical approval for the experiment.

7.3.2 Testing

Participants were provided a document, and were asked to select events line-by-line, like the system. This is to allow comparison between the two timelines.

There was no time limit for the participants, as the aim was for them to produce correct timelines instead of producing timelines quickly, since the system has a clear advantage in that regard.

The manual events produced by the participants, included the date of the event and a short summary of what occurred. In situations were the dates cannot be determined, participants would link the events to others by saying it occurred before/after another event. Currently, the system cannot establish these links.

The manual and automated produced timelines are found in the Appendix.

7.3.3 Analysis

The comparison of the manual timelines consisted in: the number of events collected, the percentage of matched and unmatched sentences, the percentage of mismatched exact dates, and the similarity in summaries.

The number of events allows to determine what is the expected amount of events for a given document. This can be combined with the other data to determine if the timeline produced is to broad in its event generation, or too specific.

The matching and unmatching of sentences, consists of identifying which sentences the users used in their generation of events and which the system used. This allows to evaluate how effective the system is in identifying an event, independently of whether the data of the generated event is correct.

Mismatched dates consists of determining how accurate the system is in producing exact dates for events. It is expected that the system will perform poorly in situations were the date of an event cannot be identified, but only that it occurred before/after another event. For test participants this is not a problem, as they understand context and can apply their own thinking

to determine whether an event occurred before or after another event, even when there is no temporal data to support this. To determine the quantity of mismatched dates, events from the manual and produced timeline are only considered if they originate from the same sentence in the document. Then the dates of the two events are compared.

The similarities in summaries determines how effective the system is in conveying the meaning of the original sentence that produced it. The events are grouped by the sentence that produced them, and then it is checked whether the summary and key words of the event, from the produced timeline, matches the meaning of the summary of the event from the manual timeline. This is generally done by checking that they have the same keywords.

The results of the following are presented in the table below (see Figure 7.2). The name of the articles used are provided, along with their source and their publishing date.

Note that the percentage of events originating from the same sentence focuses on how many events in the manual timeline were also identified in the automated timeline. This allows to determine how many of the events in the manual timeline are not identified by the automated system.

No stastical analysis could be performed to determine with certainty how accurate the system is, as there is not enough data. However, it can be concluded, that from the data gathered, that the system identifies more than 75% of the events in the documents, if the manual timelines identify all events. Events with completely unknown dates (i.e. there are no explicit temporal expressions) are not identified. The summary generation is not optimal, as just above 65% of the automated timelines capture the meaning of the manual timelines. Thus, showing the weakness of the decision-based summary algorithms, and reasons as to why the domain-dependent stastic models are more prominent. An effective point of the system was the production of exact dates from temporal expressions. For the automated inference of dates, the majority of inferred dates match the dates determined by the test participants.

Therefore, the system does not produce perfect summaries that convey the same meaning as the original sentence. However, the system is effective in the production of exact dates from temporal expressions, and does perform fairly well in identifying events.

Comparing System to Manual Timelines From...	Participant 1	Participant 2	Participant 3
Difference In collected events (+/-) (19 events collected by system)	+5	-1	-3
Percentage (%) of events originating from the same sentence	66.7	83.3	87.5
Percentage (%) of events not identified	33.3	16.7	12.5
Percentage (%) of mismatched dates	0.0	6.7	0.0
Percentage (%) of similar summaries (matching keywords)	75.0	73.3	78.6
Unkown Dates of Manual Timelines From...	4	2	2

(a) BBC: US elections 2016: Donald Trump Life Story (Published 20th of January 2017)

Comparing System to Manual Timelines From...	Participant 1	Participant 2	Participant 3
Difference In collected events (+/-) (4 events collected by system)	+5	0	+3
Percentage (%) of events originating from the same sentence	44.4	100.0	57.1
Percentage (%) of events not identified	55.6	0.0	42.9
Percentage (%) of mismatched dates	0.0	0.0	0.0
Percentage (%) of similar summaries (matching keywords)	50.0	50.0	50.0
Unkown Dates of Manual Timelines From...	0	0	-1

(b) Guardian: British Toddle Abducted, Police believe (Published 5th of May 2007)

Comparing System to Manual Timelines From...	Participant 1	Participant 2	Participant 3
Difference In collected events (+/-) (6 events collected by system)	+2	+1	+1
Percentage (%) of events originating from the same sentence	75.0	88.5	71.4
Percentage (%) of events not identified	25.0	12.5	28.6
Percentage (%) of mismatched dates (on matched events)	33.3	16.7	0.0
Percentage (%) of similar summaries (matching keywords)	66.7	66.7	80.0
Unkown Dates of Manual Timelines From...	1	0	1

(c) Guardian: Madeleine McCann detectives arrive in Portugal to question 11 suspects (Published 9th of December 2014)

Figure 7.2: Manual vs Automated Timelines Comparison

Chapter 8

Conclusion and Future Work

The project's conclusions should list the key things that have been learnt as a consequence of engaging in your project work. For example, "The use of overloading in C++ provides a very elegant mechanism for transparent parallelisation of sequential programs", or "The overheads of linear-time n-body algorithms makes them computationally less efficient than $O(n \log n)$ algorithms for systems with less than 100000 particles". Avoid tedious personal reflections like "I learned a lot about C++ programming...", or "Simulating colliding galaxies can be real fun...". It is common to finish the report by listing ways in which the project can be taken further. This might, for example, be a plan for turning a piece of software or hardware into a marketable product, or a set of ideas for possibly turning your project into an MPhil or PhD.

-what have you learned? -how can the project be carried further (neural net for summary, building on the StanfordCoreNLP for detas depending on others)

8.1 Conclusion of Project

In conclusion, the project aimed to build a system that extracts events autonomously from documents, and then produces a timeline with the events. The StanfordCoreNLP tool was the major library used to categories sets of words into predefined categories (NER annotator), and build the grammatical trees (POS annotator). A trimming algorithm was used to produce summaries of sentences through headline generation. The use of multi-threading allowed for parallel processing of documents, and thus faster times to produce the event data and populate the timelines.

Open-source will allow the project to be further developed, and included in other event

extraction systems. Since the code produced is fully documented, and the back-end of the system is separate of its graphical counterpart, the system can be integrated as a library in other projects. The intermediate produced JSON allows the Results of the system to be used in third-party applications. Through the use of Gradle, and a produced wrapper, portability is achieved as the required libraries are retrieved and the system can be ran with one command[1].

Issues occurred, and new requirements appeared during the production of the system. These include the creation of exact dates from normalized dates, wrong input documents, and the encapsulation of events for a new timeline view. These issues were identified and appropriately solved. Ofcourse, as with any project, the initial design was changed, but through Agile methodoly this could be tackled and did not hinder the implementation of the system. Changes include minor alterations in the systems architecture to allow for the independence between the logic of the system and its graphical representation.

The use of running time complexity allowed the applications efficiency to be evaulated independently of the system it is run on, and on the data size given as input. Using an expert heuristic allowed the produced UI to be evaluated for visibility. Since the heuristic is a standard, it has been thoroughly tested before-hand by others. From the evaluation of the effectiveness of the system, it can be determined that the system was effetive in identifying the events in the documentes used (that consisted of different domains). However, the summary module of the system requires further work. It is not possible to say with certainty how effective the system is in general, as not enough data is available for such analysis. However, with more time this can be done. In the following section, possible future developments of the system are discussed.

8.2 Future Work

-neural bet, cloud, building on tool to link to ambigious dates that are related to other possible known ones, edit events (machine-learning in addition to neural net)

Future works consist of areas of possible development that would improve the overall effectiveness of the system. The areas are described below. Implementing any of these would be an improvement, however the challenge is in being able to combine them all to produce a system that can be used at a commercial level, and would be unrivalled.

Neural Net - As mentioned in the Background Chapter, there is a heavy use of Neural Nets for text summarization, as can be seen from the works of [2] and [9]. These are often combined with noisy-channel models that use data sets for statistcal computation of summaries. The

[1] gradlew run

main benefit of such a system would be to provide better summaries. The reason for them not being used in the project is the large data set required and the amount of computation done to produce a summary of one sentence.

Machine Learning - Machine learning allows a system to become more accurate by incrementing its data set, which is used to produce an output (cite). In this system it could be used in the production of events. For example, when a user edits an event it can be assummed that they produced a corrected event. This data can then be used and considered in the creation of other events by the system, to produce more precise ones.

Cloud - A cloud allows a service to be provided independent of its software and hardware (cite). This is done by deploying the system on a remote server, and giving clients an interface to interact with it. Thereby it is not required for the client to have a powerful machine, and it does not affect their systems performance. The downfall is the cost. However, the performance could be improved as specific hardware could be used to improve the efficiency and it could be combined with Machine-Learning and Neural Nets to allow the system to impove in its event identification and production.

Extending StanfordCoreNLP - An issue in the tool used, is that it does not attempt to link ambigious temporal expressions, i.e. it may be known that an event occurred before another. For example, a person has to be born first before they can work (example taken from [6]). In the system when an exact date cannot be given, an ambigious date is produced. For example, for both born and working it could produce a "PAST_REF". This would then cause the system to assign both events to the same range of dates, i.e. 0001-01-01 up to the reference point used for the document. However, these can be made more precisely. Since a person has to be born first, and be over the age of 16 to work. Thereby the range of dates for the work event could be identified more precisely. The problem arises, from the tool and the produced system considering each sentence independently of each other. It would be benefitial if the events were to be linked one after the other, such that it may not be known when someone was born or when they worked, but that the event of them working is after the event of them being born. This would require building on the NLP tool used in the system.

References

[1] Sentence subjects. http://grammar.ccc.commnet.edu/GRAMMAr/subjects.htm. URL http://grammar.ccc.commnet.edu/GRAMMAr/subjects.htm. Accessed 4 Dec. 2016.

[2] S. Chopra, M. Auli, and A. Rush. Abstractive sentence summarization with attentive recurrent neural networks. In *Proceedings of NAACL*, 2016. URL http://nlp.seas.harvard.edu/papers/naacl16_summary.pdf.

[3] H. Daumé III and D. Marcu. Noisy-channel model for document compression. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 449–456, 2002. URL http://www.aclweb.org/anthology/P02-1057.

[4] B. Dorr, D. Zajic, and R. Schwartz. Hedge trimmer: A parse-and-trim approach to headline generation. In *Proceedings of the HLT-NAACL 03 on Text summarization Workshop-Volume 5 (ACL)*, pages 1–8, 2003.

[5] K. Knight and D. Marcu. Statistics-based summarization âĂŞ step one: Sentence compression. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 703–710, 2000. URL http://www.aaai.org/Papers/AAAI/2000/AAAI00-108.pdf.

[6] D. McClosky and C. Manning. Learning constraints for consistent timeline extraction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 873–883, 2012. URL http://nlp.stanford.edu/pubs/dmcc-emnlp-2012.pdf.

[7] J. Nielsen. 10 usability heuristics for user interface design. 1995. doi: https://www.nngroup.com/articles/ten-usability-heuristics/.

[8] A. Ritter, S. Clark, Mausam, and Etzioni. Named entity recognition in tweets: An experimental study. In *Proceedings of the 2011 Conference on Empirical Methods in Natural*

Language Processing, pages 1524–1534, 2011. URL https://aclweb.org/anthology/D/D11/D11-1141.pdf.

[9] A. Rush, S. Chopra, and J. Weston. A neural attention model for sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 379–389, 2015. URL http://www.aclweb.org/anthology/D15-1044.

Appendix A

Extra Information

A.1 Tables, proofs, graphs, test cases, ...

The appendices contain information that is peripheral to the main body of the report. Information typically included in the Appendix are things like tables, proofs, graphs, test cases or any other material that would break up the theme of the text if it appeared in the body of the report. It is necessary to include your source code listings in an appendix that is separate from the body of your written report (see the information on Program Listings below).

Appendix B

User Guide

B.1 Instructions

You must provide an adequate user guide for your software. The guide should provide easily understood instructions on how to use your software. A particularly useful approach is to treat the user guide as a walk-through of a typical session, or set of sessions, which collectively display all of the features of your package. Technical details of how the package works are rarely required. Keep the guide concise and simple. The extensive use of diagrams, illustrating the package in action, can often be particularly helpful. The user guide is sometimes included as a chapter in the main body of the report, but is often better included in an appendix to the main report. -how to set up (commands) -how to use given pieces of sample text -images

Appendix C

Source Code

C.1 Information

I verify that I am the sole author of the programs contained in this folder, except where explicitly stated to the contrary' - Oliver H'ohn, April 7, 2017.

C.2 Table of Contents

1. src/main/java/backend: Package for the logic of the system.
   (a) helpers: Package for utility code used in different components in the system.
      i. Sort.java: class that holds sorting methods used in the project.
      ii. ToJSON.java: class used to turn a list of Results into a JSON String. Uses the GSON library.
      iii. ToPDF.java: class used to save a list of Results into a PDF File. Uses the Apache PDFBox and Apache Commons libraries.
   (b) process: Package for modules to process documents.
      i. CallbackResult.java: interface for frontend classes that want to be notified when the files have been processed.
      ii. Engine.java: class to process the text in a document to generate Result objects. Uses the Stanford CoreNLP library.
      iii. FileData.java: class that represents data needed from a File.
      iv. ProcessFileCallback.java: interface to inform observer when a File has been processed.

v. ProcessFiles.java: class to process a list of Files (documents) and produce a set of Results. Uses the Apache PDFBox and Apache POI libraries.

vi. Result.java: class that represents the data for an event in the timeline.

vii. TimelineData: class that parses NER dates and holds start and end dates for an event. Uses the JODA Time library.

(c) ranges: Package for modules relating to placing Results in Ranges.

i. ProduceRanges.java: class to produce a list of Ranges (using the algorthm presented in the Design Chapter) from Results.

ii. Range.java: class that represents a node in a Tree, where dates encapsulate Results.

(d) system: Package for modules relating to the state and settings of the system.

i. BackEndSystem.java: class to hold the data needed by the entire Back-Entire (i.e. settings used, system state, etc.).

ii. Settings.java: class to represent the settings to be used in the system to process files, and build the UI. It saves the settings in a File. Uses the Stanford CoreNLP library.

iii. SystemState.java: enumeration of the different states the system can be in when a file is being processed.

2. src/main/java/frontend: Package for the UI of the system. Use the JavaFX libraries.

(a) controllers: Package of the view controllers of the system.

i. CustomResultRowController.java: controller for the custom Result rows in the ListView of the RangeData layout.

ii. CustomTimelineRow.java: class for the custom row in the Traditional Timeline.

iii. DocumentLoadedRowController.java: controller for each row in the Documents-Loaded ListView.

iv. DocumentReaderController.java: controller for the layout of the Document Reader window. Uses the RichTextFX library.

v. EditEventController.java: controller for the dialog used to edit events.

vi. ListViewController.java: controller for the layout where the ListView is shown.

vii. MenuBarControllerInter.java: interface for observers of a controller that has a menu bar.

viii. RangeDataController.java: controller for the layout that represents the data held by a Range.

ix. StartUpController.java: controller for the startup scene (i.e. the controller for the main window of the UI).

x. TimelineRowController.java: controller class for each row in the traditional Timeline ListView.

(b) dialogs: Package for the dialogs used in the system.

i. AboutDialog.java: class to create the dialog providing information of the system.

ii. EditEventDialog.java: class that creates a dialog to edit an event.

iii. FileConfirmationDialog.java: class that creates a dialog to edit data of the files selected to process.

iv. LoadingDialog.java: class used to show/hide the loading dialog when data is being processed.

v. RemoveConfirmationDialog.java: class to confirm the removal of a document from the timeline.

vi. SettingsDialog.java: class to show and edit the settings used in the system.

(c) helpers: Package for the helpers needed in the front-end of the system.

i. TextFieldState.java: enumeration class to represent the state the data in a TextField is in (i.e. is it validated)

(d) observers: Package for the Observer interfaces used to interact with the different layouts in the system.

i. DocumentReaderObserver.java: interface for the observer of the DocumentReaderController.

ii. DocumentsLoadedObserver.java: interface for the observer of the DocumentLoadedRowController.

iii. EditEventDialogObserver.java: interface for the observer of the EditEventDialog.

iv. MenuBarObserver.java: interface for the observer of layouts that have menu bars (and their items).

v. StartUpObserver.java: interface for the observer of the StartUpController.

vi. TimelineObserver.java: interface for the observer of the Timeline layout (includes both kinds of timelines).

vii. TimelineRowObserver.java: interface for the observer of the TimelineRowController.

(e) Main.java: main class that is ran to begin the system. It prepares the back-end and shows the UI.

3. src/main/resources: Resources used throughout the system.

(a) controllers: Package for the resource files used by the controllers of the system.

i. customErrorFields.css: CSS file for input fields.

ii. customResultRow.fxml: FXML file for layout of rows in the Result.

iii. documentLoadedRow.fxml: FXML file for layout of rows of the documents loaded.

iv. editEventDialog.fxml: FXML file for the layout of an edit event dialog.

v. icaddcircleoutline.png: PNG used to add subjects to an event.

vi. iccloseblack.png: PNG used to remove a document from the timeline.

vii. listViewTheme.css: CSS for the traditional timeline view.

viii. listViewThemeTimeline.css: CSS for the Range timeline view.

ix. rangeDataLayout.fxml: FXML file for the layout of a Range.

x. timelineRowEven.fxml: FXML file for the layout of an even row in the traditional timeline.

xi. timelineRowOdd.fxml: FXML file for the layout of an odd row in the traditional timeline.

(b) dialogs: Package for the resource files used by the dialogs of the system.

i. customErrorFields.css: CSS file for input fields in the dialogs.

ii. loadingDialog.css: CSS file for the loading dialog when processing files.

(c) documentListViewTheme.css: CSS file for the documents loaded ListView.

(d) listview.fxml: FXML file for the layout of the window with the timelines and documents loaded.

(e) startup.fxml: FXML file for the layout of the initial window shown when the system is launched.

4. src/test/backend: Unit Tests for the logic of the system. Uses the JUnit library.

(a) EngineTest.java: test class for the processing done by the Engine.java class.

(b) ProcessFileTest.java: test class for the work done by the ProcessFiles.java class.

(c) ProduceRangesTest.java: test class for the work done by the ProduceRanges.java class.

(d) SystemStateTest.java: test class for how the state of the system changes when it is processing documents.

(e) TimelineDateTest.java: test class for the parsing of the TimelineDate.java class.

(f) ToJSONTest.java: test class for the production of JSONs for a list of Results.

5. src/test/resources: Package for the test files used in the backend tests. Mostly used in ProcessFileTest.java, to test how well files are being processed.

(a) testfile1.txt: sample text to test the extraction and processing of .txt documents.

(b) testfile2.txt: sample text to test the extraction and processing of .txt documents.

(c) testfile3.txt: sample text to test the extraction and processing of .txt documents.

(d) testfile4.docx: sample text to test the extraction and processing of .docx documents.

(e) testfile5.pdf: sample text to test the extraction and processing of .pdf documents.

C.3 Instructions

Complete source code listings must be submitted as an appendix to the report. The project source codes are usually spread out over several files/units. You should try to help the reader to navigate through your source code by providing a "table of contents" (titles of these files/units and one line descriptions). The first page of the program listings folder must contain the following statement certifying the work as your own: "I verify that I am the sole author of the programs contained in this folder, except where explicitly stated to the contrary". Your (typed) signature and the date should follow this statement.

All work on programs must stop once the code is submitted to KEATS. You are required to keep safely several copies of this version of the program and you must use one of these copies in the project examination. Your examiners may ask to see the last-modified dates of your program files, and may ask you to demonstrate that the program files you use in the project examination are identical to the program files you have uploaded to KEATS. Any attempt to demonstrate code that is not included in your submitted source listings is an attempt to cheat; any such attempt will be reported to the KCL Misconduct Committee.

You may find it easier to firstly generate a PDF of your source code using a text editor and then merge it to the end of your report. There are many free tools available that allow you to merge PDF files.

C.4 src/main/java/backend

C.4.1 Helpers

Sort.java

```
package backend.helpers;


import backend.process.Result;


import java.util.Collections;
import java.util.Comparator;
import java.util.List;


/**
 * Class that holds the sorting methods used throughout the project.
 */
public class Sort {
    /**
     * Sort the given List by their Date1 (in their TimelineDate)
     *
     * @param resultList the given List.
     * @return the given List sorted in ascending order.
     */
    public static List<Result> sortByDate1(List<Result> resultList) {
        Collections.sort(resultList, new Comparator<Result>() {//will sort in ascending
             order
            @Override
            public int compare(Result o1, Result o2) {
                if (o1.getTimelineDate().getDate1() != null &&
                     o2.getTimelineDate().getDate1() != null) {
                    return o1.getTimelineDate().getDate1().compareTo(o2.getTimelineDate()
                        .getDate1());
```

```
            }
            if (o1.getTimelineDate().getDate1() == null) {
                return -1;
            }
            return 1;
        }
    });
    return resultList;
  }
}
```

ToJSON.java

```
package backend.helpers;

import backend.process.FileData;
import backend.process.Result;
import com.google.gson.*;

import java.lang.reflect.Type;
import java.util.List;

/**
 * Class used to turn a list of Results into a JSON String.
 */
public class ToJSON {
    /**
     * For the given List of Result objects, produce a JSON String of an array, where each
         index corresponds to one Result
     * in the list. Each Result is given by its range (date1, date2), its subjects (array
         of subjects), its event, and its
     * FileData (the filename, and base date used for processing the file), represented as
         a JsonObject from.
     * <p>
     * Whenever a value of data in a Result object is null, its corresponding key-pair will
         not be included in the final
```

```
     * JSON string. Such that, a completely empty Result object would be represented by:
         {subjects:[], event:"", from:{}}.
     *
     * @param results the given List of Result objects.
     * @return the JSON String representing the list of Result objects.
     */
    public static String toJSON(List<Result> results) {
        //sort the list in ascending order
        List<Result> sortedList = Sort.sortByDate1(results);
        String toReturn;
        final GsonBuilder gsonBuilder = new GsonBuilder();
        gsonBuilder.excludeFieldsWithoutExposeAnnotation();
        gsonBuilder.registerTypeAdapter(Result.class, new JsonSerializer<Result>() {
            @Override
            public JsonElement serialize(Result src, Type typeOfSrc,
                 JsonSerializationContext context) {
                JsonObject jsonObject = new JsonObject();
                //adding the range dates (which can be null)
                //whenever a value is null, the key-pair will not be included in the final
                    JSON
                jsonObject.addProperty("date1",
                    src.getTimelineDate().getDate1FormattedDayMonthYear());
                jsonObject.addProperty("date2",
                    src.getTimelineDate().getDate2FormattedDayMonthYear());
                //adding the subjects
                JsonArray subjectJsonArray = new JsonArray();
                for (String subject : src.getSubjects()) {
                    subjectJsonArray.add(subject);
                }
                jsonObject.add("subjects", subjectJsonArray);
                //adding the event
                jsonObject.addProperty("event", src.getEvent());
                //adding the file data (excluding the path, since this can be used on other
                    system where files are elsewhere)
                JsonObject fromJsonObject = new JsonObject();
                FileData fileData = src.getFileData();
```

```
            if (fileData != null) {
                fromJsonObject.addProperty("filename", fileData.getFileName());
                fromJsonObject.addProperty("baseDate",
                        fileData.getCreationDateFormattedDayMonthYear());
            }
            jsonObject.add("from", fromJsonObject);

            return jsonObject;
        }
    });
    //gsonBuilder.setPrettyPrinting();
    final Gson gson = gsonBuilder.create();
    toReturn = gson.toJson(sortedList);
    return toReturn;
  }
}
```

ToPDF.java

```
package backend.helpers;

import backend.process.Result;
import org.apache.commons.lang3.text.WordUtils;
import org.apache.pdfbox.pdmodel.PDDocument;
import org.apache.pdfbox.pdmodel.PDPage;
import org.apache.pdfbox.pdmodel.PDPageContentStream;
import org.apache.pdfbox.pdmodel.common.PDRectangle;
import org.apache.pdfbox.pdmodel.font.PDType1Font;

import java.awt.*;
import java.io.File;
import java.io.IOException;
import java.util.List;

/**
 * Class used to represent a list of Results in a PDF.
```

```
 */
public class ToPDF {
    private static int widthOfRectangle = 275;
    private static int heightOfRectangle = 150;
    private static int strokeWidthOfLine = 2;
    private static int padding = 7;
    private static int fontSize = 15;
    private static int maxNoOfCharacters = 40;//max number of characters per line in the
        event box
    private int currentX = 0;
    private int currentY = 0;
    private float widthOfPage;
    private float heightOfPage;
    private PDDocument pdDocument;
    private PDPage currentPage;
    private PDPageContentStream contentStream;

    /**
     * Constructor called to prepare for the creation of the PDF.
     */
    public ToPDF() {
        pdDocument = new PDDocument();
        try {
            reset();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * Will set all the global variables, and close any open streams.
     *
     * @throws IOException when closing a stream that could be open or closed.
     */
    private void reset() throws IOException {
        currentPage = new PDPage(PDRectangle.A4);
```

```
        pdDocument.addPage(currentPage);
        if (contentStream != null) {
            contentStream.close();
            contentStream = null;
        }
        contentStream = new PDPageContentStream(pdDocument, currentPage);
        contentStream.setStrokingColor(Color.black);
        contentStream.setLineWidth(strokeWidthOfLine);

        widthOfPage = currentPage.getMediaBox().getWidth();
        heightOfPage = currentPage.getMediaBox().getHeight();

        currentX = 0;
        currentY = (int) heightOfPage;

        contentStream.moveTo(widthOfPage / 2, heightOfPage);
        contentStream.lineTo(widthOfPage / 2, 0);
        contentStream.stroke();
    }



    /**
     * For the given Results, produce a PDF File that has all the given Results displayed
         in a timeline (like on the
     * Application but without the buttons).
     *
     * @param results the list of given Results.
     * @param file  the File where we are storing the timeline.
     * @throws IOException due to working with streams.(I.e. trying to read or remove a
         file that is already open).
     */
    public void saveToPDF(List<Result> results, File file) throws IOException {
        List<Result> sortedList = Sort.sortByDate1(results);//sort the list in ascending
            order

        int counterOfEvents = 0;
```

```
    for (int i = 0; i < sortedList.size(); i++) {
        if (counterOfEvents >= 5) {//start a new page
            reset();
            counterOfEvents = 0;
        }
        if (i \% 2 == 0) {//even
            drawEvenEvent(sortedList.get(i), contentStream, i);
        } else {//odd
            drawOddEvent(sortedList.get(i), contentStream, i);
        }
        counterOfEvents++;
    }
    if (contentStream != null) {
        contentStream.close();
    }
    pdDocument.save(file);
    pdDocument.close();
}

/**
 * Called to draw the event in the timeline for a Result that is at a Odd index in the
     list.
 *
 * @param result      the given Result object for which to draw this event.
 * @param contentStream the stream to which we are drawing.
 * @param position    the position of the Result in the timeline (to show in the event
     box)
 * @throws IOException due to working with streams.
 */
private void drawOddEvent(Result result, PDPageContentStream contentStream, int
     position) throws IOException {
    //initially y is the top right where this needs to be shown, x starts from the
         middle
    currentY -= padding; //add some padding to y
    currentX = (int) widthOfPage / 2;
    contentStream.moveTo(currentX, currentY);
```

```
        //write the text for the Event
        int lengthOfHorLine = (int) ((widthOfPage / 2) - (padding + widthOfRectangle));
        currentX += lengthOfHorLine;
        writeText(result, contentStream, currentX, position);
        //draw the rectangle to surround the text
        drawRectangle(contentStream, currentX, currentY - heightOfRectangle);
        //draw the horizontal line connecting event and timeline
        currentY -= heightOfRectangle / 2;
        contentStream.moveTo(currentX, currentY);
        contentStream.lineTo(widthOfPage / 2, currentY);
        contentStream.stroke();

        currentY -= (heightOfRectangle / 2) + padding;
    }

    /**
     * Called to draw the event in the timeline for a Result that is at a Even index in the
         list.
     *
     * @param result      the given Result object for which to draw this event.
     * @param contentStream the stream to which we are drawing.
     * @param position    the position of the Result in the timeline (to show in the event
         box)
     * @throws IOException due to working with streams.
     */
    private void drawEvenEvent(Result result, PDPageContentStream contentStream, int
         position) throws IOException {
        //initially x and y are top right in the page (0, pageHeight)
        //give some spacing between events
        currentY -= padding;//add some vertical small padding
        currentX = padding;
        contentStream.moveTo(currentX, currentY);
        //write the text for the Event
        writeText(result, contentStream, 0, position);
        //draw the rectangle to surround the text
        drawRectangle(contentStream, currentX, currentY - heightOfRectangle);
```

```
        //draw the horizontal line connecting event and timeline
        currentX += widthOfRectangle;
        currentY -= heightOfRectangle / 2;
        contentStream.moveTo(currentX, currentY);
        contentStream.lineTo(widthOfPage / 2, currentY);
        contentStream.stroke();

        currentY -= (heightOfRectangle / 2) + padding;//could set a string padding here

        //now connected the event (text surrounded by rectangle) to the timeline
    }

    /**
     * For the given Result and stream, write at the current x and y position (in new
         lines) the data held by the
     * Result object.
     *
     * @param result      the given Result.
     * @param contentStream the stream we are drawing to.
     * @param xOffset     an x-value offset from which the text is written from (starting
         at currentX)
     * @param position    the position of the Result in the timeline (to show in the event
         box)
     * @throws IOException due to using streams.
     */
    private void writeText(Result result, PDPageContentStream contentStream, int xOffset,
         int position) throws IOException {
        contentStream.beginText();
        contentStream.newLineAtOffset(2 * padding + xOffset, currentY - (fontSize +
            padding));//pad it horizontally and give vertical space for text
        contentStream.setFont(PDType1Font.TIMES_ROMAN, fontSize);
        contentStream.showText("Event #" + (position + 1));//show the position of the event
            (1st being the latest event)
        contentStream.newLineAtOffset(0, -(fontSize + padding));
        wrapText("Date: " + result.getTimelineDate(), contentStream);
        contentStream.newLineAtOffset(0, -(fontSize + padding));
```

```
        wrapText("Subjects: " + result.getSubjectsAsString(), contentStream);
        contentStream.newLineAtOffset(0, -(fontSize + padding));
        wrapText("Event: " + result.getEvent(), contentStream);
        contentStream.newLineAtOffset(0, -(fontSize + padding));
        wrapText("From: " + result.getFileData().getFileName() + " (" +
                result.getFileData().getCreationDateFormattedDayMonthYear() + ")",
                    contentStream);
        contentStream.endText();
    }

    /**
     * For the given text, check whether it needs to be wrapped around to fit the
         constraint of the max number of
     * characters in a sentence. If the text needs to be wrapped, write each sentence with
         a small space in between (no
     * padding, that's to separate data), if it does not need to be wrapped write the text
         as it is.
     *
     * @param text        the given text.
     * @param contentStream where the text needs to be written on.
     * @throws IOException due to writing to streams.
     */
    private void wrapText(String text, PDPageContentStream contentStream) throws
         IOException {
        if (text.length() > maxNoOfCharacters) {//we have text worth wrapping
            String[] wrappedText = WordUtils.wrap(text, maxNoOfCharacters).split("\\r?\\n");
            for (int i = 0; i < wrappedText.length; i++) {
                contentStream.showText(wrappedText[i]);
                if (i < wrappedText.length - 1) {
                    contentStream.newLineAtOffset(0, -(fontSize));
                }
            }
        } else {//no point doing the work of wrapping text, when its below the threshold
            contentStream.showText(text);
        }
    }
```

```
    /**
     * For the given bottom left x and y co-ordinates, draw a rectangle (its width and
         height are given by the static final
     * member values).
     *
     * @param contentStream where the Rectangle is being drawn to.
     * @param bottomLeftX the x-coordinate of the bottom left of the rectangle.
     * @param bottomLeftY the y-coordinate of the bottom left of the rectangle
     * @throws IOException
     */
    private void drawRectangle(PDPageContentStream contentStream, int bottomLeftX, int
        bottomLeftY) throws IOException {
        contentStream.addRect(bottomLeftX, bottomLeftY, widthOfRectangle,
            heightOfRectangle);
        contentStream.stroke();
    }


}
```

C.4.2 Process

CallbackResults.java

```
package backend.process;

import java.util.ArrayList;

/**
 * Interface for classes that call backend.process.ProcessFiles, so that when it finishes
     processing all Files passed in, the class (listener)
 * that called it can be informed.
 */
public interface CallbackResults {
    /**
```

```
    * Inform the Listener that all Files have been processed, and return the result of
        processing the Files.
    *
    * @param results the backend.process.Result objects obtained from processing all Files.
    * @param fileDataList the Data of the Files that Produced these results.
    */
   void gotResults(ArrayList<Result> results, ArrayList<FileData> fileDataList);
}
```

Engine.java

```
package backend.process;

import backend.system.BackEndSystem;
import edu.stanford.nlp.ling.CoreAnnotations;
import edu.stanford.nlp.ling.Label;
import edu.stanford.nlp.pipeline.Annotation;
import edu.stanford.nlp.pipeline.StanfordCoreNLP;
import edu.stanford.nlp.semgraph.SemanticGraph;
import edu.stanford.nlp.semgraph.SemanticGraphCoreAnnotations;
import edu.stanford.nlp.semgraph.SemanticGraphEdge;
import edu.stanford.nlp.trees.Tree;
import edu.stanford.nlp.trees.TreeCoreAnnotations;
import edu.stanford.nlp.util.CoreMap;
import edu.stanford.nlp.util.Pair;

import java.util.ArrayList;
import java.util.List;

/**
 * backend.process.Engine class that text in text as input and produces a list of
     backend.process.Result objects, which are events depicted in the text. An event is
     only picked out, if it has a date
 * associated with it.
 * <p>
```

```
 * Uses the algorithm proposed in: Bonnie Dorr, David Zajic and Richard Schwartz. Hedge
     Trimmer: A Parse-and-Trim Approach to Headline
 * Generation. Proceedings of the HLT-NAACL 03 on Text summarization Workshop-Volume 5.
     Association for
 * Computational Linguistics, pp. 1âĂŞ8.
 */
public class Engine {
    private static int threshold;
    private StanfordCoreNLP coreNLP;
    private String baseDate;

    /**
     * Set up the StanfordCoreNLP to analyze text.
     */
    public Engine() {
        coreNLP = BackEndSystem.getInstance().getCoreNLP();//can have it like before as the
            models will already be loaded, but this avoids having to check that
        threshold = BackEndSystem.getInstance().getSettings().getThresholdSummary();
        System.out.println("Using a threshold value of: " + threshold);
    }

    /**
     * Produces a list of Results based on the text passed in. (Determine events in the
         text).
     *
     * @param input The text for which we want to produce events for.
     * @param date The base date, from which we can determine exact dates from relative
         dates (eg Yesterday).
     * @return list of Results produced from events depicted in the text passed in, using
         the base date.
     */
    public ArrayList<Result> getResults(String input, String date) {
        ArrayList<Result> results = new ArrayList<>();
        baseDate = date;
        System.out.println("Base Date: " + baseDate);
        Annotation annotation;
```

```
        annotation = new Annotation(input);
        annotation.set(CoreAnnotations.DocDateAnnotation.class, date);//setting a reference
             so that when it finds a normalazied entity tag that isnt complete will
             determine it
        coreNLP.annotate(annotation);
        //coreNLP.prettyPrint(annotation, new PrintWriter(System.out));

        for (CoreMap sentence : annotation.get(CoreAnnotations.SentencesAnnotation.class)) {
            System.out.println(sentence);
            Result result = getResult(sentence);
            if (result != null) {
                results.add(result);
            }
        }
        return results;
    }

    /**
     * Returns a backend.process.Result object if the sentence contains a date, else
         returns null.
     * The backend.process.Result object will include a list of subjects as well as a
         summary (or the entire text, depending on the
     * length of text) of the sentence.
     *
     * @param sentence the CoreMap that contains the Sentence we want to produce a
         backend.process.Result for (if it has a date)
     * @return a backend.process.Result object if the sentence contained a Date; null
         instead.
     */
    private Result getResult(CoreMap sentence) {
        Result result = new Result();
        result.setOriginalString(sentence.toString());
        setDatesAndSubjectsNET(sentence, result);

        if (result.getDates().size() > 0) {//we have found dates, so lets find more
             subjects and the event of the sentence
```

```
        //setGrammaticalSubjects(sentence,result);//setting grammatical subjects in the
            result object
        setEvent(sentence, result);//set the summarized sentence as the event depicted
            in the sentence
        return result;
    }
    return null;
}

/**
 * Set the Dates and Subjects for the backend.process.Result based on Named-Entity Tags
     from the CoreMap passed in.
 * Dates have a DATE Named-Entity Tag. Subjects are LOCATIONs, ORGANIZATIONs, PERSONs,
     and MONEY.
 *
 * @param sentence a CoreMap that holds the sentence we want to extract data from.
 * @param result the backend.process.Result object that we are determining the data
     for.
 */
private void setDatesAndSubjectsNET(CoreMap sentence, Result result) {
    for (CoreMap mention : sentence.get(CoreAnnotations.MentionsAnnotation.class)) {
        String namedEntityTag =
            mention.get(CoreAnnotations.NamedEntityTagAnnotation.class);
        System.out.println(namedEntityTag + ": " +
            mention.get(CoreAnnotations.TextAnnotation.class));

        if (namedEntityTag.equals("DATE")) {
            //found a date for the result object
            String date_1 =
                mention.get(CoreAnnotations.NormalizedNamedEntityTagAnnotation.class);
            System.out.println("About to print time for the sentence: " + sentence);
            System.out.println("Normalized entity tag: " + date_1);
            String date = mention.get(CoreAnnotations.TextAnnotation.class);
            System.out.println("We are storing date: " + date);
            result.addDate(date);
            result.addDate_1(date_1, baseDate);
```

```
        } else if (namedEntityTag.equals("LOCATION") ||
             namedEntityTag.equals("ORGANIZATION") ||
                namedEntityTag.equals("PERSON") || namedEntityTag.equals("MONEY") ||
                     namedEntityTag.equals("MISC")) {
            //found a subject for the result object
            String subject = mention.get(CoreAnnotations.TextAnnotation.class);
            result.addSubject(subject);
        }
    }
}

/**
 * Finds the subjects of the sentence based on grammatical structure.
 * Gets the Basic Dependencies in the sentence and for each relation that contains the
     name "subj" (eg nsubj),
 * store its dependent in the backend.process.Result object, this is the grammatical
     subject of a sentence.
 *
 * @param sentence a CoreMap that holds the sentence we want to extract data from.
 * @param result the backend.process.Result object that we are determining the
     grammatical subject for.
 */
private void setGrammaticalSubjects(CoreMap sentence, Result result) {
    SemanticGraph semanticGraph =
         sentence.get(SemanticGraphCoreAnnotations.BasicDependenciesAnnotation.class);
    for (SemanticGraphEdge semanticGraphEdge : semanticGraph.edgeIterable()) {
        if (semanticGraphEdge.getRelation().getShortName().contains("subj")) {
            result.addSubject(semanticGraphEdge.getDependent().value());
        }
    }
}

/**
 * For the sentence passed in, summarize it and set it as the event of the
     backend.process.Result object.
```

```
     * <p>
     * Following the algorithm proposed in: Bonnie Dorr, David Zajic and Richard Schwartz.
         Hedge Trimmer: A
     * Parse-and-Trim Approach to Headline Generation. Proceedings of the HLT-NAACL 03 on
         Text summarization
     * Workshop-Volume 5. Association for Computational Linguistics, pp. 1âĂŞ8.
     *
     * @param sentence a CoreMap that holds the sentence we want to summarize to get the
         event for.
     */
    private void setEvent(CoreMap sentence, Result result) {
        Tree tree = sentence.get(TreeCoreAnnotations.TreeAnnotation.class);
        tree = getLeftmostLowestS(tree);//get leftmost-lowest S
        //remove time expressions
        removeTimeExpressions(tree, result);
        removeDeterminers(tree);//remove determiners
        xpOverXP(tree);//apply XP-over-XP rule
        xpBeforeNP(tree);//apply removal of XPs before NP rule
        cleanUp(tree);//remove any punctuation that could be left over
        tree = lastShorten(tree);//shorten the tree with the last two rules, removePPs and
            removeSBARs
        String event = produceString(tree);
        result.setEvent(event);
    }

    /**
     * Get the LeftmostLowest S subtree or the root of the tree if it doesn't exist.
     *
     * @param tree the Grammatical Structure of the sentence we are summarizing.
     * @return the leftmost-lowest S found in the passed in tree, can be the same tree.
     */
    private Tree getLeftmostLowestS(Tree tree) {
        //post-order to find first S
        Tree firstS = null;
        for (Tree node : tree.postOrderNodeList()) {
            if (!node.isLeaf() && node.value().equals("S")) {
```

```
            firstS = node;
            break;
        }
    }

    //if we haven't found a lowest S, then we use the root of the tree; else we just
        return the lowest S
    return (firstS == null) ? tree : firstS;
}

/**
 * Remove determiners like 'a' and 'the' from the tree.
 * Determiners are always the first child of their parent.
 *
 * @param tree the Grammatical Structure of the sentence we are summarizing.
 */
private void removeDeterminers(Tree tree) {
    for (Tree node : tree.preOrderNodeList()) {//loop over in pre order
        if (!node.isLeaf()) {//if we have children
            Tree firstChild = node.children()[0];
            if (!firstChild.isLeaf() && firstChild.value().equals("DT")) {//if it has
                children and it has the determiner tag
                Tree determiner = firstChild.children()[0];//determiners only have one
                    child
                if (determiner.value().equals("a") || determiner.value().equals("the"))
                    {//if our determiner is of the type
                    node.removeChild(0);  // 'a' or 'the' then we delete it, remove the
                        DT node that contains the 'a' or 'the'
                }
            }
        }
    }
}

/**
```

```
     * Go through the tree in pre-order, finding the last appearance of an xp over an xp
         (so check for np-np,vp-vp,and s-s),
     * then remove all children of the outer xp, except for the first first child.
     * Repeat if we are still above the threshold.
     *
     * @param tree the Grammatical Structure of the sentence we are summmarizing
     */
    private void xpOverXP(Tree tree) {
        if (tree.yield(new ArrayList<Label>()).size() > threshold) {//if we are above
             threshold, we need to reduce tree
            Tree toRemoveChild = null;
            for (Tree node : tree.preOrderNodeList()) {//need to loop over tree
                if (!node.isLeaf() && node.children().length > 1) {//if found a node
                    Tree possibleXpNode = node.children()[0];//who is of value XP
                    if (node.value().equals(possibleXpNode.value()) && // and its first
                         child is also XP, then need to
                            ((node.value().equals("NP") &&
                                 possibleXpNode.value().equals("NP")) //record it to delete
                                 its
                                    || (node.value().equals("VP") &&
                                         possibleXpNode.value().equals("VP")) //other children
                                    || (node.value().equals("S") &&
                                         possibleXpNode.value().equals("S")))) {
                        toRemoveChild = node;
                    }
                }
            }
            if (toRemoveChild != null) {//we have a node to delete all its children except
                 first from
                while (toRemoveChild.children().length > 1) {
                    toRemoveChild.removeChild(1);
                }
                xpOverXP(tree);
            }
        }
    }
```

```
/**
 * If the tree size is greater than the threshold, then delete any XP (PP,NP,VP) before
     the subject of the sentence
 * which is the NP child of S.
 * Find NP subject of S, then search through the tree in pre order:
 * if we reach the NP of S then stop
 * if we reach an XP that isn't the NP of S (the subject of the sentence), then store
     its parent and the child index
 * that it is in (in the parents children list), to later remove it from the tree.
 *
 * @param tree Grammatical structure of the sentence we are summarizing.
 */
private void xpBeforeNP(Tree tree) {
    if (tree.yield(new ArrayList<Label>()).size() > threshold &&
         tree.value().equals("S")) {
        Tree[] childrenOfS = tree.children();
        Tree pointsToFirstNP = null;
        for (int i = 0; i < childrenOfS.length; i++) {
            Tree child = childrenOfS[i];
            if (child.value().equals("NP")) {//found subject of sentence
                pointsToFirstNP = child;//remember it
                break;//no point in continue to loop, just wanted to find the NP child
                     of S
            }
        }
        if (pointsToFirstNP != null) {
            Tree toDeleteChild = null;//hold parent of node we have to delete
            int childIndexToDelete = 0;//hold index of child of parent that we have to
                 delete
            for (Tree node : tree.preOrderNodeList()) {
                if (node == pointsToFirstNP || toDeleteChild != null) {
                    break;
                } else if (!node.isLeaf()) {
                    Tree[] children = node.children();
                    for (int i = 0; i < children.length; i++) {
```

```
                    Tree child = children[i];
                    if (child == pointsToFirstNP) {
                        break;
                    } else if (child.value().equals("VP") ||
                         child.value().equals("NP") || child.value().equals("PP")) {
                        toDeleteChild = node;
                        childIndexToDelete = i;
                    }
                }
            }
        }
        if (toDeleteChild != null) {
            toDeleteChild.removeChild(childIndexToDelete);
            //xpBeforeNP(tree); ? recursively call
        }
    }
  }
}

/**
 * Applies the last two rules for iterative shortening. First will try to remove the
     rightmost-lowest PP until it
 * has reached the threshold. If it has removed all the PPs, or it can't delete anymore
     and it is still below the
 * threshold, then undo the changes and do the below.
 * Remove the SBARs until we are below the threshold, or there are no more left. If we
     are still above the threshold,
 * then delete all the PPs until we are below the threshold, or there are no more left.
 * <p>
 * According to the algorithm proposed in: Bonnie Dorr, David Zajic and Richard
     Schwartz. Hedge Trimmer: A Parse-and-Trim Approach to Headline
 * Generation. Proceedings of the HLT-NAACL 03 on Text summarization Workshop-Volume 5.
     Association for
 * Computational Linguistics, pp. 1âĂŞ8.
 *
```

```
 * @return a tree that has gone through the shortening of size smaller than threshold,
     or a tree where nothing can be removed
 */
private Tree lastShorten(Tree tree) {
    if (tree.yield(new ArrayList<Label>()).size() > threshold) {
        Tree copyOfTree = tree.deepCopy();
        //removePPs(copyOfTree);
        removeXs(tree, "PP");
        if (copyOfTree.yield(new ArrayList<Label>()).size() > threshold) {//we undo
             what we did previously and do SBAR removal and then PP
            //removeSBARs(tree);//need to remove SBARs
            removeXs(tree, "SBAR");
            //removePPs(tree);//then remove PPs according to rule
            removeXs(tree, "PP");
        } else {//just the PP removal was enough to reduce the size below the threshold
             so we return the tree which has undergone that removal
            return copyOfTree;
        }
    }
    return tree;
}

/**
 * Removes trailing Xs if we are below the threshold.
 *
 * @param tree the Tree that holds the grammatical structure of the text we are
     summarizing.
 * @param x   the type of X we are removing, eg: PP, SBAR, from the tree.
 */
private void removeXs(Tree tree, String x) {
    if (tree.yield(new ArrayList<Label>()).size() > threshold) {
        Tree toDeleteChild = null;
        int childIndexToDelete = 0;
        for (Tree node : tree.preOrderNodeList()) {
            if (!node.isLeaf()) {
                Tree[] children = node.children();
```

```
                for (int i = 0; i < children.length; i++) {
                    if (children[i].value().equals(x)) {
                        toDeleteChild = node;
                        childIndexToDelete = i;
                    }
                }
            }
        }
        if (toDeleteChild != null) {
            toDeleteChild.removeChild(childIndexToDelete);
            removeXs(tree, x);
        }
    }
}

/**
 * Produce a string based on the leaf nodes in the tree.
 *
 * @param tree the Tree that has the grammatical structure of the sentence we want to
     produce.
 * @return the String that follows the grammatical structure passed in.
 */
private String produceString(Tree tree) {
    String toReturn = "";
    List<Tree> preOrderList = tree.preOrderNodeList();
    for (int i = 0; i < preOrderList.size(); i++) {
        Tree node = preOrderList.get(i);
        if (node.isLeaf()) {
            toReturn += node.value();
            if (i != preOrderList.size() - 1) {
                toReturn += " ";
            }
        }
    }
    return toReturn;
}
```

```
/**
 * Called to remove punctuation that is at the start of the tree
 *
 * @param tree the Tree from which we are removing the punctuation
 */
private void cleanUp(Tree tree) {//if its the first character, than its the first
    child of root S
    if (!tree.isLeaf() && tree.value().equals("S") &&
        tree.children()[0].value().equals(",")) {
       tree.removeChild(0);
    }
}

/**
 * Remove time expressions in the tree.
 * According to the algorithm proposed in: Bonnie Dorr, David Zajic and Richard
    Schwartz. Hedge Trimmer: A Parse-and-Trim Approach to Headline
 * Generation. Proceedings of the HLT-NAACL 03 on Text summarization Workshop-Volume 5.
    Association for
 * Computational Linguistics, pp. 1âĂŞ8.
 *
 * @param tree  the Tree that contains the Grammatical Structure of the sentence we are
    summarizing.
 * @param result the backend.process.Result object that contains previously found time
    expressions in the original sentence.
 */
private void removeTimeExpressions(Tree tree, Result result) {
    /*go over the tree, for every PP check if it has a NP child, if so check if that NP
        contains the time expression, if so delete the PP.
    * afterwards delete any NP that contains the time expression
    * */
    //System.out.println("Tree going in:");
    //tree.pennPrint();
    ArrayList<Pair<Tree, Integer>> toDeleteNodes = new ArrayList<>();
    for (Tree node : tree.preOrderNodeList()) {
```

```
    if (!node.isLeaf()) {
        Tree[] children = node.children();
        for (int i = 0; i < children.length; i++) {
            if (children[i].value().equals("PP") && !children[i].isLeaf()) {
                Tree[] childOfChildren = children[i].children();
                for (int j = 0; j < childOfChildren.length; j++) {
                    if (childOfChildren[j].value().equals("NP") &&
                         !childOfChildren[j].isLeaf()) {
                        String childOfChildrenString =
                             produceString(childOfChildren[j]);
                        if (result.hasDate(childOfChildrenString)) {
                            System.out.println("Found a match");
                            toDeleteNodes.add(new Pair<>(node, i));
                            break;
                        }
                    }
                }
            }
        }
    }
}
//now delete
for (Pair<Tree, Integer> pair : toDeleteNodes) {
    if (pair.first.getChild(pair.second).value().equals("PP")) {
        System.out.println("Removing: " + pair.first.getChild(pair.second));
        pair.first.removeChild(pair.second);
    }
}
//tree.pennPrint();
//clear toDeleteNodes
toDeleteNodes.clear();
//now find any NP with the time expression
for (Tree node : tree.preOrderNodeList()) {
    if (!node.isLeaf()) {
        Tree[] children = node.children();
        for (int i = 0; i < children.length; i++) {
```

```
                if (children[i].value().equals("NP") && !children[i].isLeaf()) {
                    String childString = produceString(children[i]);
                    if (result.hasDate(childString)) {
                        System.out.println("Found a match");
                        toDeleteNodes.add(new Pair<>(node, i));
                    }
                }
            }
        }
        for (Pair<Tree, Integer> pair : toDeleteNodes) {
            if (pair.first.getChild(pair.second).value().equals("NP")) {
                System.out.println("Removing: " + pair.first.getChild(pair.second));
                pair.first.removeChild(pair.second);
            }
        }
        //System.out.println("Tree afterwards:");
        //tree.pennPrint();

    }

}
```

FileData.java

```
package backend.process;


import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.attribute.BasicFileAttributes;
import java.nio.file.attribute.FileTime;
import java.text.ParseException;
import java.text.SimpleDateFormat;
```

```
import java.time.LocalDate;
import java.util.Date;

/**
 * Class that holds the relevant data (for this project) of a File. In this case, the
     File's name and path in the System.
 */
public class FileData implements Comparable<FileData> {
    private final static SimpleDateFormat inputSimpleDateFormat = new
        SimpleDateFormat("dd-MM-yyyy");
    private final static SimpleDateFormat outputSimpleDateFormat = new
        SimpleDateFormat("yyyy-MM-dd");
    private final static String epochDateFormatted = "1970-01-01";
    private String fileName;
    private String filePath;
    private Date creationDate;

    /**
     * Used to create a FileData object, by providing its name and path.
     *
     * @param fileName the name of the File.
     * @param filePath the path of the File.
     */
    public FileData(String fileName, String filePath) {
        this.fileName = fileName;
        this.filePath = filePath;
    }

    public FileData(File file) {
        if (file.exists() && file.isFile()) {
            this.fileName = file.getName();
            this.filePath = file.getAbsolutePath();
            this.creationDate = getCreationDate(file);//set the creation date for this file
        }
    }
```

```java
/**
 * Get the name of the File that produced this FileData.
 *
 * @return the name of the File.
 */
public String getFileName() {
    return fileName;
}

/**
 * Set the name of the File that produced this FileData.
 *
 * @param fileName the name of the File.
 */
public void setFileName(String fileName) {
    this.fileName = fileName;
}

/**
 * Get the path of the File that produced this FileData.
 *
 * @return the path of the File.
 */
public String getFilePath() {
    return filePath;
}

/**
 * Set the path of the File that produced this FileData.
 *
 * @param filePath the path of the File.
 */
public void setFilePath(String filePath) {
    this.filePath = filePath;
}
```

```
/**
 * Print the information of this FileData.
 *
 * @return a String with the name and path of the File being represented.
 */
@Override
public String toString() {
    return String.format("Name: \%s, Path: \%s", fileName, filePath);
}

/**
 * Checks whether the given input is equal to this object.
 *
 * @param obj the Object to check equality with this object.
 * @return true if obj is a FileData object with the same data as this object; false
     otherwise.
 */
@Override
public boolean equals(Object obj) {
    try {
        FileData otherFileData = (FileData) obj;
        return fileName.equals(otherFileData.fileName) &&
             filePath.equals(otherFileData.filePath) &&
             creationDate.equals(otherFileData.creationDate);
    } catch (Exception e) {
        return false;
    }
}

/**
 * The creationDate (or baseDate) of the File this is representing.
 *
 * @return the creationDate (or baseDate) of the File as a Date object.
 */
public Date getCreationDate() {
```

```
    return creationDate;
}

/**
 * Set the creation date as a String for the File this FileData is representing.
 *
 * @param creationDate the creation date as a String input of type: dd-MM-yyyy.
 */
public void setCreationDate(String creationDate) {
    System.out.println("Set Creation Date: " + creationDate);
    try {
        this.creationDate = inputSimpleDateFormat.parse(creationDate);
    } catch (ParseException e) {
        e.printStackTrace();
    }
}

/**
 * Get the creation date in the format of day, followed by month, followed by year.
 *
 * @return the creation date in the format: dd-MM-yyyy
 */
public String getCreationDateFormattedDayMonthYear() {
    return (creationDate != null) ? inputSimpleDateFormat.format(creationDate) : null;
}

/**
 * Get the creation date in the format of year, followed by month, followed by day.
 *
 * @return the creation date in the format: yyyy-MM-dd.
 */
public String getCreationDateFormattedYearMonthDay() {
    return outputSimpleDateFormat.format(creationDate);
}

/**
```

```
     * Used to get the creation date of a input File, by looking at the attributes
         associated with the File.
     *
     * @param file the input File.
     * @return the creation date, or the current date now if no creation date is available,
         as a Date object.
     */
    private static Date getCreationDate(File file) {
        String creationDate = LocalDate.now().toString();//LocalDate.now gives you the Date
            for the current moment (default), in the format yyyy-MM-dd
        Date toReturn = null;
        //try and get file creation date
        Path filePath = Paths.get(file.getAbsolutePath());//only way to use
            BasicFileAttributes is to pass in a Path,
        try {//so get the Path for the passed in File
            BasicFileAttributes basicFileAttributes = Files.readAttributes(filePath,
                BasicFileAttributes.class);//creation date of a File is a basic file
                attribute
            FileTime creationTime = basicFileAttributes.creationTime();//can be epoch time
                if it does not exist, don't set it in that case
            String possibleDate =
                inputSimpleDateFormat.format(creationTime.toMillis());//need to check its
                not epoch time, as else the creation is not valid for this file
            if (!possibleDate.equals(epochDateFormatted)) {//epoch time date is given if it
                cant find a creation date
               creationDate = possibleDate;
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        try {
            toReturn = inputSimpleDateFormat.parse(creationDate);
        } catch (ParseException e) {
            e.printStackTrace();
        }
        return toReturn;
```

```
    }



    /**
     * Used to compare a given FileData to this FileData, by looking at their filename (to
         sort by their name, in
     * ascending order).
     *
     * @param o the other FileData object we are comparing to.
     * @return the result of comparing the file name of this FileData with the filename of
         the other FileData.
     */
    @Override
    public int compareTo(FileData o) {
        return fileName.compareTo(o.fileName);
    }
}
```

ProcessFileCallback.java

```
package backend.process;


import java.util.ArrayList;


/**
 * Interface to be implemented, to inform the Listener of ProcessFile Thread of when a
     file has been processed.
 */
public interface ProcessFileCallback {
    /**
     * Inform the Listener that the passed in File to ProcessFile has been processed, and
         pass the Results of processing
     * the given File.
     *
     * @param results the Results from processing the given File passed into the
         ProcessFile Thread.
```

```
     * @param fileData the File Data of the File that produced these Results.
     */
    void callBack(ArrayList<Result> results, FileData fileData);
}
```

ProcessFiles.java

```
package backend.process;


import backend.system.BackEndSystem;
import backend.system.SystemState;
import org.apache.pdfbox.pdmodel.PDDocument;
import org.apache.pdfbox.text.PDFTextStripper;
import org.apache.poi.xwpf.extractor.XWPFWordExtractor;
import org.apache.poi.xwpf.usermodel.XWPFDocument;


import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import java.util.concurrent.Semaphore;



/**
 * Handles the parsing of files, and the multi-threading of the backend.process.Engine.
 */
public class ProcessFiles implements ProcessFileCallback {
    private static int maxNoOfThreads;//this value is determined by the settings of the
        System (by default 2)
    private Semaphore semaphore;//will use the maxNoOfThreads to control how many Threads
        will be running in parallel
    private Semaphore semaphoreFinished = new Semaphore(0);//so that we wait until all
        threads finish
```

```
private ArrayList<Result> results = new ArrayList<>();
private int filesToGo;//to notify the listener when it is done

/**
 * For the list of Files passed in, it will process each in separate threads, while
     respecting the maximum number
 * of threads that can run at any given time (i.e. maximum 2 extra threads running at a
     time).
 * Will call gotResults() on the backend.process.CallbackResults object when all files
     have been processed, and a list of Results has
 * been produced.
 * <p>
 * <b>Should be the only thing called in this class by its Users.</b>
 *
 * @param files   the list of File objects that contain text that needs to be
     processed (atm only processes .docx/.pdf/.txt files)
 * @param fileDatas the list of FileData objects, where the index corresponds the File
     in the same index in the files list, that
 *                contains the needed data for each File.
 */
public List<Result> processFiles(List<File> files, List<FileData> fileDatas) {
    maxNoOfThreads =
         BackEndSystem.getInstance().getSettings().getMaxNoOfThreads();//get the
         Settings value
    semaphore = new Semaphore(maxNoOfThreads);//set the Max number of Threads that can
         run in parallel
    //should only run if we are not Processing
    //this will also set up the StanfordCoreNLP (when GUI is implemented, it will
         already by set up, as it will be the first thing ran)
    System.out.println("Will try to run, with maxNoOfThreads: " + maxNoOfThreads + "
         and available permits: " + semaphore.availablePermits());
    if (BackEndSystem.getInstance().getSystemState() != SystemState.PROCESSING &&
         files.size() == fileDatas.size()) {//if we arent processing, then we can begin
         to do that
        System.out.println("Is running");
        filesToGo = files.size();//and when we need to call
```

```
        BackEndSystem.getInstance().setSystemState(SystemState.PROCESSING);
        System.out.println("In Thread: " + Thread.currentThread().toString());
        for (int i = 0; i < files.size(); i++) {
            File file = files.get(i);
            FileData fileData = fileDatas.get(i);//should be the same
            //check they are the same?
            //acquire from the semaphore
            try {
                System.out.println("Trying to acquire semaphore for file: " + file);
                semaphore.acquire();//will wait if there is already maxnoofthreads
                    running, until one finishes: then it gets to run
                System.out.println("Acquired semaphore for file: " + file);
                //process file
                Thread thread = new ProcessFile(file, this, fileData);//pass a reference
                    so that the thread can call this when it finishes processing the file
                thread.start();//start processing this file(get its text and pass it to
                    the backend.process.Engine)
            } catch (InterruptedException e) {
                e.printStackTrace();
                //should release semaphore and reduce filesToGo count if the Thread is
                    interrupted
            }
        }
        try {
            System.out.println("Going to wait until the last files are processed.");
            semaphoreFinished.acquire();
            System.out.println("The last files have been processed.");
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        return results;
    }
    return null;
}

/**
```

```
     * Called when the ProcessFile Thread has finished processing a file. It is
         synchronized to avoid two separate threads
     * trying to add to the list of Results at the same time.
     * Enforces the rule of releasing the semaphore, as this Thread finished running, to
         then allow waiting Threads to run.
     * Will inform the CallbackResult object when it finishes processing all Files.
     *
     * @param results the backend.process.Result objects produced by Processing the given
         file in the Thread.
     * @param fileData the data of the File that produced these Results
     */
    public synchronized void callBack(ArrayList<Result> results, FileData fileData) {
        //we finished processing a file
        filesToGo--;//one less to look at
        System.out.println("Files to go: " + filesToGo);
        //add the results to the list held
        this.results.addAll(results);
        //release semaphore
        System.out.println("Released semaphore from Thread: " +
             Thread.currentThread().toString());
        semaphore.release();
        //check if we have processed everything, if so release the finished semaphore
        if (filesToGo == 0) {
            //has processed
            BackEndSystem.getInstance().setSystemState(SystemState.PROCESSED);
            //has returned the results so we finished
            BackEndSystem.getInstance().setSystemState(SystemState.FINISHED);
            semaphoreFinished.release();//value is now 1, so the thread that was acquiring
                 can continue
        }
    }

    /**
     * Called to return the text of a given File.
     *
     * @param file the given File.
```

```
     * @return the text of the given File.
     */
    public String getTextInFile(File file) {
        return new ProcessFile(file, null, null).getText(file);
    }

    /**
     * In charge of Processing just one File in a separate Thread.
     */
    private static class ProcessFile extends Thread {
        //Need to release even if it messes up
        File file;
        ProcessFileCallback processFileCallback;
        FileData fileData;

        /**
         * Create a ProcessFile object that holds the data needed: the File to process and
             who to callback when the
         * backend.process.Engine finishes processing.
         *
         * @param file             the File to process.
         * @param processFileCallback who to inform when the backend.process.Engine
             finished processing the given file.
         */
        ProcessFile(File file, ProcessFileCallback processFileCallback, FileData fileData)
            {//hold sempahore
            this.file = file;
            this.processFileCallback = processFileCallback;
            this.fileData = fileData;
        }

        /**
         * What starts running on a separate Thread. Will first get the Text for the given
             file, and then pass it to the
         * backend.process.Engine, which will return a list of Results that is passed to
             the backend.process.ProcessFileCallback.
```

```
     */
    @Override
    public void run() {
        super.run();
        System.out.println("For: " + file + " in Thread: " +
             Thread.currentThread().toString());//for logging purposes
        ArrayList<Result> toReturnResults = new ArrayList<>();//initially no results
        //check file exists in system
        if (fileExists(file)) {
            //fileData = new FileData(file.getName(), file.getAbsolutePath());
            //get the text for that file
            String toProcess = getText(file);//will get the text for the file
                 considering its extension
            //run engine on this
            if (!toProcess.equals("")) {//if we actually have text to process, don't
                 waste time attempting to process else
                String baseDate =
                     fileData.getCreationDateFormattedYearMonthDay();//since we will need
                     to process, get a base date to use
                System.out.println("Base Date for " + file.getName() + " is: " +
                     baseDate);
                toReturnResults = new Engine().getResults(toProcess, baseDate);//pass in
                     file data, so each result holds it
                addFileData(fileData, toReturnResults);
            }
        }
        //call the backend.process.ProcessFileCallback that we finished processing and
             return the results of processing that one File.
        processFileCallback.callBack(toReturnResults, fileData);
    }

    /**
     * For the list passed in, set for all of them the given FileData.
     *
     * @param fileData the FileData to be set to all the Results passed in.
     * @param results the Results for which the FileData needs to be set.
```

```
     */
    private void addFileData(FileData fileData, ArrayList<Result> results) {
        for (Result result : results) {
            result.setFileData(fileData);
        }
    }

    /**
     * Check that we can actually use the given File.
     *
     * @param file the File that we can check that we can read it.
     * @return true if it is indeed a File, that it still exists in the system, and
         that we can read data from it, have access for that.
     */
    private boolean fileExists(File file) {
        return file.exists() && file.isFile() && file.canRead();
    }

    /**
     * For the given File, perform the operations needed to obtain its text, as
         different file extension have different
     * encodings.
     *
     * @param file the File for which we need to get text from.
     * @return the text from the File, or empty text if it was not possible to get text
         from the File.
     */
    private String getText(File file) {
        String fileName = file.getName();//file should have format: name.extension
        int lastDotPosition = fileName.lastIndexOf(".");//to separate name from the
            extension
        if (lastDotPosition != -1) {//-1 if . was no where in the file name
            String fileExtension = fileName.substring(lastDotPosition);//everything
                after the last dot, the extension
            System.out.println("File name: " + fileName + " file extension: " +
                fileExtension);
```

```
        switch (fileExtension) {
            case ".pdf"://file has a .pdf extension
                //call method to get string from pdf
                return getTextPDF(file);
            case ".txt"://file has a .txt extension
                //call method to get string from txt
                return getTextTXT(file);
            case ".docx":
                return getTextDoc(file);
            default://could hold list of files that could not be processed
                //get string from default process (open file and use buffered reader)
                break;
        }
    }
    return "";
}



/**
 * Get the text for a PDF File using the appropriate library (Apache PDFBox).
 *
 * @param file PDF File to get text from.
 * @return the text in the PDF File, or an empty String if it was not possible.
 */
private String getTextPDF(File file) {
    String toReturn = "";//base text, if it fails we just return empty text
    try {
        PDDocument pdDocument = PDDocument.load(file);//create Document that has
            processed the bytes in the pdf file
        PDFTextStripper pdfTextStripper = new PDFTextStripper();
        //pdfTextStripper.setSortByPosition(true);//in the case the program that
            created the page, didnt place the text in the order it is shown (so
            could read text in wrong order)
        toReturn = pdfTextStripper.getText(pdDocument).replaceAll(pdfTextStripper
          .getLineSeparator(), "");//to then get its text
        System.out.println("Line Seperator: " + pdfTextStripper.getLineSeparator());
```

```
            pdDocument.close();//always remember to close the stream, or document in
                this case
        } catch (IOException e) {
            e.printStackTrace();
        }
        //System.out.println("For file:"+file.getName()+" has text: "+toReturn);
        return toReturn;
    }

    /**
     * Get the text for a TXT File, assuming the enconding is UTF-8(later determine
         encoding and pass it to the Scanner
     * that processes the File).
     *
     * @param file a TXT File to get the text from.
     * @return the text in the TXT File, or an empty String if it was not possible.
     */
    private String getTextTXT(File file) {
        //should determine character set, can look at
            https://code.google.com/archive/p/juniversalchardet/
        String toReturn = "";//base text to return, empty String
        try {
            Scanner scanner = new Scanner(file, "UTF-8");//uses the default character
                set UTF-8
            while (scanner.hasNextLine()) {//while we can read more
                toReturn += scanner.nextLine();
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        return toReturn;
    }

    /**
     * Process a DOCX Document using the appropriate library (Apache POI).
     *
```

```
         * @param file the DOCX File to get the text from.
         * @return the text in the File, or an empty String if it was not possible to
             process the File.
         */
        private String getTextDoc(File file) {
            String toReturn = "";//base text to return, empty String
            try {
                XWPFDocument xwpfDocument = new XWPFDocument(new
                     FileInputStream(file));//used to process docx documents
                XWPFWordExtractor xwpfWordExtractor = new XWPFWordExtractor(xwpfDocument);
                toReturn = xwpfWordExtractor.getText();
            } catch (IOException e) {
                e.printStackTrace();
            }
            //System.out.println("For file:"+file.getName()+" has text: "+toReturn);
            return toReturn;
        }
    }
}
```

Result.java

```
package backend.process;

import java.util.ArrayList;
import java.util.Date;
import java.util.HashSet;
import java.util.Set;

/**
 * Holds the data for one event in the Timeline.
 */
public class Result implements Comparable<Result>, Cloneable {
    private ArrayList<String> dates;
    private String event;
    private Set<String> subjects;
```

```
    private TimelineDate timelineDate;
    private FileData fileData;
    private String originalString;

    /**
     * Initialises variables.
     */
    public Result() {
        this("");
    }

    /**
     * Initialises variables and sets the event.
     *
     * @param event the event text of this event.
     */
    private Result(String event) {
        dates = new ArrayList<>();
        this.event = event;
        subjects = new HashSet<>();
        timelineDate = new TimelineDate();
        originalString = "";
    }

    /**
     * Checks whether this backend.process.Result already has a given date (in String
         format)
     *
     * @param date the String date being checked.
     * @return true if the list of String dates has date, false otherwise.
     */
    public boolean hasDate(String date) {
        for (String dDate : dates) {
            if (dDate.equals(date)) {
                return true;
            }
```

```
        }
        return false;
    }

    /**
     * The event string of this event.
     *
     * @return the string event of this event.
     */
    public String getEvent() {
        return event;
    }

    /**
     * Sets the event of this Timeline event.
     *
     * @param event what we are setting as an event.
     */
    public void setEvent(String event) {
        this.event = event;
    }

    /**
     * Add a string date to the list of held string dates, dates relevant to this event.
     *
     * @param date the String date that is being added.
     */
    public void addDate(String date) {
        dates.add(date);
    }

    /**
     * Get the list of String dates held.
     *
     * @return the list of String dates held.
     */
```

```
public ArrayList<String> getDates() {
    return dates;
}

/**
 * Get the list of Subjects relevant to this event.
 *
 * @return the list of Subjects held.
 */
public Set<String> getSubjects() {
    return subjects;
}

/**
 * Add a subject to the list of Subjects held.
 *
 * @param subject a subject to add.
 */
public void addSubject(String subject) {
    subjects.add(subject);
}

/**
 * @return a String showing all the information held by this backend.process.Result
     object.
 */
@Override
public String toString() {
    return String.format("Date: %s, Subject: %s, Event: %s", timelineDate, subjects,
        event);
}

/**
 * For the given passed in String date, it will update the backend.process.TimelineDate
     of this backend.process.Result object.
 *
```

```
     * @param date a String that contains date information that needs to be passed into the
          timeline date of this backend.process.Result object.
     */
    public void addDate_1(String date, String baseDate) {
        System.out.println("About to parse: " + date);
        timelineDate.parse(date, baseDate);
    }



    /**
     * Used to compare two backend.process.Result objects by their
          backend.process.TimelineDate.
     *
     * @param o the other backend.process.Result object.
     * @return the result of comparing the backend.process.TimelineDate of this object,
          with the backend.process.TimelineDate of the other (input) backend.process.Result
          object.
     */
    @Override
    public int compareTo(Result o) {
        return this.timelineDate.compareTo(o.timelineDate);
    }


    /**
     * Get the TimelineDate for this Result.
     *
     * @return the date data used for this Result in the Timeline.
     */
    public TimelineDate getTimelineDate() {
        return timelineDate;
    }


    /**
     * Set the TimelineDate for this Result.
     *
     * @param timelineDate the date data used for this Result in the Timeline.
```

```
     */
    public void setTimelineDate(TimelineDate timelineDate) {
        this.timelineDate = timelineDate;
    }

    /**
     * Get the FileData for this Result.
     *
     * @return the File data of the File that produced this Result.
     */
    public FileData getFileData() {
        return fileData;
    }

    /**
     * Set the FileData for this Result.
     *
     * @param fileData the File data of the File that produced this Result.
     */
    public void setFileData(FileData fileData) {
        this.fileData = fileData;
    }

    /**
     * Used to produce a String of the Subjects of this Result. The String is every item in
         the Subjects set, separated
     * by commas.
     *
     * @return a String of the Subjects of this Result, separated by commas.
     */
    public String getSubjectsAsString() {
        String toReturn = "";
        for (int i = 0; i < subjects.size(); i++) {
            toReturn += subjects.toArray()[i];
            if (i < subjects.size() - 1) {
                toReturn += ", ";
```

```
        }
    }
    return toReturn;
}

/**
 * Makes a new Result object, with the same data as this one but not referencing to the
     same place in memory. So
 * changes in the new Result object do not change the data of this Result object.
 *
 * @return a new Result object with the same data but not pointing to the data in
     memory.
 * @throws CloneNotSupportedException
 */
@Override
public Object clone() throws CloneNotSupportedException {
    super.clone();
    Result copyResult = new Result();
    //copying the dates
    TimelineDate copyTimelineDate = new TimelineDate();
    copyTimelineDate.setDate1((Date) timelineDate.getDate1().clone());
    if (timelineDate.getDate2() != null) {
        copyTimelineDate.setDate2((Date) timelineDate.getDate2().clone());
    } else {
        copyTimelineDate.setDate2(null);
    }
    copyResult.setTimelineDate(copyTimelineDate);
    //copying the event
    copyResult.setEvent(event);
    //copying the subjects
    copyResult.subjects = new HashSet<>(subjects);
    //set the filedata
    copyResult.setFileData(fileData);//all results of the same file point to the same
        filedata (not a unique one)
    //set the original sentence
    copyResult.setOriginalString(originalString);
```

```
    return copyResult;
}

@Override
public boolean equals(Object obj) {
    if(obj instanceof Result){
        Result other = (Result) obj;
        boolean areFileDataEqual = ((fileData == null && other.fileData == null) ||
                (fileData != null && other.fileData != null &&
                    fileData.equals(other.fileData)));//as FileData is the only one that
                    could be null

        return subjects.equals(other.getSubjects()) && event.equals(other.event) &&
             originalString.equals(other.originalString)
                && areFileDataEqual && timelineDate.equals(other.getTimelineDate());
    }
    return false;
}

/**
 * Get the original sentence that produced this Result object.
 *
 * @return the original sentence that produced this.
 */
public String getOriginalString() {
    return originalString;
}

/**
 * Set the original sentence that produced this Result object.
 *
 * @param originalString the original sentence that produced this.
 */
public void setOriginalString(String originalString) {
    this.originalString = originalString;
}
```

```
}
```

TimelineDate.java

```
package backend.process;


import edu.stanford.nlp.util.Pair;
import org.joda.time.DateTime;
import org.joda.time.Days;


import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.*;
import java.util.regex.Pattern;


/**
 * Attempts to generate an exact date for an event, to then order the events.
 * Holds the start and end date (appropriately) for each event in the timeline. It updates
     as new dates, relevant to the
 */
//If just year-month should create range?
public class TimelineDate implements Comparable<TimelineDate> {
    private static final String year = "0001";
    private static final String month = "01";
    private static final String day = "01";
    private static final Map<String, Pair<String, String>> seasonMap;
    private static final Map<Character, String> durationMap;
    private static final Map<Character, String> timeMap;
    private final SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd G");
    private final SimpleDateFormat dayMonthYearFormat = new SimpleDateFormat("dd-MM-yyyy
        G");


    static {
        /*
            Seasons are given according to educationuk.org, by:
                Winter December , January, February (12,1,2)
```

```
            Spring is March, April and May (3,4,5)
            Summer is June, July, August (6,7,8)
            Autumn September, October, November (9,10,11)
     */
    Map<String, Pair<String, String>> seasonM = new HashMap<>();
    seasonM.put("WI", new Pair<>("12", "02"));
    seasonM.put("SP", new Pair<>("03", "05"));
    seasonM.put("SU", new Pair<>("06", "08"));
    seasonM.put("FA", new Pair<>("09", "11"));
    seasonMap = seasonM;

    //durationMap
    Map<Character, String> durationM = new HashMap<>();
    durationM.put('P', "Period:");
    durationM.put('Y', "Year(s)");
    durationM.put('M', "Month(s)");
    durationM.put('W', "Week(s)");
    durationM.put('D', "Day(s)");
    durationMap = durationM;

    //timeMap
    Map<Character, String> timeM = new HashMap<>();
    timeM.put('T', "Time:");
    timeM.put('H', "Hour(s)");
    timeM.put('M', "Minute(s)");
    timeM.put('S', "Second(s)");
    timeMap = timeM;
}

//patterns are thread safe
private final static Pattern onlyYearPattern =
     Pattern.compile("(\\d{4})|(\\d{3}X)|(\\d{2}XX)|(\\dXXX)|(XXXX)");
private final static Pattern onlyMonthPattern = Pattern.compile("\\d{2}");
private final static Pattern onlyWeekNumberPattern = Pattern.compile("W\\d{2}");
private final static Pattern onlySeasonPattern = Pattern.compile("[A-Z]{2}");
private final static Pattern onlyDayPattern = Pattern.compile("\\d{2}.*");
```

```
private final static Pattern onlyPresentRefPattern =
    Pattern.compile(".*PRESENT_REF.*");
private final static Pattern onlyPastRefPattern = Pattern.compile(".*PAST_REF.*");
private final static Pattern onlyFutureRefPattern = Pattern.compile(".*FUTURE_REF.*");
private final static Pattern onlyBeforeYearPattern =
    Pattern.compile("(\\-\\d{4})|(\\-\\d{3}X)|(\\-\\d{2}XX)|(\\-\\dXXX)|(\\-XXXX)");
private final static Pattern onlyWeekendPattern = Pattern.compile("WE");
private final static Pattern yearMonthDayPattern =
    Pattern.compile("\\d{4}\\-\\d{2}\\-\\d{2}");
private Calendar calendar;
private Date date1;//first (min, start) date
private Date date2;//second (max, end) date
private String dateStr;
private String baseDate;
private String durationData;//holds the latest duration data (additional info to show
    with event)
//have a pair of list dates and duration string, if you use the dates pass in the
    string as additional info
private int range = -1;

/**
 * Initialises the Calendar used to determine dates based on week number.
 */
public TimelineDate() {
    calendar = Calendar.getInstance();
    simpleDateFormat.setLenient(false);//can only create correct dates
}

/**
 * Update the dates hold by this, based on the input text.
 *
 * @param date a date provided by the StanfordCoreNLP library: it is a normalized entity
 */
public void parse(String date, String baseDate) {
    System.out.println("Input: " + date);
    Pair<ArrayList<Date>, String> dateDurationPair = null;
```

```
    ArrayList<Date> dates = new ArrayList<>();
    String durationData = null;
    this.baseDate = baseDate;
    //splitting INTERSECT
    String[] splitDate = date.split("INTERSECT");
    if (splitDate.length > 0) {// on the first part of the date, which is just a date,
         get its specfic date
        String possibleDates = splitDate[0];//this date could also be a range, ie
             include /
        String[] splitRange = possibleDates.split("/");
        for (String possibleDate : splitRange) {
            dates.addAll(getDate(possibleDate.trim()));// from processing the individual
                 date, add it to dates
        }
        //process INTERSECT data
        if (splitDate.length > 1) {
            String trimmedDuration = splitDate[1].trim();
            System.out.println("Processing: " + trimmedDuration);
            //process trimmed part which contains the duration data
            durationData = processINTERSECT(trimmedDuration);
        }
        //resulting list of dates and duration data should be put in a pair that is
             processed, where durationData
        //is only set if one of our dates are set
        dateDurationPair = new Pair<>(dates, durationData);
    }
    //if we had INTERSECT then we should process it for additional info to show
    //if we have more than 2 dates in the list, then keep the minimum date and max date
         and remove all the others
    enforceRule(dateDurationPair, date);
}

/**
 * Processes the INTERSECT data (duration of an event) provided sometimes with Date
     normalized entity tags.
 * Based on the ISO Standard 8601
```

```
     *
     * @param intersectData the data after "INTERSECT" in the normalized Date entity tags.
         Should start with 'P'.
     * @return a technical String representation of that data based on the ISO Standard
         8601, i.e. Period X Year(s) Y Day(s)...
     */
    private String processINTERSECT(String intersectData) {
        String toReturn = "";
        //input starts with P
        char[] dataSplit = intersectData.toCharArray();
        if (dataSplit.length > 0 && dataSplit[0] == 'P') {
            System.out.println("Entered Period");
            for (int i = 0; i < dataSplit.length; i++) {
                char info = dataSplit[i];
                System.out.println(info);
                //when find T (go to time process method, once returned from that break)
                if (info == 'T') {
                    //go time method, and what it returns add toReturn
                    //make string from here till end of data
                    String hereToEnd = intersectData.substring(i, intersectData.length());
                    System.out.println("End Data: " + hereToEnd);
                    String toAdd = processTime(hereToEnd);
                    System.out.println("Adding: " + toAdd);
                    toReturn += toAdd;
                    break;//not processing the rest of the array as its character data
                }
                String fullText = ((durationMap.get(info) != null) ? " " +
                     durationMap.get(info) + " " : info + "");
                System.out.println("Found: " + fullText);

                toReturn += fullText;
            }
            System.out.println("Every: " + toReturn.trim());
        }
        return toReturn.trim();
    }
```

```
/**
 * Processes the Time duration part of the INTERSECT data after a Data has been given a
     normalized entity tag.
 * Based on the ISO Standard 8601.
 *
 * @param timeData time data that starts with 'T', based on ISO Standard 8601.
 * @return a technical String representation of the data passed in, i.e. Time X Hour(s)
     Y Minute(s)...
 */
private String processTime(String timeData) {
    String toReturn = "";
    char[] splitTimeData = timeData.toCharArray();
    if (splitTimeData.length > 0 && splitTimeData[0] == 'T') {
        System.out.println("Processing Time data");
        for (char charTime : splitTimeData) {
            System.out.println("Processing: " + charTime);
            String fullText = ((timeMap.get(charTime) != null) ? " " +
                 timeMap.get(charTime) + " " : charTime + "");
            System.out.println("Found: " + fullText);
            toReturn += fullText;
        }
        System.out.println(toReturn.trim());
    }
    return toReturn.trim();
}

/**
 * For the given input, produce a list of dates based on it.
 * Based on the ISO Standard 8601.
 *
 * @param date an input text that contains date information (can be exact or relative).
 * @return a list of exact Dates formed from the input.
 */
private ArrayList<Date> getDate(String date) {
    calendar.clear();
```

```
        ArrayList<Date> toReturn = new ArrayList<>();
        String year1 = year;
        String month1 = month;
        String day1 = day;
        String year2 = null;
        String month2 = null;
        String day2 = null;
        boolean isBC = false;
        System.out.println("Trying to match: " + date);
        //need to check if its a PRESENT_REF, FUTURE_REF or PAST_REF
        if (onlyPastRefPattern.matcher(date).matches()) {
            System.out.println("PAST REF");
            //past so make range from 0001-01-01 -> base date (range)
            if (yearMonthDayPattern.matcher(baseDate).matches()) {
                //base date has the format yyyy-MM-dd
                String[] splitBaseDate = baseDate.split("-");
                //year1, month1, day1 values stay with default value
                year2 = splitBaseDate[0];
                month2 = splitBaseDate[1];
                day2 = splitBaseDate[2];//safe as pattern matched
            }
        } else if (onlyPresentRefPattern.matcher(date).matches()) {
            System.out.println("PRESENT REF");
            //use the base date (single date)
            if (yearMonthDayPattern.matcher(baseDate).matches()) {
                System.out.println("BaseDate matches: " + baseDate);
                String[] splitDate = baseDate.split("-");//so its safe to split it into 3
                    parts as pattern matched above
                year1 = splitDate[0];
                month1 = splitDate[1];
                day1 = splitDate[2];
            }
        } else if (onlyFutureRefPattern.matcher(date).matches()) {
            System.out.println("FUTURE REF");
            //future, from now til the last date we allow 9999-12-31 (range)
            if (yearMonthDayPattern.matcher(baseDate).matches()) {
```

```
            System.out.println("BaseDate matches: " + baseDate);
            String[] splitDate = baseDate.split("-");//so its safe to split it into 3
                 parts as pattern matched above
            year1 = splitDate[0];
            month1 = splitDate[1];
            day1 = splitDate[2];
        }
        year2 = "9999";//future point
        month2 = "12";
        day2 = "31";
    } else {
        System.out.println("ELSE");
        //else, need to check whether it is BC or AD
        if (date.length() >= 5 && onlyBeforeYearPattern.matcher(date.substring(0,
             5)).matches()) {//got a negative date
            System.out.println("Past Date");
            isBC = true;
            date = date.substring(1, date.length());//removed - sign infront of year
        }
        //then calculate date
        boolean isWeekNumber = false;
        String[] dateInfo = date.split("-");
        for (int i = 0; i < dateInfo.length; i++) {
            if (i == 0) {//this can only be a year
                //check year format
                if (onlyYearPattern.matcher(dateInfo[i]).matches()) {
                    if (isBC) {
                        year1 = dateInfo[i].replace("X", "9");
                    } else {
                        year1 = dateInfo[i].replace("X", "0");
                        //check if its all 0000, then its BC
                        if (year1.equals("0000")) {
                            year1 = "0001";//we will start from year 1 (AD)
                        }
                    }
```

```
            if (dateInfo[i].contains("X")) {//if we do have a range then we need
                 to set the values for the second date
                if (isBC) {
                    year2 = dateInfo[i].replace("X", "0");
                } else {
                    year2 = dateInfo[i].replace("X", "9");//could be the case
                         that we dont want it to point to the last day in the year

                }
                month2 = "12";//last day of the second year
                day2 = "31";//assuming a range for 1980s means 1980 to the last
                     day of 1989 (maximum possible range)
                //could have every day in january, would want it to end in
                     january?
            }
        }
    } else if (i == 1) {//this can be a week number, a month number or a season
        //checking if its a month
        System.out.println("Checking: " + dateInfo[i]);
        if (onlyMonthPattern.matcher(dateInfo[i]).matches()) {
            System.out.println("In onlyMonthPattern");
            month1 = dateInfo[i];
        } else if (onlyWeekNumberPattern.matcher(dateInfo[i]).matches())
             {//checking if its a week number
            isWeekNumber = true;
            //calculate month and start day-end
            //split W from actual week number
            if (isBC) {
                calendar.set(Calendar.ERA, GregorianCalendar.BC);
            } else {
                calendar.set(Calendar.ERA, GregorianCalendar.AD);
            }
            String weekNumber = dateInfo[i].substring(1);//W is the first part
                 of the string, after it is the week number
            calendar.set(Calendar.YEAR, getInt(year1));//should be set from
                 previously
```

```
            calendar.set(Calendar.WEEK_OF_YEAR, getInt(weekNumber));
            calendar.set(Calendar.DAY_OF_WEEK, Calendar.MONDAY);//assuming we
                start on monday and end on sunday
            year1 = new SimpleDateFormat("yyyy").format(calendar.getTime());//in
                the case the week goes into the next year, update our year1
            month1 = new SimpleDateFormat("MM").format(calendar.getTime());
            day1 = new SimpleDateFormat("dd").format(calendar.getTime());
            //now for the end of the week
            calendar.set(Calendar.DAY_OF_WEEK, Calendar.SUNDAY);
            month2 = new SimpleDateFormat("MM").format(calendar.getTime());
            day2 = new SimpleDateFormat("dd").format(calendar.getTime());
            year2 = new SimpleDateFormat("yyyy").format(calendar.getTime());
        } else if (onlySeasonPattern.matcher(dateInfo[i]).matches()) {//checking
            if its a season
            String season = dateInfo[i];
            Pair<String, String> seasonPair = seasonMap.get(season);//get the
                start and end month for the season
            if (seasonPair != null) {//year1 should be set previously
                //not quite sure what to do with BC here
                month1 = seasonPair.first;
                month2 = seasonPair.second;
                day2 = "31";//set the second day as we are using a range
                if (seasonPair.first.equals("12")) {//move onto the next year, so
                    year2 should be updated
                    year2 = (getInt(year1) + 1) + "";
                } else {//we dont need to increment the year as the season is
                    within the same year
                    year2 = year1;
                }
            }
        }
    } else if (i == 2) {//can be a day, or previously had week this could be
        weekend
        System.out.println("Checking: " + dateInfo[i] + "size: " +
            dateInfo[i].length());
```

```
if (isWeekNumber) {//then if it has another digit, its the day of the
     week (so its not a week, but a specific day of the week)
    System.out.println("Day of the week: " + dateInfo[i]);
    int day = getInt(dateInfo[i]);
    //as the calendar here starts with sunday and in iso it starts with
         monday, we need to increase by one and mod
    day = (day % 7) + 1;
    calendar.set(Calendar.DAY_OF_WEEK, day);
    year1 = new SimpleDateFormat("yyyy").format(calendar.getTime());//in
         the case the week goes into the next year, update our year1
    month1 = new SimpleDateFormat("MM").format(calendar.getTime());
    day1 = new SimpleDateFormat("dd").format(calendar.getTime());
    day2 = null;
    month2 = null;
    year2 = null;
}
if (onlyDayPattern.matcher(dateInfo[i]).matches()) {//got the day
    System.out.println("Got day: " + dateInfo[i]);
    day1 = dateInfo[i].substring(0, 2);
} else if (onlyWeekendPattern.matcher(dateInfo[i]).matches())
     {//previously should have had week number so its a range
    //checking its range has been set before
    if (day1 != null && day2 != null && month1 != null && month2 !=
         null) {
        //then lets refine the dates that were just before a week long,
             now down to a weekend
        if (isBC) {
            calendar.set(Calendar.ERA, GregorianCalendar.BC);
        } else {
            calendar.set(Calendar.ERA, GregorianCalendar.AD);
        }
        calendar.set(Calendar.DAY_OF_WEEK, Calendar.SATURDAY);
        year1 = new
             SimpleDateFormat("yyyy").format(calendar.getTime());//in the
             case the week goes into the next year, update our year1
        month1 = new SimpleDateFormat("MM").format(calendar.getTime());
```

```
                    day1 = new SimpleDateFormat("dd").format(calendar.getTime());
                    calendar.set(Calendar.DAY_OF_WEEK, Calendar.SUNDAY);
                    month2 = new SimpleDateFormat("MM").format(calendar.getTime());
                    day2 = new SimpleDateFormat("dd").format(calendar.getTime());
                    year2 = new SimpleDateFormat("yyyy").format(calendar.getTime());
                }
            }
        }
    }
    //make the dates
    Date date1;
    date1 = createDates(year1, month1, day1, isBC);
    toReturn.add(date1);
    System.out.println(date1);
    if (year2 != null && month2 != null && day2 != null) {
        Date date2 = createDates(year2, month2, day2, isBC);
        toReturn.add(date2);
        System.out.println(date2);
    }
    return toReturn;
}

/**
 * Creates a Data with the given input data. In the worst case we overestimated the day
     value (i.e. 31 when for
 * that month it can be 30), so we check that it is a legal date. If it isn't, then we
     reduce the day value,until we
 * get a correct day value. This is assuming that only the date values are wrong not
     month and years(these are given
 * by normalized entity tags, which should be correct according to SUTime annotator)
 *
 * @param year the year value for this Date
 * @param month the month value for this Date
 * @param day  the day value for this Date (which can be wrong, so we reduce it until
     its right, assuming we always overestimate.
```

```
     * @param isBC whether or not this is a BC or AD date; true if it is a BC Date.
     * @return a correct Date object based on the data passed in.
     */
    private Date createDates(String year, String month, String day, boolean isBC) {
        Date toReturn;
        String date = returnDate(year, month, day, isBC);
        System.out.println("Trying to create date for: " + date);
        try {
            toReturn = simpleDateFormat.parse(date);
        } catch (ParseException e) {
            //e.printStackTrace();//could comment this out
            System.out.println("Couldnt create date, so trying for a lower value");
            //couldnt create date, so most likely day value is to high, so reduce it
            int dayUpdate = Integer.parseInt(day);
            int monthUpdate;
            dayUpdate--;
            //should update month and year if fall below a certain threshold?
            if (dayUpdate <= 0) {//should decrease month val and set dayUpdate to 31
                dayUpdate = 31;
                System.out.println("Our day value fell below threshold, so go to month
                    before this");
                monthUpdate = Integer.parseInt(month);
                monthUpdate--;//assuming we will never produce a month below 1
                //which makes sense, as we only ever over estimate day values
                toReturn = createDates(year, Integer.toString(monthUpdate),
                    Integer.toString(dayUpdate), isBC);
            } else {
                //update day value, so try again with this value
                toReturn = createDates(year, month, Integer.toString(dayUpdate), isBC);
            }
        }
        System.out.println("Created date for: " + toReturn);
        return toReturn;
    }

    /**
```

```
     * Used to find update the min/max dates held. Will update the dates held if a new
         min/max has been found.
     * MinMax Algorithm.
     *
     * @param dateDurationPair a Pair that has a list of possible new min/max dates, and
         their corresponding duration data (which can be null).
     * @param date           the string that produced these dates.
     */
    private void enforceRule(Pair<ArrayList<Date>, String> dateDurationPair, String date) {
        if (dateDurationPair != null) {
            ArrayList<Date> newDates = dateDurationPair.first();
            for (Date newDate : newDates) {
                if (this.date1 == null || this.date1.compareTo(newDate) > 0) {//if we dont
                     have a date1, or we have a smaller one
                    this.date1 = newDate;
                    dateStr = date;
                    //else, we found a newDate that we havent set that is bigger than date1,
                         look at date2
                    durationData = dateDurationPair.second();
                } else if ((this.date2 == null || this.date2.compareTo(newDate) < 0) &&
                     !this.date1.equals(newDate)) {//if we dont have date2, or we found a
                     bigger date
                    this.date2 = newDate;                   //and the new date is not the
                         first date
                    dateStr = date;
                    durationData = dateDurationPair.second();
                }
            }
            updateRange();
        }
    }

    /**
     * Updates the int value of the range based on the date1 and date2 values. If date2 is
         null then the range is 0, ie
     * the size of the range of dates is 0.
```

```
     */
    private void updateRange() {
        if (date1 != null && date2 != null) {
            DateTime dateTime1 = new DateTime(date1);
            DateTime dateTime2 = new DateTime(date2);
            range = Days.daysBetween(dateTime1, dateTime2).getDays();
        } else {
            range = 0;
        }
    }

    /**
     * Get the Range, ie the number of days between date1 and date2.
     *
     * @return the number of days between date1 and date2 (0 if date2 == null)
     */
    public int getRange() {
        updateRange();
        return range;
    }

    /**
     * Produces a date String of the format yyyy-MM-dd, for the given input.
     *
     * @param year the year of the date
     * @param month the month of the date
     * @param day  the day of the date
     * @return a String of the format yyyy-MM-dd
     */
    private String returnDate(String year, String month, String day, boolean isBC) {
        if (isBC) {
            return String.format("%s-%s-%s BC", year, month, day);
        }
        return String.format("%s-%s-%s AD", year, month, day);
    }
```

```
    /**
     * On the given input, produce an integer.
     *
     * @param number a number in String form.
     * @return the integer value corresponding to the string, or 1 if it is not possible to
         produce an int.
     */
    private int getInt(String number) {
        try {
            return Integer.parseInt(number);
        } catch (Exception e) {
            return 1;
        }
    }

    /**
     * @return date1 -> date2, or just date1 if we don't have a date2
     */
    @Override
    public String toString() {
        String toReturn = "";
        SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd G");
        if (date1 != null) {
            toReturn += simpleDateFormat.format(date1);
        }
        if (date2 != null) {
            toReturn += " -> " + simpleDateFormat.format(date2);
        }
        if (durationData != null) {
            toReturn += String.format(" (%s)", durationData);
        }
        return toReturn;
    }

    /**
     * Compares two TimelineDates based on their start date.
```

```
     *
     * @param o the other backend.process.TimelineDate that is being compared to.
     * @return the comparison of this date1 to the other's date1, or -1 if the other result
          does not have a date1, or 1 if this does not have a date1.
     */
    @Override
    public int compareTo(TimelineDate o) {
        if (range == -1) {
            updateRange();
        }
/*        if (o.date1 != null && this.date1 != null) {
            return o.date1.compareTo(this.date1);
        }
        if (this.date1 == null) {
            return -1;
        }
        return 1;*/
        if (range > o.range) {
            return 1;
        } else if (range < o.range) {
            return -1;
        } else {
            return 0;
        }
    }

    /**
     * Checks whether a given backend.process.TimelineDate object is equal to this object.
     *
     * @param obj the other object we are comparing to.
     * @return true if the other backend.process.TimelineDate object has the same dates as
          this.
     */
    @Override
    public boolean equals(Object obj) {
        boolean toReturn;
```

```
    try {
        TimelineDate other = (TimelineDate) obj;
        toReturn = date1.equals(other.date1);
        if (toReturn) {
            if (date2 != null || other.date2 != null) {//if we have a scond date then so
                 should other, if we dont then they shouldnt either
                toReturn = date2.equals(other.date2);
            }
        }
    } catch (Exception e) {
        return false;
    }
    return toReturn;
}

/**
 * Set the Date for date1.
 *
 * @param date1 Date for date1.
 */
public void setDate1(Date date1) {
    this.date1 = date1;
    updateRange();
}

/**
 * Set the Date for date2.
 *
 * @param date2 Date for date2.
 */
public void setDate2(Date date2) {
    this.date2 = date2;
    updateRange();
}

/**
```

```
     * Get the Date for date1.
     *
     * @return Date for date1.
     */
    public Date getDate1() {
        return date1;
    }

    /**
     * Get the Date for date2.
     *
     * @return Date for date2.
     */
    public Date getDate2() {
        return date2;
    }

    /**
     * Get the additional duration data for a date.
     *
     * @return String representing the duration an event occurred for (every when it
         repeated).
     */
    public String getDurationData() {
        return durationData;
    }

    /**
     * Get date1 as a String in the format dd-MM-yyyy G, or null if that is not possible.
     *
     * @return a formatted String of date1.
     */
    public String getDate1FormattedDayMonthYear() {
        try {
            return (date1 != null) ? dayMonthYearFormat.format(date1) : null;
        } catch (Exception e) {
```

```
            return null;
        }
    }

    /**
     * Get date2 as a String in the format dd-MM-yyyy G, or null if that is not possible.
     *
     * @return a formatted String of date2.
     */
    public String getDate2FormattedDayMonthYear() {
        try {
            return (date2 != null) ? dayMonthYearFormat.format(date2) : null;
        } catch (Exception e) {
            return null;
        }
    }
}
```

C.4.3 Ranges

ProduceRanges.java

```
package backend.ranges;

import backend.process.Result;

import java.util.*;

/**
 * Produces a List of Range (that contain ranges - like a Tree, so it Returns a Forest) to
     use in showing the Timeline.
 */
public class ProduceRanges {
    private List<Range> trees = new ArrayList<>();

    /**
```

```
     * Get the list of Range Trees formed by processing the list of the Results.
     *
     * @return a list of Range Trees.
     */
    public List<Range> getTrees() {
        return trees;
    }

    /**
     * For the given Result list, sort them by size of range of dates held by their
         TimelineDate, then make a Forest of
     * Range Trees using the list of Results, and then finally sort the Forest by their
         date1 value (descending)
     *
     * @param resultList the given Result list.
     */
    public void produceRanges(List<Result> resultList) {
        System.out.println("Number of Results: " + resultList.size());
        sortByRange(resultList);//sort the list by their Range
        makeForest(resultList);//from the sorted list attempt to add it to existing Range
            Trees or create a new Tree if not possible.
        sortForest();//sort the Range (recursively) Forest by their date1 value
    }

    /**
     * For the given list of Results, sort them by the number of days in between their
         first range of dates and second.
     *
     * @param inputResults the given list of Results.
     * @return the list of Results sorted by their Range (descending)
     */
    private List<Result> sortByRange(List<Result> inputResults) {
        Collections.sort(inputResults);//java +7 mergesort with O(nlogn) but if its almost
            sorted its closer to O(n)
        Collections.reverse(inputResults);//as the list has been sorted in ascending order
            and we want descending
```

```
        System.out.println("Sorted by Range");
        for(Result result: inputResults){
            System.out.println(result+" "+result.getTimelineDate().getRange());
        }
        return inputResults;
    }

    /**
     * For the given sorted list of Results (has to be sorted for this algorithm to work
         properly as we are making the
     * Range trees starting from the highest Range) attempt to add each Result to existing
         Range trees, if that is not
     * possible then create a new Range tree for the Result.
     *
     * @param sortedResults the list of sorted Result objects.
     */
    private void makeForest(List<Result> sortedResults) {
        //if have to update parent, make copy of current, then reset, then set copy as child
        for (Result result : sortedResults) {//should be sorted from largest range to lowest
            //try to add to one of the trees if not possible make new range
            if (!addToRange(result)) {
                //we couldnt add it so we make a new Range
                Range range = new Range(result.getTimelineDate().getDate1(),
                    result.getTimelineDate().getDate2());
                range.add(result);
                trees.add(range);
            }
        }
    }

    /**
     * Attempt to add the given Result to the the different Range trees.
     *
     * @param result the given Result.
     * @return true if the Result was added to one of the existing Range trees; false
         otherwise.
```

```
     */
    private boolean addToRange(Result result) {
        for (Range range : trees) {
            if (range.add(result)) {
                return true;
            }
        }
        return false;
    }

    /**
     * For the forest of Range trees recursively sort the trees (and their subtrees) by
          their date1 in ascending order.
     */
    private void sortForest() {
        //sort the roots first
        Collections.sort(trees);
        //for all the roots call sortChildren (which will recursively sort)
        for (Range root : trees) {
            root.sortChildren();
        }
    }


}
```

Range.java

```
package backend.ranges;


import backend.process.Result;
import backend.process.TimelineDate;


import java.text.SimpleDateFormat;
import java.util.*;
```

```java
/**
 * A Range of Dates (which can be only one date) which can be represented as a Node in a
     Tree, due to it holding a list of
 * its sub-ranges.
 */
public class Range implements Cloneable, Comparable<Range> {
    private List<Result> results;//results that are exactly in this range
    private Date date1;//first part of range
    private Date date2;//only if we have a range of size > 0
    private List<Range> children;//the children in this range within constraint
         children.date1 >= date1 children.date1 < date2 && children.date2 <= date2

    public Range(Date date1, Date date2) {
        this.date1 = date1;
        this.date2 = date2;
        results = new ArrayList<>();
        children = new ArrayList<>();
    }

    /**
     * For the given Result, check that the dates of the Result are partially in the scope
         of this Range, if that is not
     * possible then return false.
     * If the Result is within this Rage, then attempt to find between the current Range
         and its children a Range where
     * the Results range of dates fit in perfectly, if it can be added then add it else
         attempt to create a new Range
     * child of this Range and find its location to place it.
     * <p>
     * Algorithm given by:
     * 1.check constraints, if they are false then return false else move onto step 2.
     * 2.attempt to add the Result to one of the children (or children-children, or etc) of
         this Range where they both
     * have the same Range; move onto step 3 if that is unsuccessful.
     * 3.create a new Range and find the location of where to add it in the Range
         (extending the Range where necessary)
```

```
     *
     * @param result the given Result.
     * @return true if the Result was added to this Range (i.e. this Tree) or its children
         or its children-children etc;
     * false otherwise.
     */
    public boolean add(Result result) {
        TimelineDate timelineDate = result.getTimelineDate();
        //check constraints
        if (!shouldAdd(timelineDate)) {//check constraints if we can even add to this range
             (ie are we in the root constraints)
            return false;
        }
        //attempt to add through an existing range
        Range toAdd = checkCanAdd(result);
        if (toAdd != null) {
            toAdd.results.add(result);//add to the results of the given range where we
                 could add
            return true;
        }
        //now we try to extend the range
        return createRangeAndAdd(result);
    }

    /**
     * Whether or not the given TimelineDate is partially or fully in the range of Dates of
         this Range. Whether or not
     * we should try to add the Result of this TimelineDate to this Range.
     *
     * @param timelineDate the given TimelineDate.
     * @return true if the TimelineDate is within/partially (ie if date1 is in the Range
         but not date2, or vice versa,
     * or both dates are within the range)in this Range; false otherwise.
     */
    private boolean shouldAdd(TimelineDate timelineDate) {
        //check the date1 is in constraint to t.date1 then if date2 then check with date2
```

```
    return isWithinConstraints(date1, date2, timelineDate.getDate1())
            || (timelineDate.getDate2() != null && isWithinConstraints(date1, date2,
                timelineDate.getDate2()));
}

/**
 * Attempt to find a Range (either this or one of its children, or its
     children-children, etc) that has the date
 * values that match the given Result.
 *
 * @param result the given Result.
 * @return a Range that holds the same exact date values as the TimelineDate of the
     Result or null if that is not
 * possible.
 */
private Range checkCanAdd(Result result) {
    TimelineDate timelineDate = result.getTimelineDate();
    if (timelineDate.getDate1().equals(date1) && ((date2 == null &&
         timelineDate.getDate2() == null)
            || (date2 != null && timelineDate.getDate2() != null &&
                date2.equals(timelineDate.getDate2())))) {
        return this;
    }
    Range toReturn = null;
    for (Range child : children) {
        toReturn = child.checkCanAdd(result);
        if (toReturn != null) {
            return toReturn;
        }
    }
    return toReturn;
}

/**
 * For the given Result, create a Range using its TimelineDate and attempt to find the
     location where this Range
```

```
     * should be added in the Tree structure (ie find the lowest/best possible Range that
         should contain the created
     * Range).
     * Normally called if we couldn't find any Range that has the same exact Dates so we
         create a Range to contain it
     * and attempt to find its spot (extending Ranges where necessary to self contain
         Ranges in bigger Ranges).
     *
     * @param result the given Result.
     * @return true if the created Range was added to this (sub-)tree, false otherwise.
     */
    private boolean createRangeAndAdd(Result result) {
        //we couldn't add it to any of the childrens so make new child and try to add it
        TimelineDate timelineDate = result.getTimelineDate();
        Range newChild = new Range(timelineDate.getDate1(), timelineDate.getDate2());
        newChild.add(result);//the new potential range child holds its result
        return addChild(newChild);// try and add this child (need to check constraints
    }



    /**
     * For the given Range (r') attempt to add it to this Range (r). For this to occur it
         must be within the range of dates of
     * this Range (i.e. r'.date1 >= r.date1 && r'.date1 <r.date2 && r'.date2 < r.date2) or
         it must overlap (i.e. date 1
     * is within the range, but date2 isn't or vice versa). In the latter case we extend
         this Range, such that it holds
     * the old range and the given range.
     *
     * @param potentialChild the given Range (r').
     * @return true if the potentialChild could be added to the children of this Range, or
         we could extend the Range;
     * false otherwise.
     */
    public boolean addChild(Range potentialChild) {
        /*
```

```
        if you are within the constraints
            then attempt to add it to the children,if it cant because there are no
                children, or it doesnt fit the childrens constraints then add it to this
                    Range's children
        check for partial constraints
            if it matches then extend the ranges and add it to the new extended range
                 (that holds as a child the old Range of this)
        otherwise return false (it doesn't belong in this area)
     */
    if (isWithinConstraints(date1, date2, potentialChild.date1)
            && (potentialChild.date2 == null || isWithinConstraints(date1, date2,
                 potentialChild.date2))) {
        //we are within the constraints, try to add to children Ranges, if cant because
             there are none, or we cant then add directly to this Ranges children
        for (Range child : children) {
            if (child.addChild(potentialChild)) {//successfully added to one of them
                return true;
            }
        }
        //we couldnt add to any of them, then add directly to my children
        children.add(potentialChild);
        return true;
        //partial constraints
    } else if (isWithinConstraints(date1, date2, potentialChild.date1) &&
         (potentialChild.date2 != null && !isWithinConstraints(date1, date2,
         potentialChild.date2))) {
        //checking if we need to extend the range (this case the first date is within
             the constraints but date2 isnt
        //date1 is in the range but date2 isnt so extend and us to the extensions
             children as we didnt fit in previous range (which is now its child)
        extendRange(date1, potentialChild.date2);//extend the range and add the child
             to the now extended range (as we fit in the now extended
        children.add(potentialChild);//range, but not in its other child which was the
             previous range held before extending, else we wouldnt be here)
        return true;//if we called addChild it would just loop on the child that was
             the previous range, and thats the only child so there is no point checking
```

```
            just add to this
    } else if (!isWithinConstraints(date1, date2, potentialChild.date1) &&
         (potentialChild.date2 != null && isWithinConstraints(date1, date2,
         potentialChild.date2))) {
        //date2 is in the range but date1 isnt so extend and add to the children of the
            extended range.
        extendRange(potentialChild.date1, date2);
        children.add(potentialChild);
        return true;
    }
    //We werent able to add the potentialRange to this Range,
    return false;//so return false (this method is recursive, so it will try on the
         next child of the parent Range that called it on this child)
}

/**
 * Used to extend the current Range to the given Dates, and hold the previous data in a
     new Range as a child.
 *
 * @param date1 the start of the extended Range.
 * @param date2 the end of the extended Range.
 */
private void extendRange(Date date1, Date date2) {
    try {
        Range cloneRange = (Range) clone();//the date of this will be a new child, we
            are setting its parent
        clear();//clear all the values and set the new ones
        this.date1 = date1;//extended range value
        this.date2 = date2;//extended range value
        children.add(cloneRange);//add what used to be this (but now we cloned it) as
            out child
    } catch (CloneNotSupportedException e) {
        e.printStackTrace();
    }
}
```

```
    /**
     * For the given Date, return a String of the format: dd-MM-yyyy G (days-months-years
         ERA)
     *
     * @param date the given Date.
     * @return a String of the format dd-MM-yyyy G or null if the given date can't be
         formatted (i.e. is null)
     */
    private String printDate(Date date) {
        SimpleDateFormat simpleDateFormat = new SimpleDateFormat("dd-MM-yyyy G");
        try {
            return simpleDateFormat.format(date);
        } catch (Exception e) {
        }
        return null;
    }

    /**
     * Clear all the data held by the current Range.
     */
    private void clear() {
        date1 = null;
        date2 = null;
        children = new ArrayList<>();
        results = new ArrayList<>();
    }

    /**
     * Check that the given Date toCheck is within the other two given Dates constraint1
         and constraint2 (both inclusive)
     *
     * @param constraint1 the given start constraint Date (inclusive).
     * @param constraint2 the given end constraint Date (inclusive).
     * @param toCheck   the given Date we are checking.
     * @return true if the given Date to check is within the two constraints, or if any of
         them are null equal to the other;
```

```
     * false otherwise.
     */
    private boolean isWithinConstraints(Date constraint1, Date constraint2, Date toCheck) {
        if (constraint1 != null && constraint2 != null) {
            return toCheck.compareTo(constraint1) >= 0 && toCheck.compareTo(constraint2) <=
                0;
        }
        if (constraint1 == null && constraint2 != null) {
            return toCheck.equals(constraint2);
        }
        if (constraint1 != null) {
            return toCheck.equals(constraint1);
        }
        return false;
    }

    /**
     * Return a clone of this Range. Holds the same data.
     *
     * @return an Object that can be cast to a Range object, which holds the current Data
         of the Range that called it
     * (when clone() was called).
     * @throws CloneNotSupportedException
     */
    @Override
    protected Object clone() throws CloneNotSupportedException {
        super.clone();
        Range cloneRange = new Range(date1, date2);
        cloneRange.children = new ArrayList<>(children);
        cloneRange.results = new ArrayList<>(results);
        return cloneRange;
    }

    /**
     * Returns a String of the given Range.
     *
```

```
     * @return a String that shows the range of Dates held by this Range, its Result and
         the its child Ranges (recursive).
     */
    @Override
    public String toString() {
        String toReturn = printDate(date1);
        if (date2 != null) {
            toReturn += " -> " + printDate(date2);
        }
        toReturn += " has results: " + results;
        toReturn += " and has children: \n" + children;
        return toReturn;
    }

    /**
     * Overwrote the equals method to check whether two Ranges are equal to each other:
         that is they hold the same Dates,
     * results and their children are equal (recursive).
     *
     * @param obj the other Range we are checking if its equal to this Range.
     * @return true if the other Range has the same Dates as this Range, holds the same
         Results as this Range, and their
     * children are equal.
     */
    @Override
    public boolean equals(Object obj) {
        try {
            Range other = (Range) obj;
            if (date1.equals(other.date1) && ((date2 == null & other.date2 == null) ||
                 (date2 != null & other.date2 != null && date2.equals(other.date2)))) {
                if (children.isEmpty() && other.children.isEmpty()) {
                    return results.equals(other.results);
                } else {//they have children
                    return results.equals(other.results) && children.equals(other.children);
                }
            }
```

```
    } catch (Exception e) {
        return false;
    }
    return false;
}



/**
 * Sort the children held by this Range, and calls sortChildren on its children.
 * Sorts them such that the resulting children list is in descending order based on
     date1 (ie at index 0 you have
 * the newest range and at the last index the range with date1 furthest away.
 */
public void sortChildren() {
    if (children.size() > 0) {
        Collections.sort(children);
        for (Range range : children) {
            range.sortChildren();
        }
    }
}



/**
 * Implemented the Comparable interface to sort Ranges.
 *
 * @param o the other Range we are comparing to.
 * @return 1 if the date1 of this Range is greater than the other, 0 if they are equal
     and -1 if this Range's date1
 * is less than the other.
 */
@Override
public int compareTo(Range o) {
    if (o.date1 != null && this.date1 != null) {
        return date1.compareTo(o.date1);
    }
```

```
        if (this.date1 == null) {
            return -1;
        }
        return 1;
    }

    /**
     * Get the list of Range children held by this Range.
     *
     * @return the list of Range children.
     */
    public List<Range> getChildren() {
        return children;
    }

    /**
     * Get a String representation of date1 and date2 (if set) of this Range (i.e. a String
         of the range of dates held
     * by this Range).
     *
     * @return a String representation of the range of dates held.
     */
    public String getDateRange() {
        String toReturn = printDate(date1);
        if (date2 != null) {
            toReturn += " -> " + printDate(date2);
        }
        return toReturn;
    }

    /**
     * A list of the Results held by this Range. These are Results that have the same exact
         TimelineDate range of dates
     * as this Range (ie date1 and date2 of the Results and this Range are equal).
     *
     * @return the list of Results held by this Range.
```

```
     */
    public List<Result> getResults() {
        return results;
    }
}
```

C.4.4 System

BackEndSystem.java

```
package backend.system;


import edu.stanford.nlp.pipeline.StanfordCoreNLP;
import edu.stanford.nlp.util.PropertiesUtils;


/**
 * Holds all the data needed by the entire Back-end: the StanfordCoreNLP used to process
     text, the System state, etc.
 * Follows a Singleton design pattern, as there should be just one
     backend.system.BackEndSystem during the entire lifetime of the
 * system, and allows to hold data needed in different areas to be held at one place
     accessible to all areas.
 * Before the Object is created the backend.system.SystemState is NOT_STARTED, it moves to
     STARTED when it is created.
 */
public class BackEndSystem {
    private static BackEndSystem ourInstance = new BackEndSystem();
    private StanfordCoreNLP coreNLP;
    private SystemState systemState = SystemState.NOT_STARTED;
    private Settings settings;


    /**
     * Will make a new backend.system.BackEndSystem if one has not been created yet, or
         else return the one that was previously
     * created.
     *
```

```
     * @return a backend.system.BackEndSystem that was just created or was previously
         created.
     */
    public static BackEndSystem getInstance() {
        return ourInstance;
    }

    /**
     * Initialises the StanfordCoreNLP and sets the backend.system.SystemState to STARTED.
     */
    private BackEndSystem() {
        coreNLP = new StanfordCoreNLP(PropertiesUtils.asProperties(
                "annotators",
                "tokenize,ssplit,pos,lemma,ner,entitymentions,parse,dcoref",
                "tokenize.language", "en"
        ));
        System.out.println("Finished running");
        systemState = SystemState.STARTED;
        settings = new Settings(true);//load the settings of the file, or use default
            Settings
    }

    /**
     * Get a reference to the StandfordCoreNLP that has already been loaded with all the
         models.
     *
     * @return a StanfordCoreNLP with all the models loaded.
     */
    public StanfordCoreNLP getCoreNLP() {
        return coreNLP;
    }

    /**
     * Get the current state of the System according to the backend.system.SystemState enum.
     *
     * @return the current state of the System.
```

```
     */
    public SystemState getSystemState() {
        return systemState;
    }

    /**
     * Update the current state of the System, i.e. when the we are beginning to process
         Files the backend.system.SystemState should be
     * set to PROCESSING.
     *
     * @param systemState the state to which the System should be set to.
     */
    public void setSystemState(SystemState systemState) {
        this.systemState = systemState;
        System.out.println("System is now in state: " + systemState);
    }

    /**
     * Get the Settings that this System uses (i.e preferred width/height on startup, how
         many threads to run in parallel, etc).
     *
     * @return the Settings used by this System.
     */
    public Settings getSettings() {
        return settings;
    }

    /**
     * Set the Settings that this System uses.
     *
     * @param settings the Settings used by this System.
     */
    public void setSettings(Settings settings) {
        this.settings = settings;
        this.settings.saveSettingsFile();
    }
```

```
}
```

Settings.java

```
package backend.system;



import com.google.gson.*;
import com.google.gson.annotations.Expose;

import java.io.*;
import java.net.URISyntaxException;
import java.util.Scanner;

/**
 * Class that represents the possible Settings of the System.
 * Settings are attempted to be retrieved and/or stored in a Settings file, where the data
     is encoded in JSON.
 */
public class Settings implements Cloneable {
    //default values (to reset to default, or if settings file not found)
    public final static int defaultMaxNoThreads = 2;
    public final static int defaultThresholdSummary = 10;
    public final static int defaultWidth = 1024;
    public final static int defaultHeight = 800;
    private final static String threadTag = "maxNoOfThreads";
    private final static String thresholdTag = "thresholdSummary";
    private final static String widthTag = "width";
    private final static String heightTag = "height";
    //actual values
    @Expose(deserialize = false)
    private int maxNoOfThreads;
    @Expose(deserialize = false)
    private int thresholdSummary;
    @Expose(deserialize = false)
    private int width;
```

```
@Expose(deserialize = false)
private int height;

/**
 * Used to create an instance of the Settings of this System, by attempting to load the
     Settings file.
 * If this can't be done the default values of the Settings are used to populate the
     fields.
 *
 * @param loadFromFile whether or not the Settings File should be loaded.
 */
public Settings(boolean loadFromFile) {
    //try to read from settings file, if not possible, then use default values
    if (!loadFromFile || !loadSettingsFile()) {
        reset();
    }
}

/**
 * Used to create an instance of the Settings of this System, where the fields are
     populated by the default values.
 */
public Settings() {
    this(false);
}

/**
 * Attempt to load the Settings from the Settings file.
 *
 * @return whether or not the Settings file could be retrieved and the fields of the
     Settings be populated.
 */
private boolean loadSettingsFile() {
    System.out.println("Trying to load from file");
    //load the file, return true if it could
    try {
```

```
            File settingsFile = new File("settings.ini");
            //now need to read settings file
            System.out.println("Could find settings.ini file");
            String text = getJSONString(settingsFile);
            System.out.println("String in file: " + text);
            JsonElement jsonElement = new JsonParser().parse(text);
            JsonObject jsonObject = jsonElement.getAsJsonObject();
            System.out.println(jsonObject);
            setValues(jsonObject);
        } catch (Exception e) {
            return false;
        }
        return true;
    }

    /**
     * For the given File attempt to retrieve the JSON stored in it.
     *
     * @param file the given File (which should have the JSON settings data).
     * @return a String of the JSON encoding of the Settings in the File.
     * @throws FileNotFoundException thrown if the File could not be found.
     */
    private String getJSONString(File file) throws FileNotFoundException {
        Scanner scanner = new Scanner(file);
        StringBuilder stringBuilder = new StringBuilder();
        while (scanner.hasNext()) {
            stringBuilder.append(scanner.nextLine());
        }
        scanner.close();
        return stringBuilder.toString();
    }

    /**
     * For the given JsonObject, populate the Settings field with its data, if its present,
         else use the default values.
     *
```

```
     * @param jsonObject the given JsonObject that should have the Settings data.
     */
    private void setValues(JsonObject jsonObject) {
        maxNoOfThreads = (jsonObject.get(threadTag) != null) ?
             jsonObject.get(threadTag).getAsInt() : defaultMaxNoThreads;
        thresholdSummary = (jsonObject.get(thresholdTag) != null) ?
             jsonObject.get(thresholdTag).getAsInt() : defaultThresholdSummary;
        width = (jsonObject.get(widthTag) != null) ? jsonObject.get(widthTag).getAsInt() :
             defaultWidth;
        height = (jsonObject.get(heightTag) != null) ? jsonObject.get(heightTag).getAsInt()
             : defaultHeight;
        if (!isConstrained()) {//if constraints have not been set then reset
            reset();
        }
    }



    /**
     * For the Settings fields in this case, store them in JSON format in the Settings File.
     *
     * @return whether or not the Settings of this File could be saved in a File.
     */
    public boolean saveSettingsFile() {
        final GsonBuilder gsonBuilder = new GsonBuilder();
        gsonBuilder.excludeFieldsWithoutExposeAnnotation();
        final Gson gson = gsonBuilder.create();
        try {
            String json = gson.toJson(this);
            System.out.println(json);
            saveToSettingsFile(json);
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
        return true;
    }
```

```
/**
 * Save the given text in the "settings.ini" File.
 *
 * @param text the given Text
 * @throws IOException if an error occurs while accessing the File
 */
private void saveToSettingsFile(String text) throws IOException {
    PrintWriter writer = new PrintWriter("settings.ini");
    writer.println(text);
    writer.close();
}

/**
 * For all the Fields in the Settings class, set their values to the default values.
 */
public void reset() {
    //resets all the values to default
    maxNoOfThreads = defaultMaxNoThreads;
    thresholdSummary = defaultThresholdSummary;
    width = defaultWidth;
    height = defaultHeight;
}

/**
 * Getter for the Max Number of Threads that are allowed to run in parallel in this
     System.
 *
 * @return the Max Number of Threads that are allowed to run in parallel in this System.
 */
public int getMaxNoOfThreads() {
    return maxNoOfThreads;
}

/**
```

```
     * Setter for the Max Number of Threads that are allowed to run in parallel in this
         System.
     *
     * @param maxNoOfThreads the Max Number of Threads that are allowed to run in parallel
         in this System.
     */
    public void setMaxNoOfThreads(int maxNoOfThreads) {
        if (isMaxNoOfThreadsConstrained(maxNoOfThreads)) {
            this.maxNoOfThreads = maxNoOfThreads;
        }
    }

    /**
     * Getter for the Threshold used when attempting to produce a Summary of a Sentence.
     *
     * @return the Threshold used when attempting to produce a Summary of a Sentence.
     */
    public int getThresholdSummary() {
        return thresholdSummary;
    }

    /**
     * Setter for the Threshold used when attempting to produce a Summary of a Sentence.
     *
     * @param thresholdSummary the Threshold used when attempting to produce a Summary of a
         Sentence.
     */
    public void setThresholdSummary(int thresholdSummary) {
        if (isThresholdConstrained(thresholdSummary)) {
            this.thresholdSummary = thresholdSummary;
        }
    }

    /**
     * Getter for the preferred start up width of the Program window.
     *
```

```
     * @return the preferred start up width of the Program window.
     */
    public int getWidth() {
        return width;
    }

    /**
     * Setter for the preferred start up width of the Program window.
     *
     * @param width the preferred start up width of the Program window.
     */
    public void setWidth(int width) {
        if (isWidthConstrained(width)) {
            this.width = width;
        }
    }

    /**
     * Getter for the preferred start up height of the Program window.
     *
     * @return the preferred start up height of the Program window.
     */
    public int getHeight() {
        return height;
    }

    /**
     * Setter for the preferred start up height of the Program window.
     *
     * @param height the preferred start up height of the Program window.
     */
    public void setHeight(int height) {
        if (isHeightConstrained(height)) {
            this.height = height;
        }
    }
```

```
/**
 * Produce a clone of this Object, such that the populated data in the clone is a clone
     of the data in this Object.
 *
 * @return a Clone of this Object (data values are the same but stored in a different
     area in memory)
 * @throws CloneNotSupportedException
 */
@Override
public Object clone() throws CloneNotSupportedException {
    super.clone();
    Settings clonedSettings = new Settings();
    clonedSettings.setHeight(height);
    clonedSettings.setMaxNoOfThreads(maxNoOfThreads);
    clonedSettings.setThresholdSummary(thresholdSummary);
    clonedSettings.setWidth(width);
    return clonedSettings;
}

/**
 * Whether or not the given threshold summary is within the constraints.
 *
 * @param thresholdSummary the given threshold summary
 * @return true if the threshold summary is within the constraints; false otherwise.
 */
private boolean isThresholdConstrained(int thresholdSummary) {
    return thresholdSummary > 0 && thresholdSummary <= 20;
}

/**
 * Whether or not the given max number of threads is within the constraints.
 *
 * @param maxNoOfThreads the given max number of threads.
 * @return true if the max number of threads is within the constraints; false otherwise.
 */
```

```
private boolean isMaxNoOfThreadsConstrained(int maxNoOfThreads) {
    return maxNoOfThreads > 0 && maxNoOfThreads <= 20;
}

/**
 * Whether or not the given width is within the constraints.
 *
 * @param width the given width.
 * @return true if the width is within the constraints; false otherwise.
 */
private boolean isWidthConstrained(int width) {
    return width > 0 && width <= 1920;
}

/**
 * Whether or not the given height is within the constraints.
 *
 * @param height the given height.
 * @return true if the height is within the constraints; false otherwise.
 */
private boolean isHeightConstrained(int height) {
    return height > 0 && height <= 1080;
}

/**
 * Whether or not all the fields in this Settings object are all within their
     constraints.
 *
 * @return true if all the fields are within their constraints; false otherwise.
 */
private boolean isConstrained() {
    return isThresholdConstrained(thresholdSummary) &&
        isMaxNoOfThreadsConstrained(maxNoOfThreads)
           && isWidthConstrained(width) && isHeightConstrained(height);
}
```

```
}
```

SystemState.java

```
package backend.system;


/**
 * Represents all the different states the Back-end (specifically the
     backend.process.Engine) can be in.
 */
public enum SystemState {
    NOT_STARTED,//initially when the StanfordCoreNLP has not been started
    STARTED,//when the StanfordCoreNLP has been started
    PROCESSING,//when the back-end is processing files
    PROCESSED,//when it has processed all files
    FINISHED//when it has returned the Results of processing the Files
}
```

C.5 src/main/java/frontend

C.5.1 Controllers

CustomResultRowController.java

```
package frontend.controllers;


import backend.process.FileData;
import backend.process.Result;
import frontend.dialogs.EditEventDialog;
import frontend.observers.DocumentReaderObserver;
import frontend.observers.TimelineRowObserver;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
```

```
import javafx.scene.control.Button;
import javafx.scene.control.Dialog;
import javafx.scene.control.Label;
import javafx.scene.layout.Pane;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

import java.io.IOException;
import java.util.Optional;
import java.util.function.Consumer;

/**
 * Controller for the custom Result rows in the ListView of the RangeData layout.
 */
public class CustomResultRowController {
    @FXML
    private VBox rootVBox;
    @FXML
    private Button viewButton;
    @FXML
    private Button editButton;
    @FXML
    private Label subjectsLabel;
    @FXML
    private Label eventLabel;
    @FXML
    private Label fromLabel;
    private Result result;
    private TimelineRowObserver timelineRowObserver;
    private int rangePosition;
    /**
     * For the given Result, set up the data to be shown for this row, and the onclick
         events.
     * Builds a layout that shows the subjects, and the event of the given Result, along
         with buttons to view the
     * original document that produced this Result and the edit dialog to edit the Result.
```

```
     *
     * @param result the given Result.
     */
    public CustomResultRowController(Result result, TimelineRowObserver
         timelineRowObserver, int rangePosition) {
        this.result = result;
        this.timelineRowObserver = timelineRowObserver;
        this.rangePosition = rangePosition;
        FXMLLoader fxmlLoader = new
             FXMLLoader(getClass().getResource("customResultRow.fxml"));
        fxmlLoader.setController(this);
        try {
            rootVBox = fxmlLoader.load();//set the root layout
            setUpData();//set the data
            setUpOnClicks();//set the onclicks
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * Set the data held by the Result in the appropriate labels.
     */
    private void setUpData() {
        if (result != null) {
            subjectsLabel.setText(result.getSubjectsAsString());
            eventLabel.setText(result.getEvent());
            if(result.getFileData() != null){
                FileData fileData = result.getFileData();
                fromLabel.setText(fileData.getFileName()+"
                     ("+fileData.getCreationDateFormattedDayMonthYear()+")");
            }
        }
    }

    /**
```

```
 * Set the EventHandlers for the onClicks of the buttons of this Result row.
 */
private void setUpOnClicks() {
    viewButton.setOnAction(new EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent event) {
            System.out.println("Go to Document button for timeline event: " +
                 result.getTimelineDate() + " has been pressed");
            Stage stage = new Stage();
            DocumentReaderController documentReaderController = new
                 DocumentReaderController(result, new DocumentReaderObserver() {
                @Override
                public void close() {
                    System.out.println("Closing document reader window");
                    stage.close();
                }
            });
            Pane rootLayout = documentReaderController.getRootBorderPane();
            if (rootLayout != null) {
                stage.setScene(new Scene(documentReaderController.getRootBorderPane(),
                     1024, 800));
                stage.setTitle("Document Reader - " +
                     result.getFileData().getFileName());
                stage.show();
            }
        }
    });

    editButton.setOnAction(new EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent event) {
            System.out.println("Clicked Event on: " + result);
            Dialog dialog = EditEventDialog.getEditEventDialog(result, (rangePosition +
                 1));
            Optional<EditEventDialog.DialogResult> response = dialog.showAndWait();
            response.ifPresent(new Consumer<EditEventDialog.DialogResult>() {
```

```
                @Override
                public void accept(EditEventDialog.DialogResult dialogResult) {
                    if (dialogResult.getResultType() ==
                            EditEventDialog.DialogResult.ResultType.DELETE) {
                        System.out.println("Delete the event");
                        timelineRowObserver.delete(result);
                    } else if (dialogResult.getResultType() ==
                            EditEventDialog.DialogResult.ResultType.SAVE) {
                        System.out.println("Update the timeline");
                        Result copy = dialogResult.getResult();
                        timelineRowObserver.update(result, copy);
                    } else if (dialogResult.getResultType() ==
                            EditEventDialog.DialogResult.ResultType.CANCEL) {
                        System.out.println("Dont do anything");
                    }
                }
            });
        }
    });
  }

  /**
   * Get the root layout represented by this Controller (i.e. labels showing the subjects
       and events held by this
   * Result, and buttons to view the document that produced the Result and edit the
       Result.
   *
   * @return the root layout represented by this Controller.
   */
  public Pane getRootLayout() {
      return rootVBox;
  }
}
```

CustomTimelineRow.java

```java
package frontend.controllers;

import backend.ranges.Range;
import frontend.observers.TimelineRowObserver;
import javafx.geometry.HPos;
import javafx.geometry.Orientation;
import javafx.geometry.VPos;
import javafx.scene.control.Separator;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.Pane;

import java.util.ArrayList;
import java.util.List;

/**
 * Class for the Custom Rows in the Timeline. Builds a row made of a GridPane with the
     root Range (given in the
 * constructor) and recursively adds to the other coloumn the Ranges below this Range.
 */
public class CustomTimelineRow {
    private Range root;
    private GridPane rootLayout;
    private TimelineRowObserver timelineRowObserver;

    /**
     * Constructor used to set up the layout of the Timeline row. Afterwards a call to
         getPane() will return the row
     * to be added to the Timeline.
     *
     * @param root the root Range (which can hold Ranges as children which are added
         Recursively to the 2nd column of
     *            the GridPane.
     */
    public CustomTimelineRow(Range root, TimelineRowObserver timelineRowObserver) {
        this.timelineRowObserver = timelineRowObserver;
```

```
        this.root = root;
        setUpLayout();//set up the layout of this Timeline Row
    }

    /**
     * Called to create the Row based on the given Range (given to the constructor). The
         RootLayout (a root GridPane)
     * will have then been set.
     */
    private void setUpLayout() {
        ArrayList<Range> list = new ArrayList<>();
        list.add(root);
        rootLayout = getGridPane(list, false);//get the root GridPane
        rootLayout.setGridLinesVisible(true);//add grid lines to the root only (for optical
            purposes)
    }

    /**
     * For the given list of Range objects, add them to the first column of the GridPane
         (that will be returned), if we
     * need to add the vertical Separator (given by addVerticalSep) add it to the second
         column of the GridPane. If we
     * have multiple Ranges in the RangeList then they are separated by a horizontal
         Separator.
     * Recursively adds to the children of each Range in the list, in the third column of
         the row of their Parent, this
     * way the Parent contains all of its children in its row (and the children do the same
         for their children, etc).
     * <p>
     * Range data in column 1 and row i (given by their index in the list).
     * Children Range's (if they have) in column 2 and row i (same as parent).
     *
     * @param rangeList    the given list of Range objects.
     * @param addVerticalSep whether or not we need to add a Vertical Separator
     * @return the GridPane built based on the given list of Range objects.
     */
```

```
private GridPane getGridPane(List<Range> rangeList, boolean addVerticalSep) {
    GridPane gridPane = loadGridPane();//get the root GridPane layout that we are
        adding to
    if (gridPane != null) {
        for (int i = 0; i < 2 * rangeList.size() - 1; i++) {
            //add a separator between them
            if (i % 2 == 1) {
                Separator separator = new Separator(Orientation.HORIZONTAL);
                gridPane.add(separator, 0, i, GridPane.REMAINING, 1);
            } else {
                //layout for this Range
                Range range = rangeList.get(i / 2);
                Pane toAdd = rangeDataLayout(range, (i / 2));//add the layout for this
                    given Range
                gridPane.add(toAdd, 0, i);
                GridPane.setValignment(toAdd, VPos.TOP);//set the items added to top
                    left of the layout
                GridPane.setHalignment(toAdd, HPos.LEFT);
                if (range.getChildren().size() > 0) {//if this Range has children, then
                    build their layout (recursively) and add them
                    if (addVerticalSep) {//we need to add a separator
                        Separator separator = new Separator(Orientation.VERTICAL);
                        gridPane.add(separator, 1, i);//add the separator
                    }
                    GridPane gridPane1 = getGridPane(range.getChildren(), true);
                    gridPane.add(gridPane1, 2, i);
                }
            }
        }
    }
    return gridPane;//return the gridpane layout we added to
}

/**
 * Get the layout to show the data for the given Range (ie its Date range and its
     Results)
```

```
     *
     * @param range the given Range.
     * @return the layout representing the data of the given Range.
     */
    private Pane rangeDataLayout(Range range, int position) {
        RangeDataController rangeDataController = new RangeDataController(range,
            timelineRowObserver, position);
        return rangeDataController.getRootBorderPane();
    }

    /**
     * Get the base GridPane used to add a Range's data and its Children data
         (recursively). The GridPane is of the size
     * of its content, but it can be increased to any size.
     *
     * @return a base GridPane used to add a Range's data and its Children data.
     */
    private GridPane loadGridPane() {
        GridPane gridPane = new GridPane();
        gridPane.setPrefSize(GridPane.USE_COMPUTED_SIZE, GridPane.USE_COMPUTED_SIZE);
        gridPane.setMinSize(GridPane.USE_PREF_SIZE, GridPane.USE_PREF_SIZE);
        gridPane.setMaxSize(Double.MAX_VALUE, Double.MAX_VALUE);
        return gridPane;
    }



    /**
     * Get the root layout of this row in the Timeline (i.e. a GridPane that holds the
         information of the Root range,
     * passed in the timeline, in the first column with the Roots childrens information in
         the second column, and them
     * holding their children in their columns and so on).
     *
     * @return the root layout of a row in the Timeline given by the Range passed into the
         constructor.
     */
```

```
    public Pane getPane() {
        return rootLayout;
    }

}
```

DocumentLoadedRowController.java

```
package frontend.controllers;

import backend.process.FileData;
import frontend.observers.DocumentsLoadedObserver;
import javafx.event.EventHandler;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.control.Label;
import javafx.scene.image.ImageView;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.GridPane;

import java.io.IOException;

/**
 * Controller for each row in the Documents Loaded listview.
 */
public class DocumentLoadedRowController {
    @FXML
    private ImageView removingImageView;
    @FXML
    private Label documentLabel;
    @FXML
    private GridPane gridPane;
    private DocumentsLoadedObserver documentsLoadedObserver;

    /**
```

```
     * Creates a Controller and loads the layout for a row in the Loaded Documents
         listview. Observer is informed when
     * that file needs to be removed.
     *
     * @param fileData             FileData for which this row is made for.
     * @param documentsLoadedObserver observer that gets notified when the Results for the
         given FileData need to be removed.
     */
    public DocumentLoadedRowController(FileData fileData, DocumentsLoadedObserver
         documentsLoadedObserver) {
        this.documentsLoadedObserver = documentsLoadedObserver;
        FXMLLoader fxmlLoader = new
             FXMLLoader(getClass().getResource("documentLoadedRow.fxml"));
        fxmlLoader.setController(this);
        try {
            fxmlLoader.load();
        } catch (IOException e) {
            e.printStackTrace();
        }
        setData(fileData);
        removingImageView.setOnMouseClicked(new EventHandler<MouseEvent>() {
            @Override
            public void handle(MouseEvent event) {
                System.out.println("Remove this Document: " + fileData);
                documentsLoadedObserver.remove(fileData);
            }
        });
    }

    /**
     * For the label that is supposed to show the File name, set it.
     *
     * @param fileData label for which we set the File name.
     */
    private void setData(FileData fileData) {
```

```
        documentLabel.setText(fileData.getFileName()+"
             ("+fileData.getCreationDateFormattedDayMonthYear()+")");
    }

    /**
     * Get the root layout for this row (i.e. its GridPane, under which everything is set).
     *
     * @return the root GridPane layout for this row.
     */
    public GridPane getGridPane() {
        return gridPane;
    }
}
```

DocumentReaderController.java

```
package frontend.controllers;

import backend.process.FileData;
import backend.process.ProcessFiles;
import backend.process.Result;
import frontend.observers.DocumentReaderObserver;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.control.Alert;
import javafx.scene.control.ButtonType;
import javafx.scene.control.ContextMenu;
import javafx.scene.control.MenuItem;
import javafx.scene.layout.BorderPane;
import org.fxmisc.richtext.InlineCssTextArea;

import java.awt.*;
```

```
import java.awt.datatransfer.StringSelection;
import java.io.File;
import java.io.IOException;

/**
 * Controller for the layout of the Document Reader
 */
public class DocumentReaderController {
    @FXML
    private InlineCssTextArea documentInlineCssTextArea;
    @FXML
    private MenuItem closeMenuItem;
    @FXML
    private MenuItem copyMenuItem;
    @FXML
    private BorderPane rootBorderPane;
    private DocumentReaderObserver documentReaderObserver;

    /**
     * Called to create the layout for the Document Reader. It creates a text area with the
         text of the File where the
     * given Result originates from, and it highlights the specific sentence that produced
         the given Result.
     * The Observer is used to inform the creator of the window that uses this layout, to
         inform them when the Close
     * menu item was pressed (to close the window).
     *
     * @param result              the given Result.
     * @param documentReaderObserver the Observer that holds this layout, to inform when to
         close the window.
     */
    public DocumentReaderController(Result result, DocumentReaderObserver
        documentReaderObserver) {
        this.documentReaderObserver = documentReaderObserver;
        FXMLLoader fxmlLoader = new
            FXMLLoader(getClass().getResource("documentReader.fxml"));
```

```
        fxmlLoader.setController(this);
        try {
            fxmlLoader.load();
        } catch (IOException e) {
            e.printStackTrace();
        }
        setData(result);
        setOnActionListeners();
    }


    /**
     * Called to set the data displayed in the layout, which in this case is the text of
         the File set in the text area
     * with the relevant sentence that produced the given result highlighted.
     *
     * @param result the given Result (used to determine the File and sentence that
         produced this Result).
     */
    private void setData(Result result) {
        String stringInFile;
        if ((stringInFile = getStringInFile(result.getFileData())) != null) {
            documentInlineCssTextArea.clear();
            documentInlineCssTextArea.replaceText(stringInFile);
            ContextMenu contextMenu = new ContextMenu();
            MenuItem menuItemCopy = new MenuItem("Copy");
            menuItemCopy.setDisable(true);//initially cant copy as no text is selected
            copyMenuItem.setDisable(true);
            menuItemCopy.setOnAction(new EventHandler<ActionEvent>() {
                @Override
                public void handle(ActionEvent event) {
                    copy();//copy the selected text
                }
            });
            contextMenu.getItems().setAll(menuItemCopy);
```

```
        documentInlineCssTextArea.selectedTextProperty().addListener(new
             ChangeListener<String>() {
            @Override
            public void changed(ObservableValue<? extends String> observable, String
                 oldValue, String newValue) {
                if (newValue.equals("")) {
                    menuItemCopy.setDisable(true);
                    copyMenuItem.setDisable(true);
                } else {
                    menuItemCopy.setDisable(false);
                    copyMenuItem.setDisable(false);
                }
            }
        });
        documentInlineCssTextArea.setContextMenu(contextMenu);
        highlightText(result.getOriginalString(), stringInFile);
    } else {
        System.out.println("File is unavailable, cant be read");
        documentInlineCssTextArea.replaceText("");
        Alert documentUnvailable = documentUnavailableDialog(result.getFileData());
        documentUnvailable.showAndWait();
        //should close this window, as the file is unavailable
        rootBorderPane = null;
    }
}

/**
 * For the given FileData (holds the Files name, and represents it), produce an Alert
     Dialog to show to the User,
 * to inform them that the File which they wish to read is unavailable.
 *
 * @param fileData the given FileData
 * @return the Alert Dialog the informs the User the File is unavailable to read.
 */
private Alert documentUnavailableDialog(FileData fileData) {
    Alert alert = new Alert(Alert.AlertType.INFORMATION);
```

```
        alert.setTitle("Document Unavailable");
        alert.setHeaderText(null);
        alert.setContentText("The File: " + fileData.getFileName() + " is Unavailable");
        alert.getDialogPane().getButtonTypes().setAll(ButtonType.OK);
        return alert;
    }

    /**
     * Called to set the action listeners for the Menu Items: Close and Copy
     */
    private void setOnActionListeners() {
        copyMenuItem.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                copy();
            }
        });
        closeMenuItem.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                close();
            }
        });
    }

    /**
     * Called to highlight the text "textToHighlight" in the "from" text, in the text area
         of this layout.
     *
     * @param textToHighlight the given text that needs to be highlighted. Part of the
         "from" text
     * @param from          the full text, that contains the text that needs to be
         highlighted.
     */
    private void highlightText(String textToHighlight, String from) {//highlight first
        occurrence of the text to highlight
```

```
        int startHighlight = from.indexOf(textToHighlight);
        int endHighlight = startHighlight + textToHighlight.length();//as we highlight the
            original sentence, which we know its length, and its start index
        if (endHighlight > from.length()) {//if we are over the limit of the text, then we
            go up to that point
            endHighlight = documentInlineCssTextArea.getText().length();
        }
        System.out.println("EndHighlight: " + endHighlight);
        if (startHighlight >= 0 && startHighlight < from.length() && endHighlight >= 0) {
            documentInlineCssTextArea.setStyle(startHighlight, endHighlight, "-fx-fill:
                blue; -fx-font-weight: bold");
        }
    }

    /**
     * Called to extract the text in the File which the given FileData represents.
     *
     * @param fileData the FileData that represents the File from which we are extracting
         text from.
     * @return null, if the File has been moved/deleted or we don't have read rights;
         otherwise the text of the File is
     * returned.
     */
    private String getStringInFile(FileData fileData) {
        if (fileData != null) {
            File file = new File(fileData.getFilePath());
            if (file.exists() && file.isFile() && file.canRead()) {
                ProcessFiles processFiles = new ProcessFiles();
                return processFiles.getTextInFile(file);
            }
        }
        return null;
    }

    /**
```

```
     * Called when the Close Menu Item is pressed. Inform the observer that they need to
          close the window in which this
     * layout resides in.
     */
    private void close() {
        System.out.println("Close Window");
        if (documentReaderObserver != null) {
            documentReaderObserver.close();
        }
    }

    /**
     * Called when the Copy Menu Item is pressed. Used to copy the text in the file to the
          clipboard.
     */
    private void copy() {
        if (documentInlineCssTextArea != null) {
            System.out.println("Copy Text to Clipboard: " +
                 documentInlineCssTextArea.getSelectedText());
            Toolkit toolkit = Toolkit.getDefaultToolkit();
            if (toolkit != null && documentInlineCssTextArea.getSelectedText() != null) {
                toolkit.getSystemClipboard().setContents(new
                     StringSelection(documentInlineCssTextArea.getSelectedText()), null);
            }
        }
    }

    /**
     * Get the BorderPane layout of the layout produced when a new object is made (that was
          populated with the given
     * Result object). This is the root layout.
     *
     * @return the BorderPane (root) layout, produced when the constructor is invoked.
     */
    public BorderPane getRootBorderPane() {
        return rootBorderPane;
```

```
    }
}
```

EditEventController.java

```
package frontend.controllers;


import backend.process.Result;
import frontend.helpers.TextFieldState;
import frontend.observers.EditEventDialogObserver;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.css.PseudoClass;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.control.*;
import javafx.scene.image.ImageView;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.HBox;


import java.io.IOException;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.*;


/**
 * Controller of the Content in the EditEventDialog
 */
public class EditEventController {
    private final static int charLimit = 100;//100 characters should be enough for the
        user to write a summary of a sentence (might reduce it to less)
    private final static String maxChar = "Maximum 100 characters ";
    @FXML
```

```
    private TextField firstDateTextField;
    @FXML
    private TextField secondDateTextField;
    @FXML
    private TextField subjectTextField;
    @FXML
    private ImageView addImageView;
    @FXML
    private TextArea eventTextArea;
    @FXML
    private HBox subjectsHBox;
    @FXML
    private GridPane rootGridPane;
    @FXML
    private Label maxCharacterLabel;
    private Result result;
    private SimpleDateFormat simpleDateFormat = new SimpleDateFormat("dd-MM-yyyy");
    private SimpleDateFormat validInputFormat = new SimpleDateFormat("dd-MM-yyyy");
    private final PseudoClass errorClass = PseudoClass.getPseudoClass("error");
    private EditEventDialogObserver editEventDialogObserver;
    private ArrayList<TextFieldState> textFieldStates;
    private final static int subjectLimit = 5;
    private int subjectCount = 0;



    /**
     * Constructor to create the layout for the Content of the EditEventDialog.
     *
     * @param result               the Result object used to populate the fields of the
         layout of the Dialog content.
     * @param editEventDialogObserver the Observer of this content (the holder of the
         Dialog), to inform when to enable
     *                             and disable the buttons (which aren't part of the
         content of the dialog, but
     *                             separate).
     */
```

```
public EditEventController(Result result, EditEventDialogObserver
     editEventDialogObserver) {
    this.result = result;
    this.editEventDialogObserver = editEventDialogObserver;
    textFieldStates = new ArrayList<>(3);
    FXMLLoader fxmlLoader = new
         FXMLLoader(getClass().getResource("editEventDialog.fxml"));
    fxmlLoader.setController(this);
    try {
        fxmlLoader.load();
    } catch (IOException e) {
        e.printStackTrace();
    }
    setData();
    addListeners();
}


/**
 * For the Result object that has been set, populate the fields.
 */
private void setData() {
    //for the set result object populate the fields
    if (result != null) {
        firstDateTextField.setText(getStringFromDate(result.getTimelineDate()
              .getDate1()));
        firstDateTextField.getStylesheets().add(getClass()
              .getResource("customErrorFields.css").toExternalForm());
        textFieldStates.add(0, TextFieldState.CORRECT);
        secondDateTextField.setText(getStringFromDate(result.getTimelineDate()
              .getDate2()));
        secondDateTextField.getStylesheets().add(getClass()
              .getResource("customErrorFields.css").toExternalForm());
        textFieldStates.add(1, TextFieldState.CORRECT);
        subjectsHBox.getChildren().setAll(getSubjectLabels(result.getSubjects()));
        eventTextArea.setText(result.getEvent());
        eventTextArea.getStylesheets().add(getClass()
```

```
            .getResource("customErrorFields.css").toExternalForm());
        textFieldStates.add(2, TextFieldState.CORRECT);
    }
}

/**
 * For the set of Strings (which should be Subjects) produce a list of Labels (to add
     to the HBox Subjects layout).
 *
 * @param subjects the set of Strings.
 * @return a list of Labels (corresponding to each each item in the set and a listener
     to show a dialog to remove the
 * subject from the Result).
 */
private List<Label> getSubjectLabels(Set<String> subjects) {
    List<Label> labels = new ArrayList<>();
    subjectCount = subjects.size();
    for (String subject : subjects) {
        Label label = new Label(subject);
        label.setUnderline(true);
        label.setOnMouseClicked(new EventHandler<MouseEvent>() {
            @Override
            public void handle(MouseEvent event) {
                System.out.println("Remove this label and subject from the Result: " +
                     subject);
                if (shouldDeleteSubjectLabel(subject)) {
                    System.out.println("Removed: " + subject);
                    result.getSubjects().remove(subject);//remove the given subject
                    System.out.println("Remaining: " + result.getSubjects());//need to
                         update the subjects shown
                    subjectsHBox.getChildren()
                       .setAll(getSubjectLabels(result.getSubjects()));//so get the new
                            labels and set them

                }
            }
```

```
        });
        labels.add(label);
    }
    return labels;
}

/**
 * For a given Date (which can be null), return it in string format-
 *
 * @param date the given Date.
 * @return an empty String if the given date is null; otherwise the date as a String in
     the format dd-MM-yyyy.
 */
private String getStringFromDate(Date date) {
    return (date != null) ? simpleDateFormat.format(date) : "";
}

/**
 * Add the listeners (event handlers) to the input fields (text fields, and the image
     view).
 */
private void addListeners() {
    //for the fields add their event listeners (eg iamgeview add to hbox of label)
    addImageView.setOnMouseClicked(new EventHandler<MouseEvent>() {
        @Override
        public void handle(MouseEvent event) {
            System.out.println("Add subject in textfield to subject list: " +
                 subjectTextField.getText());
            addSubjectAndClear(subjectTextField);
        }
    });
    subjectTextField.setOnAction(new EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent event) {
            System.out.println("Enter is pressed, add subject to list: " +
                 subjectTextField.getText());
```

```
            addSubjectAndClear(subjectTextField);
        }
    });
    firstDateTextField.focusedProperty().addListener(new ChangeListener<Boolean>() {
        @Override
        public void changed(ObservableValue<? extends Boolean> observable, Boolean
             oldValue, Boolean newValue) {
            //validate date before setting it
            if (isValidDateInput(firstDateTextField.getText())) {
                firstDateTextField.pseudoClassStateChanged(errorClass, false);
                result.getTimelineDate().setDate1(getDate(firstDateTextField.getText().trim()));
                textFieldStates.set(0, TextFieldState.CORRECT);
                enableSave();
            } else {
                firstDateTextField.pseudoClassStateChanged(errorClass, true);
                textFieldStates.set(0, TextFieldState.WRONG);
                disableSave();
            }
        }
    });
    secondDateTextField.focusedProperty().addListener(new ChangeListener<Boolean>() {
        @Override
        public void changed(ObservableValue<? extends Boolean> observable, Boolean
             oldValue, Boolean newValue) {
            //validate date before setting it
            if (secondDateTextField.getText().trim().length() == 0) {
                //no second date
                result.getTimelineDate().setDate2(null);
                secondDateTextField.pseudoClassStateChanged(errorClass, false);
                textFieldStates.set(1, TextFieldState.CORRECT);
                enableSave();
            } else if (isValidDateInput(secondDateTextField.getText()) &&
                 getDate(secondDateTextField.getText().trim())
                 .compareTo(result.getTimelineDate().getDate1()) > 0) {
                //valid input
                secondDateTextField.pseudoClassStateChanged(errorClass, false);
```

```
            result.getTimelineDate()
             .setDate2(getDate(secondDateTextField.getText().trim()));
            textFieldStates.set(1, TextFieldState.CORRECT);
            enableSave();
        } else {
            secondDateTextField.pseudoClassStateChanged(errorClass, true);//disable
                button (need observer of this to tell)
            textFieldStates.set(1, TextFieldState.WRONG);
            disableSave();
        }
    }
});
eventTextArea.lengthProperty().addListener(new ChangeListener<Number>() {
    @Override
    public void changed(ObservableValue<? extends Number> observable, Number
         oldValue, Number newValue) {
        if (updateLabel(newValue.intValue())) {
            textFieldStates.set(2, TextFieldState.WRONG);
            eventTextArea.pseudoClassStateChanged(errorClass, true);
            disableSave();
        } else {
            textFieldStates.set(2, TextFieldState.CORRECT);
            eventTextArea.pseudoClassStateChanged(errorClass, false);
            enableSave();
        }
    }
});
eventTextArea.focusedProperty().addListener(new ChangeListener<Boolean>() {
    @Override
    public void changed(ObservableValue<? extends Boolean> observable, Boolean
         oldValue, Boolean newValue) {
        //save the event once we leave it
        if (textFieldStates.size() == 3 && textFieldStates.get(2) ==
             TextFieldState.CORRECT) {
            System.out.println("Saving result event: " + eventTextArea.getText());
            result.setEvent(eventTextArea.getText());
```

```
            }
        }
    });
}

/**
 * Update the label that shows how many characters the users has left to type into the
     text area, or by how many
 * characters they have exceeded the limit. Returns whether or not the user has
     exceeded the character limit.
 *
 * @param charAmount the amount of characters the user has written in the textarea.
 * @return true if the charAmount has exceeded the maximum number of characters.
 */
private boolean updateLabel(int charAmount) {
    String textToSet = maxChar;
    if (charAmount > charLimit) {
        textToSet += "(" + (charAmount - charLimit) + " too many)";
    } else {
        textToSet += "(" + (charLimit - charAmount) + " remaining)";
    }
    maxCharacterLabel.setText(textToSet);
    return charAmount > charLimit;
}

/**
 * For the given TextField, add its text to the list of Subjects of the Result
     (including showing its label), and
 * clear the field (as the subjects has already been added).
 *
 * @param textFieldSubject the givenTextField.
 */
private void addSubjectAndClear(TextField textFieldSubject) {
    if (canAdd()) {
        addTextToSubjects(textFieldSubject.getText());
        textFieldSubject.setText("");
```

```
    } else {
        //inform the user they cant add more subjects
        Alert maxNumberSubjectDialog = getInformMaxSubject();
        maxNumberSubjectDialog.show();
    }
}

/**
 * For the given String (which should be a Subject), add it to Subjects of the Result
     (and update the list of
 * Subjects that is being shown).
 *
 * @param newSubject the given String.
 */
private void addTextToSubjects(String newSubject) {
    if (newSubject.length() >= 1) {
        result.addSubject(newSubject);
        List<Label> labels = getSubjectLabels(result.getSubjects());
        subjectsHBox.getChildren().setAll(labels);
    }
}

/**
 * For the given String, produce a Date (assuming the String is in the format:
     dd-MM-yyyy), else null will be
 * returned.
 *
 * @param date the given String.
 * @return a non-null Date object if the given String is of the format: dd-MM-yyyy;
     otherwise null.
 */
private Date getDate(String date) {
    Date toReturn = null;
    try {
        toReturn = validInputFormat.parse(date);
    } catch (ParseException e) {
```

```
            e.printStackTrace();
        }
        return toReturn;
    }

    /**
     * Called to disable the save button in the Dialog.
     */
    private void disableSave() {
        //tell the dialog to disable save
        editEventDialogObserver.disableSave(true);
    }

    /**
     * Called to enable the save button in the Dialog.
     */
    private void enableSave() {
        boolean canSave = true;
        for (TextFieldState textFieldState : textFieldStates) {
            if (textFieldState == TextFieldState.WRONG) {
                canSave = false;
                break;
            }
        }
        if (canSave) {
            //tell the dialog to enable save
            editEventDialogObserver.disableSave(false);
        }
    }

    /**
     * Checks whether the given input is of the valid format (dd-MM-yyyy).
     *
     * @param input a String date
     * @return true if the input is of the format dd-MM-yyyy, false otherwise.
     */
```

```
    private boolean isValidDateInput(String input) {
        try {
            validInputFormat.setLenient(false);
            validInputFormat.parse(input);
            return input.length() == 10;
        } catch (ParseException e) {
            return false;
        }
    }

    /**
     * Getter method for the root layout that this Controller is representing, which is a
         GridPane.
     *
     * @return the GridPane of the layout this controller is representing, which is the
         root layout.
     */
    public GridPane getRootGridPane() {
        return rootGridPane;
    }

    /**
     * Creates a Confirmation Alert Dialog, to allow the user to choose whether the given
         Subject String should be
     * removed from the Result. The users response is returned.
     *
     * @param subject the given Subject String.
     * @return true if the YES option is picked in the Dialog (that is the subject should
         be deleted); false otherwise
     */
    private boolean shouldDeleteSubjectLabel(String subject) {
        Alert confirmationDeleteDialog = new Alert(Alert.AlertType.CONFIRMATION);
        confirmationDeleteDialog.setTitle("Deleting a Subject");
        confirmationDeleteDialog.setContentText("Are you sure you want to delete: \"" +
            subject + "\" from the subjects of this event?");
```

```
        confirmationDeleteDialog.getButtonTypes().setAll(ButtonType.YES, ButtonType.NO,
            ButtonType.CANCEL);
        Optional<ButtonType> response = confirmationDeleteDialog.showAndWait();
        return response.get() == ButtonType.YES;
    }

    /**
     * Whether or not a Subject can be added to the list of Subjects of the Result object.
     *
     * @return true if the current subject count is below the subjectLimit (5); false
         otherwise.
     */
    private boolean canAdd() {
        return subjectCount < subjectLimit;
    }

    /**
     * Get a Information Alert Dialog that informs the user that the subject limit has been
         reached, and that they can
     * remove subjects by pressing them in the list (to make space for the new subject).
         This is so that the user doesn't
     * add too many subjects for an event which was given by just 1 sentence.
     *
     * @return a Information Dialog to inform the user the subject limit has been reached.
     */
    private Alert getInformMaxSubject() {
        Alert alert = new Alert(Alert.AlertType.INFORMATION);
        alert.setTitle("Add Subject Information");
        alert.setHeaderText(null);
        alert.setContentText("The maximum number of Subjects allowed has been reached. To
            remove a Subject just click on it.");
        return alert;
    }

}
```

ListViewController.java

```
package frontend.controllers;


import backend.helpers.Sort;
import backend.process.FileData;
import backend.process.Result;
import backend.ranges.ProduceRanges;
import backend.ranges.Range;
import frontend.dialogs.LoadingDialog;
import frontend.dialogs.RemoveConfirmationDialog;
import frontend.observers.DocumentsLoadedObserver;
import frontend.observers.TimelineObserver;
import frontend.observers.TimelineRowObserver;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.geometry.Insets;
import javafx.scene.Node;
import javafx.scene.control.*;
import javafx.scene.input.ScrollEvent;
import javafx.scene.layout.StackPane;
import javafx.scene.layout.VBox;
import javafx.util.Callback;


import java.net.URL;
import java.util.*;



/**
 * Controller for the layout where the ListView is shown. Allows the listview to be
     populated with Result data.
 */
```

```
public class ListViewController implements Initializable, MenuBarControllerInter,
     DocumentsLoadedObserver, TimelineRowObserver {
    private enum ViewType {
        RANGE, DATE
    }

    private final static double MAX_ZOOM = 1.0d;
    private final static double MIN_ZOOM = 0.75d;

    @FXML
    private StackPane stackPane;
    @FXML
    private VBox vBox;
    @FXML
    private ListView<Object> timelineListView;
    @FXML
    private Button loadDocumentsButton;
    @FXML
    private Button saveToButton;
    @FXML
    private ListView<FileData> documentListView;
    @FXML
    private RadioMenuItem dateView;
    @FXML
    private RadioMenuItem rangeView;
    @FXML
    private ScrollPane scrollPane;
    private List<Result> results;
    private List<FileData> fileDatas;
    private ObservableList<Object> timelineObservableList =
        FXCollections.observableArrayList();
    private ObservableList<FileData> documentsLoadedObservableList =
        FXCollections.observableArrayList();
    private TimelineObserver timelineObserver;
    private LoadingDialog loadingDialog;
    private ToggleGroup radioMenuItemGroup;
```

```
    private ViewType viewType = ViewType.DATE;



    /**
     * Called when the layout is created.
     */
    @Override
    public void initialize(URL location, ResourceBundle resources) {
        System.out.println("Initialised timelineListView");
        System.out.println("documentListView: " + documentListView);
        System.out.println("loadDocumentsButton: " + loadDocumentsButton);
        System.out.println("saveToButton: " + saveToButton);
        loadingDialog = new LoadingDialog(stackPane, vBox);//pass the root layout and main
             content layout to know where
        //to show the loading dialog, and what to disable.
        //set up the group of the radio buttons in the menu
        //by default the dateView is shown
        dateView.setSelected(true);
        radioMenuItemGroup = new ToggleGroup();
        rangeView.setToggleGroup(radioMenuItemGroup);
        dateView.setToggleGroup(radioMenuItemGroup);
        rangeView.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                showRangeTimeline();
            }
        });
        dateView.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                showDateTimeline();
            }
        });

    }
```

```
/**
 * Called to set the Observer for this Scene.
 *
 * @param timelineObserver the Observer for this Timeline Scene.
 */
public void setTimelineObserver(TimelineObserver timelineObserver) {
    this.timelineObserver = timelineObserver;
    loadDocumentsButton.setOnAction(new EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent event) {
            System.out.println("Load in Documents");
            if (timelineObserver != null) {
                timelineObserver.loadDocuments();
            }
        }
    });
    saveToButton.setText("Save \nTo...");
    saveToButton.setOnAction(new EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent event) {
            if (timelineObserver != null) {
                timelineObserver.saveTo(results);
            }
        }
    });
}

/**
 * For the input List, sort and reverse it.
 * O(n)
 *
 * @param results the input List.
 */
private void sortAndReverse(List<Result> results) {
    Sort.sortByDate1(results);//sort the results by their date1 value
    Collections.reverse(results);
```

```
}

/**
 * For the given Node make it visible and enable it depending on shouldShow. If
     shouldShow is true then make the
 * node visible and enable it; if it false make the node invisible and disable it.
 *
 * @param node     the given Node.
 * @param shouldShow whether or not the Node should be shown and enabled or not.
 */
private void show(Node node, boolean shouldShow) {
    node.setVisible(shouldShow);
    node.setDisable(!shouldShow);
}

/**
 * For the input List of Results, set it as the items of the TimelineList. The type of
     timeline shown in the
 * ListView is given by the ViewType (member of the class that is changed with the
     RadioMenuItems).
 * <p>
 * The ListView and its ObservableList both set their generic type to Object, this way
     we don't have to create a
 * separate ListView when we want to swap from Results to Ranges (and vice versa). As
     when a ListView is set to
 * use a list of Ranges (for example) it will set its type to that, so then when it
     attempts to set a list of
 * Results it will throw an exception of the Casting failing (because its type is
     Ranges but its attempting to cast
 * it to Result).
 * <p>
 * When a row needs to be created from an item in that list, its type is looked at to
     determine what layout to use.
 * <p>
 * In order to change between Timeline Views use a ViewType that sets which View to
     use, where the RadioMenuItems
```

```
     * change its value according, with the default Timeline view being the Date Timeline
         (showing Results
     * individually). To allow the ListView to show different views of lists that each hold
         different kind of data, the
     * list and the observable had to be made of type Object as otherwise swapping from a
         list of Results to a list of
     * Ranges and setting them to the ListView would throw an exception of the Casting
         failing of a Range to a Result
     * (this also happens vice versa).
     * <p>
     * The timeline for the Range view is a ScrollPane that can be zoomed in/out.
     *
     * @param results the input List.
     */
    private void setTimelineList(List<Result> results) {
        //should show loading dialog while its setting the timeline
        timelineObservableList.clear();
        //check what kind of view we need to show
        if (viewType == ViewType.RANGE) {
            //we need to show the range view, which supports zooming in and out (so we need
                to use a vbox and scrollpane,
            //to be able to zoom)
            ProduceRanges produceRanges = new ProduceRanges();
            produceRanges.produceRanges(results);//produce the results


            VBox listVBox = new VBox();//the vbox that holds the Ranges
            listVBox.setPadding(new Insets(10));
            for (Range range : produceRanges.getTrees()) {//for each tree build its layout
                 and add it to the vbox (row by row)
                CustomTimelineRow customTimelineRow = new CustomTimelineRow(range, this);
                listVBox.getChildren().add(customTimelineRow.getPane());
            }
            scrollPane.setContent(listVBox);//set the content of the scrollpane
            scrollPane.addEventFilter(ScrollEvent.ANY, new EventHandler<ScrollEvent>()
                 {//add the event handler for the zooming
                @Override
```

```
            public void handle(ScrollEvent event) {
                if (event.isControlDown()) {//if we are holding the control button
                    double scale = getScale(event, listVBox);//get the value by which we
                        scale
                    listVBox.setScaleX(scale);//and set it
                    listVBox.setScaleY(scale);
                    event.consume();
                }
            }

            private double getScale(ScrollEvent scrollEvent, Node node) {
                double scale = node.getScaleX() + scrollEvent.getDeltaY() / 100;
                if (scale <= MIN_ZOOM) {//we only want the user to zoom out (not in,
                     hence the scale is never over 1)
                    scale = MIN_ZOOM;
                } else if (scale >= MAX_ZOOM) {
                    scale = MAX_ZOOM;
                }
                return scale;
            }
        });
        show(scrollPane, true);//show the scrollpane
        show(timelineListView, false);//and hide the timeline list view
        return;//no need to follow the rest
    } else if (viewType == ViewType.DATE) {//if we have to show a date timeline
        show(scrollPane, false);//hide the scrollpane
        show(timelineListView, true);//show the timeline
        timelineListView.getStylesheets().setAll(getClass().getResource("listViewTheme.css")
              .toExternalForm());
        timelineObservableList.setAll(results);//add the results
    }
    //assuming the observable list items have been set
    timelineListView.setItems(timelineObservableList);
    timelineListView.setCellFactory(new Callback<ListView<Object>, ListCell<Object>>() {
        @Override
        public ListCell<Object> call(ListView<Object> param) {
```

```
            return new ListCell<Object>() {
                /**
                 * Called whenever a row needs to be shown/created on the screen.
                 * @param item the Range object for which this row has to be displayed
                     for.
                 * @param empty whether or nor the row is empty.
                 */
                @Override
                protected void updateItem(Object item, boolean empty) {
                    super.updateItem(item, empty);
                    if (item != null && !empty) {
                        if (item instanceof Range) {
                            Range range = (Range) item;
                            CustomTimelineRow customTimelineRow = new
                                 CustomTimelineRow(range, ListViewController.this);
                            setGraphic(customTimelineRow.getPane());
                        } else if (item instanceof Result) {
                            Result result = (Result) item;
                            TimelineRowController timelineRowController = new
                                 TimelineRowController(getIndex(),
                                 ListViewController.this);
                            timelineRowController.setData(result);
                            setGraphic(timelineRowController.getGroup());
                        } else {
                            setGraphic(null);
                        }
                    } else {
                        setGraphic(null);
                    }
                }
            };
        }
    });
}

/**
```

```
     * For the input List of FileData, set it as the items of the Documents Loaded List.
     *
     * @param fileDatas the input List.
     */
    private void setDocumentListView(List<FileData> fileDatas) {
        documentsLoadedObservableList.clear();
        documentsLoadedObservableList.addAll(fileDatas);
        documentListView.setItems(documentsLoadedObservableList);
        documentListView.setCellFactory(new Callback<ListView<FileData>,
             ListCell<FileData>>() {
            @Override
            public ListCell<FileData> call(ListView<FileData> param) {
                return new ListCell<FileData>() {
                    @Override
                    protected void updateItem(FileData item, boolean empty) {
                        super.updateItem(item, empty);
                        if (item != null && !empty) {
                            DocumentLoadedRowController documentLoadedRowController = new
                                 DocumentLoadedRowController(item, ListViewController.this);
                            setGraphic(documentLoadedRowController.getGridPane());
                        } else {
                            setGraphic(null);
                        }
                    }
                };
            }
        });
    }

    /**
     * For the given Results and FileData, set the data of the timelineListView and
         documentListView with them.
     *
     * @param results a list of Result objects which contain data to populate the rows of
         the timelineListView with.
```

```
     * @param fileDatas a list of FileData objects which needs to populate the rows in the
          documentListView.
     */
    public void setTimelineListView(List<Result> results, List<FileData> fileDatas) {
        this.results = results;
        if (viewType == ViewType.DATE) {//to not waste time sorting Results that will be
             sorted by their Ranges later anyways
            sortAndReverse(this.results);
        }
        setTimelineList(this.results);

        this.fileDatas = fileDatas;
        Collections.sort(this.fileDatas);
        setDocumentListView(this.fileDatas);
    }

    /**
     * For the given input, add it to their appropriate lists.
     *
     * @param results a list of Result objects which contain data to add to the
          timelineListView.
     * @param fileDatas a list of FileData objects which needs to be added to the
          documentListView.
     */
    public void addToTimelineListView(List<Result> results, List<FileData> fileDatas) {
        cleanRepeatedResults(results, this.fileDatas);
        cleanRepeatedFileData(this.fileDatas, fileDatas);
        this.results.addAll(results);
        sortAndReverse(this.results);
        setTimelineList(this.results);

        this.fileDatas.addAll(fileDatas);
        Collections.sort(this.fileDatas);
        setDocumentListView(this.fileDatas);
    }
```

```
/**
 * Called to remove from the newFileData list, all the FileData objects that are
     already present in the oldFileData.
 *
 * @param oldFileData the given oldFileData.
 * @param newFileData the given newFileData.
 */
private void cleanRepeatedFileData(List<FileData> oldFileData, List<FileData>
     newFileData) {
    Iterator<FileData> newFileDataIterator = newFileData.iterator();
    while (newFileDataIterator.hasNext()) {//for each new result
        FileData fileData = newFileDataIterator.next();
        if (oldFileData.contains(fileData)) {//if its in the old list
            newFileDataIterator.remove();//then dont add it to it (so remove it)
        }
    }
}

/**
 * Called to remove all the Result objects in results list that have their FileData
     object present in the fileDatas
 * List.
 *
 * @param results the given Result list.
 * @param fileDatas the given FileData list.
 */
private void cleanRepeatedResults(List<Result> results, List<FileData> fileDatas) {
    Iterator<Result> resultIterator = results.iterator();
    while (resultIterator.hasNext()) {
        //for each result, check it with the filedata, if its filedata is already
             there, then remove it
        Result result = resultIterator.next();
        if (fileDatas.contains(result.getFileData())) {
            //this file data already exists, so remove the result
            resultIterator.remove();
            System.out.println("Removed: " + result + " from results to be added");
```

```
        }
    }
}

/**
 * Called when the Close menu item is pressed.
 */
@Override
public void close() {
    System.out.println("Close pressed");
    if (timelineObserver != null) {
        timelineObserver.close();
    }
}

/**
 * Called when the About menu item is pressed.
 */
@Override
public void about() {
    System.out.println("About pressed");
    if (timelineObserver != null) {
        timelineObserver.showAbout();
    }
}

/**
 * Called when the Timeline menu item is pressed.
 */
@Override
public void timeline() {
    System.out.println("Timeline pressed");
    if (timelineObserver != null) {
        timelineObserver.timeline();
    }
}
```

```
    /**
     * When the preferences menu item is pressed.
     */
    @Override
    public void preferences() {
        if (timelineObserver != null) {
            timelineObserver.preferences();
        }
    }

    /**
     * Called when a given row in the Loaded Documents listview is removed.
     *
     * @param fileData the FileData for the File where we want to remove its results from
         the Timeline.
     */
    @Override
    public void remove(FileData fileData) {
        System.out.println("Need to remove: " + fileData);
        Alert removeConfirmationDialog =
             RemoveConfirmationDialog.getRemoveConfirmationDialog(fileData.getFileName());
        Optional<ButtonType> response = removeConfirmationDialog.showAndWait();
        if (response.isPresent() && response.get() == ButtonType.YES) {
            fileDatas.remove(fileData);
            removeResults(results, fileData);
            setTimelineListView(results, fileDatas);
        }//else dont remove the events related to that file (as we keep it)
    }

    /**
     * Removes the given FileData from the FileData list and all the Results linked to it
         in the Results list.
     *
     * @param results list of Results for which we we need to delete the Results linked to
         the given FileData.
```

```
     * @param fileData FileData for which in the given Results list we need to remove the
          linked Results.
     */
    private void removeResults(List<Result> results, FileData fileData) {
        Iterator<Result> resultIterator = results.iterator();
        while (resultIterator.hasNext()) {
            Result result = resultIterator.next();
            if (result.getFileData().equals(fileData)) {
                System.out.println("Need to remove: " + result);
                resultIterator.remove();
            }
        }
    }

    /**
     * Called by a member of the ListView, to inform the ListView to update, as it updated
          its values.
     * The updated list can have this new Result row somewhere else as the user could have
          changed its date (and the list
     * is sorted by dates).
     *
     * @param updatedResult the Result object of the row that was edited.
     * @param position    the position this cell was in the ListView.
     */
    @Override
    public void update(Result updatedResult, int position) {
        System.out.println("Update the list");
        if (results.size() > position) {
            results.set(position, updatedResult);
            setTimelineListView(results, fileDatas);
        }

    }

    /**
```

```
     * Called by a member of the ListView, to inform the ListView to update, as it updated
         its values.
     * The updated list can have this new Result row somewhere else as the user could have
         changed its date (and the list
     * is sorted by dates).
     *
     * @param previous    the Result object that was previously in this Row.
     * @param updatedResult the updated Result object of the TimelineRow.
     */
    @Override
    public void update(Result previous, Result updatedResult) {
        int pos = results.indexOf(previous);
        System.out.println("Previous: " + pos);
        if (pos != -1) {
            update(updatedResult, pos);
        }
    }

    /**
     * Called by a member of the ListView, to inform the ListView that it needs to be
         removed from the List.
     * Thereby the list needs to be updated (i.e. set again).
     *
     * @param position the position of the event that needs to be deleted.
     */
    @Override
    public void delete(int position) {
        System.out.println("Deleting the event");
        if (results.size() > position) {
            results.remove(position);
            setTimelineListView(results, fileDatas);
        }
    }

    /**
```

```
     * Called by a member of the ListView, to inform the ListView that it needs to be
         removed from the List.
     * Thereby the list needs to be updated (i.e. set again).
     *
     * @param result the given event (Result) to be deleted.
     */
    @Override
    public void delete(Result result) {
        int pos = results.indexOf(result);
        System.out.println("pos: " + pos);
        if (pos != -1) {
            delete(pos);
        }
    }

    /**
     * Called to show the loading dialog. (Only if the layouts have been passed to
         LoadingDialog)
     */
    public void showLoadingDialog() {
        loadingDialog.showLoadingDialog();
    }

    /**
     * Called to remove the loading dialog.
     */
    public void removeLoadingDialog() {
        loadingDialog.removeLoadingDialog();
    }

    /**
     * Called to show the Timeline with the individual dates and events in separate rows.
     */
    private void showDateTimeline() {
        if (viewType != ViewType.DATE) {
            viewType = ViewType.DATE;
```

```
            //show what is shown
            setTimelineListView(results, fileDatas);
        }
    }

    /**
     * Called to show the Timeline with the Results grouped into their ranges and then
          shown.
     */
    private void showRangeTimeline() {
        if (viewType != ViewType.RANGE) {
            viewType = ViewType.RANGE;
            setTimelineListView(results, fileDatas);
        }
    }
}
```

MenuBarControllerInter.java

```
package frontend.controllers;


/**
 * Implemented for Observers of a Controller that has a menu bar.
 */
public interface MenuBarControllerInter {
    /**
     * Called when the close menu item is selected.
     */
    void close();

    /**
     * Called when the about menu item is selected.
     */
    void about();

    /**
```

```
    * Called when the timeline menu item is selected.
    */
   void timeline();

   /**
    * Called when the preferences menu item is selected.
    */
   void preferences();
}
```

RangeDataController.java

```
package frontend.controllers;

import backend.process.Result;
import backend.ranges.Range;
import frontend.observers.TimelineRowObserver;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.control.Label;
import javafx.scene.control.ListCell;
import javafx.scene.control.ListView;
import javafx.scene.layout.Pane;
import javafx.scene.layout.VBox;
import javafx.util.Callback;

import java.io.IOException;

/**
 * Controller for the layout that represents the data held by a Range (i.e. its Data and
     its Results).
 */
public class RangeDataController {
   private Range range;
```

```
@FXML
private VBox rootVBox;
@FXML
private ListView<Result> resultsListView;
@FXML
private Label dateLabel;
private ObservableList<Result> results;
private TimelineRowObserver timelineRowObserver;
private int rangePosition;

/**
 * Constructor that sets up the layout to represent the data held by the given Range.
 *
 * @param range the given Range.
 */
public RangeDataController(Range range, TimelineRowObserver timelineRowObserver, int
     rangePosition) {
    this.range = range;
    this.timelineRowObserver = timelineRowObserver;
    this.rangePosition = rangePosition;
    FXMLLoader fxmlLoader = new
         FXMLLoader(getClass().getResource("rangeDataLayout.fxml"));//load the base
         layout
    fxmlLoader.setController(this);
    try {
        rootVBox = fxmlLoader.load();
        setUp();//set the data in the layout
    } catch (IOException e) {
        e.printStackTrace();
    }
}

/**
 * Set up the data shown in this layout. This means setting the date that this Range
     layout is representing, and the
 * list of Result objects held by the given Range (in the constructor).
```

```
 */
private void setUp() {
    if (range != null) {
        dateLabel.setText(range.getDateRange());//set the date text
        if (range.getResults().size() > 0) {//if we have results then show the list
            results = FXCollections.observableArrayList();
            results.addAll(range.getResults());//set up the observable list of Results
                 used to add to the listView
            resultsListView.setItems(results);
            resultsListView.setCellFactory(new Callback<ListView<Result>,
                 ListCell<Result>>() {
                @Override
                public ListCell<Result> call(ListView<Result> param) {
                    return new ListCell<Result>() {//set up the listcell used in this
                         listview
                        @Override
                        protected void updateItem(Result item, boolean empty) {
                            super.updateItem(item, empty);
                            if (item != null) {//if we have a valid item, build its
                                 layout and set it
                                CustomResultRowController customResultRowController = new
                                     CustomResultRowController(item, timelineRowObserver,
                                     rangePosition);
                                setGraphic(customResultRowController.getRootLayout());//set
                                     the layout built with the result
                            } else {//dont have a valid result so dont show anything
                                setGraphic(null);
                            }
                        }
                    };
                }
            });
        } else {//dont have results then dont show the list
            rootVBox.getChildren().remove(resultsListView);
        }
    }
```

```
    }

    /**
     * Get the root layout that this Controller controls (ie a VBox with the Date of the
          given Range - passed in the
     * constructor; and a list of the Results held by the Range).
     *
     * @return the root layout that this Controller represents.
     */
    public Pane getRootBorderPane() {
        return rootVBox;
    }


}
```

StartUpController.java

```
package frontend.controllers;

import frontend.dialogs.LoadingDialog;
import frontend.observers.StartUpObserver;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.scene.layout.VBox;

import java.net.URL;
import java.util.ResourceBundle;

/**
 * Controller for the Startup Scene. Holds a StartUpObserver, where it calls the relevant
      methods, when the Controller is
 * called. As the scene has a menu bar, it implements the MenuBarControllerInterface, by
      implementing the relevant methods
```

```
 * for the menu bar.
 */
public class StartUpController implements Initializable, MenuBarControllerInter {
    private final static String TAG = "STARTUPCONTROLLER: ";
    private StartUpObserver observer;
    @FXML
    private Button loadDocumentsButton;
    @FXML
    private StackPane stackPane;
    @FXML
    private VBox vBox;//main layout
    private LoadingDialog loadingDialog;

    /**
     * Called on creation of the Scene.
     */
    @Override
    public void initialize(URL location, ResourceBundle resources) {
        System.out.println(TAG + "StartUp fxml is starting to run");
        loadingDialog = new LoadingDialog(stackPane, vBox);
    }

    /**
     * Set the StartUpObserver to be used when the Controller has to process an action.
     *
     * @param observer the StartUpObserver linked to this Controller.
     */
    public void setObserver(StartUpObserver observer) {
        System.out.println("Observer has been set");
        this.observer = observer;
    }

    /**
     * Called when the button to load documents is pressed.
     */
    public void loadDocuments() {
```

```
    System.out.println(TAG + "load documents");
    if (observer != null) {
        observer.loadFiles();//should disable button to not load in more files while
            doing this
    }
}

/**
 * When the close menu item is pressed.
 */
@Override
public void close() {
    if (observer != null) {
        observer.close();
    }
}

/**
 * When the about menu item is pressed.
 */
@Override
public void about() {
    if (observer != null) {
        observer.showAbout();
    }
}

/**
 * When the timeline menu item is pressed.
 */
@Override
public void timeline() {
    if (observer != null) {
        observer.timeline();
    }
}
```

```
    /**
     * When the preferences menu item is pressed.
     */
    @Override
    public void preferences() {
        if (observer != null) {
            observer.preferences();
        }
    }

    /**
     * Set whether or not the Load Documents Button should be disabled (so that it cannot
         be pressed)
     *
     * @param disable whether or not the Load Documents Button should be disabled.
     */
    public void setDisableLoadDocumentsButton(boolean disable) {
        loadDocumentsButton.setDisable(disable);
    }

    /**
     * Called to show the loading dialog. (Only if the layouts have been passed to
         LoadingDialog)
     */
    public void showLoadingDialog() {
        loadingDialog.showLoadingDialog();
    }

    /**
     * Called to remove the loading dialog.
     */
    public void removeLoadingDialog() {
        loadingDialog.removeLoadingDialog();
    }
}
```

TimelineRowController.java

```
package frontend.controllers;


import backend.process.Result;
import frontend.dialogs.EditEventDialog;
import frontend.observers.DocumentReaderObserver;
import frontend.observers.TimelineRowObserver;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Dialog;
import javafx.scene.control.Label;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.Pane;
import javafx.stage.Stage;


import java.io.IOException;
import java.util.Optional;
import java.util.function.Consumer;


/**
 * Controller class for each row in the timeline listview.
 */
public class TimelineRowController {
    @FXML
    private Group group;
    @FXML
    private Label eventNumberLabel;
    @FXML
    private Label dateLabel;
    @FXML
```

```
    private Label subjectsLabel;
    @FXML
    private Label eventLabel;
    @FXML
    private BorderPane borderPane;
    @FXML
    private Button editButton;
    @FXML
    private Button viewButton;
    @FXML
    private Label fromLabel;
    private int position;
    private Result result;
    private TimelineRowObserver timelineRowObserver;

    /**
     * Loads the layout for the row (appropriate layout picked depending on whether row is
         even or not).
     *
     * @param position the position this row is in the timeline (to determine if its odd or
         even and to display its index).
     */
    public TimelineRowController(int position, TimelineRowObserver timelineRowObserver) {
        this.position = position;
        this.timelineRowObserver = timelineRowObserver;
        FXMLLoader fxmlLoader;
        boolean isEven = (position % 2) == 0;
        if (isEven) {
            fxmlLoader = new FXMLLoader(getClass().getResource("timelineRowEven.fxml"));
        } else {
            fxmlLoader = new FXMLLoader(getClass().getResource("timelineRowOdd.fxml"));
        }
        fxmlLoader.setController(this);
        try {
            fxmlLoader.load();
        } catch (IOException e) {
```

```
            e.printStackTrace();
        }
    }

    /**
     * For the row, where the layout has already been loaded, set the values at the labels,
         and the event handlers for
     * the buttons.
     *
     * @param result the Result object with wich the values of the labels will be populated
         (e.g. date, subjects, event,
     *             etc.).
     */
    public void setData(Result result) {
        this.result = result;
        eventNumberLabel.setText("Event #" + (position + 1));
        dateLabel.setText("Date: " + result.getTimelineDate().toString());
        subjectsLabel.setText("Subjects: " + result.getSubjectsAsString());
        eventLabel.setText("Event: " + result.getEvent());
        fromLabel.setText("From: " + result.getFileData().getFileName() + " (" +
            result.getFileData().getCreationDateFormattedDayMonthYear() + ")");
        borderPane.setStyle("-fx-border-color: black; -fx-border-width: 4;
            -fx-border-style: solid inside;");
        editButton.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                System.out.println("Edit button for timeline event: " +
                    result.getTimelineDate() + " has been pressed");
                Dialog dialog = EditEventDialog.getEditEventDialog(result, (position + 1));
                Optional<EditEventDialog.DialogResult> response = dialog.showAndWait();
                response.ifPresent(new Consumer<EditEventDialog.DialogResult>() {
                    @Override
                    public void accept(EditEventDialog.DialogResult dialogResult) {
                        if (dialogResult.getResultType() ==
                            EditEventDialog.DialogResult.ResultType.DELETE) {
                            System.out.println("Delete the event");
```

```
                    timelineRowObserver.delete(position);
                } else if (dialogResult.getResultType() ==
                     EditEventDialog.DialogResult.ResultType.SAVE) {
                    System.out.println("Update the timeline");
                    Result copy = dialogResult.getResult();
                    timelineRowObserver.update(copy, position);
                } else if (dialogResult.getResultType() ==
                     EditEventDialog.DialogResult.ResultType.CANCEL) {
                    System.out.println("Dont do anything");
                }
            }
        });
    }
});
viewButton.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        System.out.println("Go to Document button for timeline event: " +
             result.getTimelineDate() + " has been pressed");
        Stage stage = new Stage();
        DocumentReaderController documentReaderController = new
             DocumentReaderController(result, new DocumentReaderObserver() {
            @Override
            public void close() {
                System.out.println("Closing document reader window");
                stage.close();
            }
        });
        Pane rootLayout = documentReaderController.getRootBorderPane();
        if (rootLayout != null) {
            stage.setScene(new Scene(documentReaderController.getRootBorderPane(),
                 1024, 800));
            stage.setTitle("Document Reader - " +
                 result.getFileData().getFileName());
            stage.show();
        }
```

```
            }
        });
    }

    /**
     * After the layout has been loaded (and the data has been populated), return the
          layout to use it in the timeline.
     *
     * @return the parent layout of the row in the timeline.
     */
    public Group getGroup() {
        return group;
    }
}
```

C.5.2 Dialogs

AboutDialog.java

```
package frontend.dialogs;

import javafx.scene.control.ButtonType;
import javafx.scene.control.Dialog;
import javafx.scene.control.Label;
import javafx.scene.layout.GridPane;

/**
 * Class to make the About Dialog that shows information of the Application.
 */
public class AboutDialog {
    private final static String AUTHOR = "Oliver Philip Hohn";
    private final static String APP_NAME = "Automated Timeline Extraction";
    private final static String DEFAULT_VERSION = "DEVELOPMENT";

    /**
```

```
    * Get a Dialog that shows the information of the System (the version number, who
        created it, etc.)
    *
    * @return a Dialog that shows the information of the System.
    */
   public Dialog getAboutDialog() {
       Dialog aboutDialog = new Dialog();
       aboutDialog.setTitle("About");
       aboutDialog.setHeaderText(null);

       //layout of body gridpanes
       GridPane gridPane = new GridPane();
       String version = (getClass().getPackage().getImplementationVersion() != null) ?
           getClass().getPackage().getImplementationVersion() : DEFAULT_VERSION;
       Label appNameLabel = new Label("App Name: ");
       Label appLabel = new Label(APP_NAME + " v" + version);
       Label authorLabel = new Label(AUTHOR);
       Label authorNameLabel = new Label("Created By ");
       gridPane.add(appNameLabel, 0, 0);
       gridPane.add(appLabel, 1, 0);
       gridPane.add(authorNameLabel, 0, 1);
       gridPane.add(authorLabel, 1, 1);
       aboutDialog.getDialogPane().setContent(gridPane);
       aboutDialog.getDialogPane().getButtonTypes().setAll(ButtonType.OK);
       return aboutDialog;
   }

}
```

EditEventDialog.java

```
package frontend.dialogs;

import backend.process.Result;
import frontend.controllers.EditEventController;
import frontend.observers.EditEventDialogObserver;
```

```
import javafx.event.EventHandler;
import javafx.scene.Node;
import javafx.scene.control.Alert;
import javafx.scene.control.ButtonBar;
import javafx.scene.control.ButtonType;
import javafx.scene.control.Dialog;
import javafx.scene.input.MouseEvent;
import javafx.util.Callback;

import java.util.Optional;

/**
 * Class that creates an EditEvent Dialog and returns it.
 */
public class EditEventDialog {
    /**
     * Called to produce a edit Dialog for the given Result object. Any changes made do not
         change the given Result
     * object (as it is cloned).
     *
     * @param result the Result object for which this Edit Dialog needs to be produced
         (i.e. to populate the fields).
     * @param position the position the Result is in the timeline (to show at the top of
         the Edit Dialog)
     * @return a Dialog where if the user decides to Save the changes made then its result
         will be a new Result object with the changes made by the
     * user (which can be the same as the given Result object if no changes are made, or a
         new Result object with the data of the previous plus the changes made).
     * This Result object is encapsulated in a DialogResult, that holds the option selected
         by the user (SAVE, DELETE, CANCEL),
     * to determine what to do with the copy Result object and the original Result object
         in the ListView.
     */
    public static Dialog<DialogResult> getEditEventDialog(Result result, int position) {
        //make a copy of the passed in Result object, use it to change values, and pass
            that to return.
```

```
Dialog<DialogResult> dialog = new Dialog<>();
try {
    Result copyResult = (Result) result.clone();
    DialogResult dialogResult = new DialogResult(copyResult);
    dialog.setTitle("Editing Event");
    dialog.setHeaderText("Event #" + position);
    ButtonType buttonTypeSave = new ButtonType("Save",
        ButtonBar.ButtonData.OK_DONE);
    ButtonType buttonTypeCancel = new ButtonType("Cancel",
        ButtonBar.ButtonData.CANCEL_CLOSE);
    ButtonType buttonTypeDelete = new ButtonType("Delete",
        ButtonBar.ButtonData.LEFT);

    dialog.getDialogPane().getButtonTypes().addAll(buttonTypeDelete,
        buttonTypeSave, buttonTypeCancel);
    Node deleteButton = dialog.getDialogPane().lookupButton(buttonTypeDelete);
    deleteButton.setOnMousePressed(new EventHandler<MouseEvent>() {
        @Override
        public void handle(MouseEvent event) {
            System.out.println("Delete Pressed");
            //show dialog to delete event, to confirm etc, tell listener they need
                to delete this event
            Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
            alert.setTitle("Delete");
            alert.setHeaderText(null);
            alert.setContentText("Are you sure you want to delete?");
            alert.getButtonTypes().setAll(ButtonType.YES, ButtonType.NO);
            Optional<ButtonType> result = alert.showAndWait();
            if (result.isPresent() && result.get() == ButtonType.YES) {
                //delete event
                dialogResult.setResultType(DialogResult.ResultType.DELETE);
                //then stop showing
                if (dialog.isShowing()) {
                    dialog.close();
                }
            }
```

```
            }
        });
        dialog.getDialogPane().setContent(new EditEventController(copyResult, new
             EditEventDialogObserver() {
            @Override
            public void disableSave(boolean disableSave) {
                Node saveButton = dialog.getDialogPane().lookupButton(buttonTypeSave);
                saveButton.setDisable(disableSave);
            }
        }).getRootGridPane());
        dialog.setResultConverter(new Callback<ButtonType, DialogResult>() {
            @Override
            public DialogResult call(ButtonType param) {
                if (param.equals(buttonTypeSave)) {
                    dialogResult.setResultType(DialogResult.ResultType.SAVE);
                }
                return dialogResult;
            }
        });
    } catch (CloneNotSupportedException e) {
        e.printStackTrace();
    }
    return dialog;
}

/**
 * Class that holds the option selected by the user in the Edit Event Dialog, and the
     Result object with the changes
 * made it to it through the input fields in the Dialog.
 */
public static class DialogResult {
    /**
     * The different options the user has available in the Edit Event Dialog.
     */
    public enum ResultType {
        SAVE, CANCEL, DELETE
```

```
    }

    private Result result;
    private ResultType resultType;

    /**
     * To create a new DialogResult object.
     *
     * @param result the copy of the Result object that the user will modify with in
         the input fields.
     */
    DialogResult(Result result) {
        this.result = result;
        this.resultType = ResultType.CANCEL;
    }

    /**
     * Getter for the copy of the Result object that the user modified with the input
         fields.
     *
     * @return the copy of the Result object.
     */
    public Result getResult() {
        return result;
    }

    /**
     * Getter for the option selected by the User when they leave the Edit Event Dialog.
     *
     * @return the option selected.
     */
    public ResultType getResultType() {
        return resultType;
    }

    /**
```

```
        * Setter for the option selected by the User when they leave the Edit Event Dialog.
        *
        * @param resultType the option selected.
        */
       void setResultType(ResultType resultType) {
           this.resultType = resultType;
       }
   }
}
```

FileConfirmationDialog.java

```
package frontend.dialogs;


import backend.process.FileData;
import frontend.helpers.TextFieldState;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.css.PseudoClass;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.control.*;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.Priority;


import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.List;


/**
 * Used to create and the confirmation dialog that will contain the filenames, and
     editable base dates for the File.
 */
public class FileConfirmationDialog {
    private final PseudoClass errorClass = PseudoClass.getPseudoClass("error");
```

```
private final SimpleDateFormat validInputFormat = new SimpleDateFormat("dd-MM-yyyy");
private ArrayList<TextFieldState> textFieldStates;

/**
 * For the List of FileData, produce an Alert Confirmation Dialog, where the filename
     and file creation date are
 * shown, so that the user can determine whether or not to use it as a base date.
 *
 * @param fileDatas the List of FileData for which we need to create the list of labels
     and text fields in the dialog.
 * @return an Alert object that can be shown for the user to confirm to use these dates
     for the given files.
 */
public Alert getConfirmationFileDialog(List<FileData> fileDatas) {
    Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
    alert.setTitle("Confirm the Base Dates");
    alert.setContentText("The following Base Dates will be used for the given Files.
        Change them appropriately: ");

    ScrollPane scrollPane = new ScrollPane();
    scrollPane.setPadding(new Insets(10));
    scrollPane.setFitToWidth(true);//its content should resize to fit the width of the
        scroll pane (if it gets resized)
    GridPane fileDateGridPane = new GridPane();
    fileDateGridPane.setMaxWidth(Double.MAX_VALUE);
    fileDateGridPane.setPadding(new Insets(10));
    fileDateGridPane.setHgap(10);//add spacing between the cells in the gridpane
    fileDateGridPane.setVgap(10);
    scrollPane.setContent(fileDateGridPane);
    textFieldStates = new ArrayList<>(fileDatas.size());
    for (int i = 0; i < fileDatas.size(); i++) {
        FileData fileData = fileDatas.get(i);
        Label fileNameLabel = new Label(fileData.getFileName());
        TextField dateTextField = new
            TextField(fileData.getCreationDateFormattedDayMonthYear());
```

```
dateTextField.setPromptText("dd-MM-yyyy");//give the user a prompt in case they
    dont know what the input format is
dateTextField.getStylesheets().add(getClass().getResource("customErrorFields.css")
    .toExternalForm());
dateTextField.setMinWidth(150);//150 pixels is enough for 12 characters
dateTextField.setMaxHeight(35);//enough for one line, if text size is 15
dateTextField.setAlignment(Pos.CENTER);
textFieldStates.add(i, TextFieldState.CORRECT);
GridPane.setVgrow(dateTextField, Priority.NEVER);//1 line always
GridPane.setHgrow(dateTextField, Priority.ALWAYS);
int finalI = i;//to input in the array list at this index
dateTextField.focusedProperty().addListener(new ChangeListener<Boolean>() {
    @Override
    public void changed(ObservableValue<? extends Boolean> observable, Boolean
        oldValue, Boolean newValue) {
        if (!newValue) {
            //check validate text field input
            String input = dateTextField.getText().trim();
            System.out.println("Inputted: " + input);
            if (isValidDateInput(input)) {
                dateTextField.pseudoClassStateChanged(errorClass, false);
                fileData.setCreationDate(input);
                textFieldStates.set(finalI, TextFieldState.CORRECT);
                if (isAllCorrect(textFieldStates)) {
                    alert.getDialogPane().lookupButton(ButtonType.OK)
                        .setDisable(false);
                }
            } else {
                dateTextField.pseudoClassStateChanged(errorClass, true);
                textFieldStates.set(finalI, TextFieldState.WRONG);
                alert.getDialogPane().lookupButton(ButtonType.OK)
                    .setDisable(true);
            }
        }
    }
});
```

```
            fileDateGridPane.add(fileNameLabel, 0, i);
            fileDateGridPane.add(dateTextField, 1, i);
        }
        alert.getDialogPane().setExpandableContent(scrollPane);
        //alert.getDialogPane().setPrefSize(480,250);
        return alert;
    }

    /**
     * Whether or not all the states in the given list of TextFieldStates are CORRECT
         (true) or not (false).
     *
     * @param textFieldStates a list of TextFieldStates to check
     * @return true if all TextFieldStates in the given input are CORRECT, false otherwise.
     */
    private boolean isAllCorrect(ArrayList<TextFieldState> textFieldStates) {
        boolean toReturn = true;
        for (TextFieldState textFieldState : textFieldStates) {
            if (textFieldState == TextFieldState.WRONG) {
                toReturn = false;
                break;
            }
        }
        System.out.println("Checked: " + textFieldStates + " returned: " + toReturn);
        return toReturn;
    }

    /**
     * Checks whether the given input is of the valid format (dd-MM-yyyy).
     *
     * @param input a String date
     * @return true if the input is of the format dd-MM-yyyy, false otherwise.
     */
    private boolean isValidDateInput(String input) {
        try {
```

```
            validInputFormat.setLenient(false);
            validInputFormat.parse(input);
            return input.length() == 10;
        } catch (ParseException e) {
            return false;
        }
    }
}
```

LoadingDialog.java

```
package frontend.dialogs;

import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.control.ProgressIndicator;
import javafx.scene.layout.*;
import javafx.scene.text.Text;

/**
 * Class used to show/hide the loading dialog in the current shown screen
 */
public class LoadingDialog {
    private StackPane stackPane;
    private VBox loadingDialog;
    private Pane mainLayout;

    /**
     * Creates a constructor used to show a loading dialog, and remove it.
     *
     * @param stackPane the Root layout of the window. (Has to be stack pane, as we are
         adding a loading dialog on top
     *                  of the main layout of the window).
     * @param mainLayout the main content layout, which is disabled while the loading
         dialog is being shown (to avoid
     *                  the user from pressing buttons while the system is "Loading".
```

```
     */
    public LoadingDialog(StackPane stackPane, Pane mainLayout) {
        this.stackPane = stackPane;
        this.mainLayout = mainLayout;//the layout on which we are adding on top the loading
            dialog
    }

    /**
     * If the loading dialog (the VBox layout) has not been initialized, ie its null, then
         set it up: creates a layout
     * with text and a progress wheel, indicating the System is loading.
     */
    private void initializeLoadingDialog() {
        if (this.loadingDialog == null) {
            VBox loadingDialog = new VBox();//main layout to hold the dialog
            loadingDialog.getStyleClass().add("loading-dialog");//for the css
            loadingDialog.setPadding(new Insets(20, 15, 20, 15));//add padding
            loadingDialog.getStylesheets().add(getClass().getResource("loadingDialog.css")
                  .toExternalForm());//load the css styles
            //add the "title" of the dialog
            Text pleaseWaitText = new Text("Please Wait... ");
            pleaseWaitText.getStyleClass().add("please-wait-text");
            loadingDialog.getChildren().add(pleaseWaitText);
            loadingDialog.setMargin(pleaseWaitText, new Insets(0, 0, 10, 0));
            //the "body" of the dialog: the progress indicator and text
            GridPane gridPane = new GridPane();
            gridPane.setAlignment(Pos.CENTER);
            gridPane.setVgap(10);
            gridPane.setHgap(28);
            //add progress circle
            ProgressIndicator progressIndicator = new ProgressIndicator();
            progressIndicator.setProgress(-1.0f);
            gridPane.add(progressIndicator, 0, 0, 1, 1);
            //add the text
            Text processingText = new Text("Processing Files... ");
            processingText.getStyleClass().add("processing-text");
```

```
        gridPane.add(processingText, 1, 0, 2, 1);//want the text to be wider than the
            progress indicator
        //add the body
        loadingDialog.getChildren().add(gridPane);
        StackPane.setAlignment(loadingDialog, Pos.CENTER);//to the center of the given
            stack pane
        loadingDialog.setPrefSize(VBox.USE_COMPUTED_SIZE, VBox.USE_COMPUTED_SIZE);//so
            that the dialog is never squashed
        loadingDialog.setMaxWidth(VBox.USE_PREF_SIZE);//and never stretched, but
            instead fit to the size of its content
        loadingDialog.setMaxHeight(VBox.USE_PREF_SIZE);
        this.loadingDialog = loadingDialog;
    }
}

/**
 * Will initialize the loading layout if it hasn't been. If the provided StackPane and
     main layout Pane aren't null,
 * the loading dialog will be shown on top of the layout in the Stack Pane, and the
     main layout will be disabled
 * until the System finishes "loading".
 */
public void showLoadingDialog() {
    initializeLoadingDialog();
    if (stackPane != null && mainLayout != null) {
        mainLayout.setDisable(true);
        stackPane.getChildren().add(loadingDialog);
    }
}

/**
 * Used to indicate the System has finished "loading". If the Pane's aren't null then
     the main layout Pane will be
 * enabled again, and the loading dialog will be removed from on top of the Stack Pane.
 * If the loadingDialog has not been shown, then nothing happens (the main layout is
     enabled - but it should already
```

```
     * be enabled).
     */
    public void removeLoadingDialog() {
        if (stackPane != null && mainLayout != null) {
            mainLayout.setDisable(false);
            if (loadingDialog != null) {
                stackPane.getChildren().remove(loadingDialog);
            }
        }
    }
}
```

RemoveConfirmationDialog.java

```
package frontend.dialogs;


import javafx.scene.control.Alert;
import javafx.scene.control.ButtonType;


/**
 * Used to produce a Confirmation Dialog to ask whether the events for a specified File
      should be removed from the
 * Timeline.
 */
public class RemoveConfirmationDialog {
    /**
     * Produces a Dialog to allow the User to confirm whether or not the events related to
          the selected File should be
     * removed or not.
     *
     * @param fileName the name of the File which links to the events to be removed.
     * @return a Confirmation Alert Dialog with options to remove or not the events linked
          to the given File name.
     */
    public static Alert getRemoveConfirmationDialog(String fileName) {
        Alert removeConfirmationDialog = new Alert(Alert.AlertType.CONFIRMATION);
```

```
        removeConfirmationDialog.setTitle("Removing File");
        removeConfirmationDialog.setContentText("Are you sure you want to remove the events
            for " + fileName + "?");
        removeConfirmationDialog.getButtonTypes().setAll(ButtonType.YES, ButtonType.NO,
            ButtonType.CANCEL);
        return removeConfirmationDialog;
    }
}
```

SettingsDialog.java

```
package frontend.dialogs;

import backend.system.BackEndSystem;
import backend.system.Settings;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.scene.Node;
import javafx.scene.control.*;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.GridPane;
import javafx.scene.text.Text;
import javafx.util.Callback;
import javafx.util.StringConverter;

import java.util.Optional;

/**
 * Class to create the Settings Dialog.
 */
public class SettingsDialog {
    private Spinner<Integer> threadCountSpinner;
    private Spinner<Integer> thresholdSpinner;
    private Spinner<Integer> widthSpinner;
    private Spinner<Integer> heightSpinner;
```

```
/**
 * Get a Settings Dialog, that will allow the user to change the Settings of the System
     and either Save them or
 * discard them. The option to return to the Default Settings is also added. The
     changed Settings would be applied
 * to the cloned Settings held in the BackEndSystem, which should be then set to the
     Settings of the System.
 *
 * @return the Settings Dialog.
 * @throws CloneNotSupportedException when cloning the BackEndSystem Settings.
 */
public Dialog<Settings> settingsDialog() throws CloneNotSupportedException {
    Settings copy = (Settings) BackEndSystem.getInstance().getSettings().clone();
    Dialog<Settings> settingsDialog = new Dialog<>();
    //set up dialog layout
    settingsDialog.setTitle("Preferences");
    settingsDialog.setHeaderText(null);
    settingsDialog.getDialogPane().setContent(getDialogLayout(copy));

    //buttons
    ButtonType buttonTypeSave = new ButtonType("Save", ButtonBar.ButtonData.OK_DONE);
    ButtonType buttonTypeCancel = new ButtonType("Cancel",
        ButtonBar.ButtonData.CANCEL_CLOSE);
    ButtonType buttonTypeDefault = new ButtonType("Default", ButtonBar.ButtonData.LEFT);
    settingsDialog.getDialogPane().getButtonTypes().setAll(buttonTypeDefault,
        buttonTypeSave, buttonTypeCancel);

    //set default onaction
    Node defaultButton = settingsDialog.getDialogPane().lookupButton(buttonTypeDefault);
    defaultButton.setOnMousePressed(new EventHandler<MouseEvent>() {
        @Override
        public void handle(MouseEvent event) {
            Alert confirmation = new Alert(Alert.AlertType.CONFIRMATION);
            confirmation.setTitle("Reset to Default?");
            confirmation.setHeaderText(null);
```

```
            confirmation.setContentText("Are you sure you wish reset the Settings to
                 Default? ");
            confirmation.getDialogPane().getButtonTypes().setAll(ButtonType.YES,
                 ButtonType.NO);
            Optional<ButtonType> response = confirmation.showAndWait();
            if (response.isPresent() && response.get() == ButtonType.YES) {
                //reset to default
                copy.reset();
                settingsDialog.getDialogPane().setContent(getDialogLayout(copy));
            }
        }
    });

    //set result
    settingsDialog.setResultConverter(new Callback<ButtonType, Settings>() {
        @Override
        public Settings call(ButtonType param) {
            if (param == buttonTypeSave) {
                copy.setWidth(widthSpinner.getValue());
                copy.setHeight(heightSpinner.getValue());
                copy.setThresholdSummary(thresholdSpinner.getValue());
                copy.setMaxNoOfThreads(threadCountSpinner.getValue());
                return copy;
            }
            return null;
        }
    });
    return settingsDialog;
}

/**
 * For the Settings Dialog, get its Content Root Layout: a GridPane with Text and
     Spinners for the different Settings
 * options. The given Settings are used to populate the fields in the layout.
 *
 * @param settings the given Settings.
```

```
     * @return the root Content layout of the Dialog.
     */
    private GridPane getDialogLayout(Settings settings) {
        GridPane gridPane = new GridPane();
        gridPane.setMaxWidth(Double.MAX_VALUE);
        gridPane.setPadding(new Insets(10));
        gridPane.setHgap(10);
        gridPane.setVgap(10);

        Text fileTitle = new Text("File Processing");
        gridPane.add(fileTitle, 0, 0);

        Text threadText = new Text("Maximum Number of Threads running in parallel: ");
        threadCountSpinner = new Spinner<>();
        SpinnerValueFactory.IntegerSpinnerValueFactory spinnerValueFactory = new
            SpinnerValueFactory.IntegerSpinnerValueFactory(1, 20,
            settings.getMaxNoOfThreads());
        spinnerValueFactory.setAmountToStepBy(1);
        spinnerValueFactory.setConverter(new IntegerStringConverter(threadCountSpinner));
        threadCountSpinner.setValueFactory(spinnerValueFactory);
        threadCountSpinner.setEditable(true);
        gridPane.add(threadText, 0, 1);
        gridPane.add(threadCountSpinner, 1, 1);

        Text thresholdText = new Text("Threshold of Text Summary: ");
        thresholdSpinner = new Spinner<>();
        spinnerValueFactory = new SpinnerValueFactory.IntegerSpinnerValueFactory(1, 20,
            settings.getThresholdSummary());
        spinnerValueFactory.setAmountToStepBy(1);
        spinnerValueFactory.setConverter(new IntegerStringConverter(thresholdSpinner));
        thresholdSpinner.setValueFactory(spinnerValueFactory);
        thresholdSpinner.setEditable(true);
        gridPane.add(thresholdText, 0, 2);
        gridPane.add(thresholdSpinner, 1, 2);

        Separator separator = new Separator();//by default its horizontal
```

```
        gridPane.add(separator, 0, 3, 2, 1);

        Text appearanceText = new Text("Appearance");
        gridPane.add(appearanceText, 0, 4);

        Text widthText = new Text("Width at Startup: ");
        widthSpinner = new Spinner<>();
        spinnerValueFactory = new SpinnerValueFactory.IntegerSpinnerValueFactory(720, 1920,
            settings.getWidth());
        spinnerValueFactory.setAmountToStepBy(5);
        spinnerValueFactory.setConverter(new IntegerStringConverter(widthSpinner));
        widthSpinner.setValueFactory(spinnerValueFactory);
        widthSpinner.setEditable(true);
        gridPane.add(widthText, 0, 5);
        gridPane.add(widthSpinner, 1, 5);

        Text heightText = new Text("Height at Startup: ");
        heightSpinner = new Spinner<>();
        spinnerValueFactory = new SpinnerValueFactory.IntegerSpinnerValueFactory(600, 1080,
            settings.getHeight());
        spinnerValueFactory.setAmountToStepBy(5);
        spinnerValueFactory.setConverter(new IntegerStringConverter(heightSpinner));
        heightSpinner.setValueFactory(spinnerValueFactory);
        heightSpinner.setEditable(true);
        gridPane.add(heightText, 0, 6);
        gridPane.add(heightSpinner, 1, 6);

        return gridPane;
    }

    /**
     * Private Class to deal with the conversion of Strings to Ints in the Spinner text
         fields.
     */
    private static class IntegerStringConverter extends StringConverter<Integer> {
        Spinner<Integer> spinner;
```

```
/**
 * Constructor that takes in the Spinner where the changes will be applied on.
 *
 * @param spinner the given Spinner.
 */
IntegerStringConverter(Spinner<Integer> spinner) {
    this.spinner = spinner;
}

/**
 * For the given Integer held in the Spinner, show its String version.
 *
 * @param object the given Integer.
 * @return the Integer as a String.
 */
@Override
public String toString(Integer object) {
    return String.valueOf(object);
}

/**
 * For the given String (in the textfield) attempt to turn it into an Integer, if
     that is not possible, use the
 * previous value held by the Spinner (which would be an Integer).
 *
 * @param string the text in the textfield.
 * @return an Integer made from the text in the textfield, or the previous
     available Integer.
 */
@Override
public Integer fromString(String string) {
    Integer integer;
    try {
        integer = Integer.parseInt(string);
    } catch (Exception e) {
```

```
                integer = spinner.getValue();
            }
            return integer;
        }
    }
}
```

C.5.3 Helpers

TextFieldState.java

```
package frontend.helpers;


/**
 * Enum class that represents the state of a TextField in the FileConfirmationDialog in
     terms of its input being correct
 * or wrong
 */
public enum TextFieldState {
    CORRECT,
    WRONG
}
```

C.5.4 Observers

DocumentReaderObserver.java

```
package frontend.observers;


/**
 * Interface to be implemented by the Observer of the DocumentReaderController.
 */
public interface DocumentReaderObserver {
    /**
     * Called to allow the Controller to inform the Observer when they need to close the
         window in which the layout
```

```
     * of the Controller resides.
     */
    void close();
}
```

DocumentsLoadedObserver.java

```
package frontend.observers;

import backend.process.FileData;

/**
 * Interface for the Observers of the DocumentLoadedRowController.
 */
public interface DocumentsLoadedObserver {
    /**
     * Called when all the Results connected to the given FileData need to be removed.
          (i.e. removing the File given by
     * FileData from the Timeline).
     * @param fileData the FileData for the File where we want to remove its results from
          the Timeline.
     */
    void remove(FileData fileData);
}
```

EditEventDialogObserver.java

```
package frontend.observers;

/**
 * Implemented by the Observer of the EditEventDialog Content. To inform, from the content
      of the Dialog, to the holder
 * of the Dialog to disable/enable the save button.
 */
public interface EditEventDialogObserver {
```

```
    /**
     * Called when the Dialog Save button should be disabled or enabled.
     */
    void disableSave(boolean disableSave);
}
```

MenuBarObserver.java

```
package frontend.observers;


/**
 * Implemented by Observers of Controllers that contain a Menu bar, to handle the relevant
     events.
 */
public interface MenuBarObserver {
    /**
     * Called when the about menu item is pressed.
     */
    void showAbout();


    /**
     * Called when the close menu item is pressed.
     */
    void close();


    /**
     * Called when the timeline menu item is pressed.
     */
    void timeline();


    /**
     * Called when the preferences menu item is pressed.
     */
    void preferences();
}
```

StartUpObserver.java

```java
package frontend.observers;


/**
 * Interface for the Observer of the StartUpController. It extends the MenuBarObserver as
     the StartUp scene has a menu
 * bar.
 */
public interface StartUpObserver extends MenuBarObserver {//menu options included


    /**
     * Called when the load files button is pressed.
     */
    void loadFiles();
}
```

TimelineObserver.java

```java
package frontend.observers;


import backend.process.Result;


import java.util.List;


/**
 * Interface for the Observer of the Timeline scene.
 */
public interface TimelineObserver extends MenuBarObserver {
    /**
     * Called when the "Load Documents" button is pressed.
     */
    void loadDocuments();


    /**
     * Called when the "Save to ..." button is pressed.
```

```
     * @param resultList the List of Results to save.
     */
    void saveTo(List<Result> resultList);
}
```

TimelineRowObserver.java

```
package frontend.observers;

import backend.process.Result;

/**
 * Implemented by the Observer of this TimelineRow (that is the holder of the ListView),
     so that the cells can communicate
 * with the overall listView.
 */
public interface TimelineRowObserver {
    /**
     * Called by a TimelineRow to inform the Observer (holder of the ListView), that the
         data it is holding has been
     * updated.
     *
     * @param updatedResult the updated data of this TimelineRow.
     * @param position    the position this cell had in the ListView.
     */
    void update(Result updatedResult, int position);

    /**
     * Called by a TimelineRow to inform the Observer (holder of the ListView), that the
         data it is holding has been
     * updated.
     *
     * @param previous    the Result object that was previously in this Row.
     * @param updatedResult the updated Result object of the TimelineRow.
     */
    void update(Result previous, Result updatedResult);
```

```
    /**
     * Called by a TimelineRow to inform the Observer (holder of the ListView), that this
         event (in the given position),
     * needs to be deleted from the list.
     *
     * @param position the position of the event to be deleted.
     */
    void delete(int position);

    /**
     * Called by a TimelineRow to inform the Observer (holder of the ListView), that this
         event (the given Result) needs
     * to be deleted from the list.
     *
     * @param result the given event (Result) to be deleted.
     */
    void delete(Result result);
}
```

C.5.5 Main.java

```
package frontend;


import backend.helpers.ToJSON;
import backend.helpers.ToPDF;
import backend.process.FileData;
import backend.process.ProcessFiles;
import backend.process.Result;
import backend.system.BackEndSystem;
import backend.system.Settings;
import frontend.controllers.ListViewController;
import frontend.controllers.StartUpController;
import frontend.dialogs.AboutDialog;
import frontend.dialogs.FileConfirmationDialog;
import frontend.dialogs.SettingsDialog;
```

```
import frontend.observers.StartUpObserver;
import frontend.observers.TimelineObserver;
import javafx.application.Application;
import javafx.concurrent.Task;
import javafx.concurrent.WorkerStateEvent;
import javafx.event.EventHandler;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.ButtonBar;
import javafx.scene.control.ButtonType;
import javafx.scene.control.Dialog;
import javafx.stage.FileChooser;
import javafx.stage.Stage;

import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
import java.util.function.Consumer;

/**
 * Main class that is used to run the program (i.e. show the UI that uses the backend). It
     handles loading the different
 * layouts, and being the observer of the layouts controllers.
 */
public class Main extends Application implements StartUpObserver, TimelineObserver {
    private final static String TAG = "MAIN: ";
    private Stage primaryStage;
    private StartUpController startUpController;
    private ListViewController listViewController;

    /**
```

```
     * Called to start showing the window of the program (i.e. the please load documents
         layout).
     *
     * @param primaryStage the root window of the application
     * @throws Exception
     */
    @Override
    public void start(Stage primaryStage) throws Exception {
        //need to start engine
        BackEndSystem.getInstance();//thread waits for this to be done
        Settings settings = BackEndSystem.getInstance().getSettings();
        System.out.println("Called getInstance");
        FXMLLoader fxmlLoader = new FXMLLoader(getClass().getResource("startup.fxml"));
        primaryStage.setScene(new Scene(fxmlLoader.load(), settings.getWidth(),
             settings.getHeight()));
        primaryStage.setTitle("Automated Timeline Extractor - Oliver Philip Hohn");
        startUpController = fxmlLoader.getController();
        startUpController.setObserver(this);
        primaryStage.show();
        this.primaryStage = primaryStage;
    }

    /**
     * First method that gets called when the program runs.
     *
     * @param args the arguments passed in when the instruction to run the program is given.
     */
    public static void main(String[] args) {
        launch(args);
    }

    /**
     * Produces a list of Files, picked by the user in the File Chooser.
     *
     * @param primaryStage the window where the program is running on.
     * @return a list of Files picked by the user in the File Chooser.
```

```
     */
    private List<File> loadFiles(Stage primaryStage) {
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("Choose Document Files");
        fileChooser.getExtensionFilters().addAll(new FileChooser.ExtensionFilter("Text
            Files", "*.txt", "*.pdf", "*.docx"));
        return fileChooser.showOpenMultipleDialog(primaryStage);
    }

    /**
     * For the given List of Files produce a List of FileData, where each FileData
        corresponds to its File (i.e. have the
     * same index in the list), and holds its Files name, and path.
     *
     * @param files the given List of Files.
     * @return the output List of FileData.
     */
    private List<FileData> getFileData(List<File> files) {
        ArrayList<FileData> toReturn = new ArrayList<>();
        for (File file : files) {
            toReturn.add(new FileData(file));
        }
        return toReturn;
    }

    /**
     * Called to produce a Task object, that will run a set of operations, when given to a
        Thread, in parallel. This
     * Task object will, for the given Lists of Files and FileData, produce the List of
        Results that emerge from
     * processing the text in the Files, and linking each Result object to its
        corresponding FileData.
     *
     * @param files    the given List of Files.
     * @param fileDatas the given List of FileData.
```

```
     * @return a Task object to run in a Thread in parallel, to process the given Files and
         produce Result objects.
     */
    private Task<List<Result>> prepareTask(List<File> files, List<FileData> fileDatas) {
        return new Task<List<Result>>() {
            @Override
            protected List<Result> call() throws Exception {
                ProcessFiles processFiles = new ProcessFiles();
                return processFiles.processFiles(files, fileDatas);
            }
        };
    }

    /**
     * For the global Stage, load the listView layout, set its Observer as Main.this, and
         hold its controller.
     *
     * @return the Controller of the listView layout.
     */
    private ListViewController showListView() {
        FXMLLoader fxmlLoader = new FXMLLoader(getClass().getResource("listView.fxml"));
        try {
            primaryStage.setScene(new Scene(fxmlLoader.load(), primaryStage.getWidth(),
                 primaryStage.getHeight()));
            listViewController = fxmlLoader.getController();
            listViewController.setTimelineObserver(this);
            primaryStage.show();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return listViewController;
    }


    /**
```

```
 * Called from the Controller of the initial layout (the "load-button-layout"), to
     indicate to Main, that it must
 * provide the user the option to allow the User to pick Files, pass those Files to the
     backend to process, and
 * then update the layout to show the timeline of the events depicted in the text of
     the Files.
 * The button to load Files is disabled, to stop the user from pressing it again, while
     the Files are being loaded.
 */
@Override
public void loadFiles() {
    List<File> files = loadFiles(primaryStage);
    if (files != null) {
        List<FileData> fileDatas = getFileData(files);
        Alert fileConfirmationDialog = new
             FileConfirmationDialog().getConfirmationFileDialog(fileDatas);
        Optional<ButtonType> response = fileConfirmationDialog.showAndWait();
        if (response.isPresent() && response.get() == ButtonType.OK) {
            //disable button
            startUpController.setDisableLoadDocumentsButton(true);
            System.out.println(TAG + "Process Files and set them in the Timeline");
            Task<List<Result>> task = prepareTask(files, fileDatas);
            task.setOnSucceeded(new EventHandler<WorkerStateEvent>() {
                @Override
                public void handle(WorkerStateEvent event) {
                    List<Result> results = task.getValue();
                    listViewController = showListView();
                    listViewController.setTimelineListView(results, fileDatas);
                    //stop showing the loading dialog as we have the other layout ready
                         to show
                    startUpController.removeLoadingDialog();
                    startUpController = null;
                    primaryStage.show();
                }
            });
            task.setOnRunning(new EventHandler<WorkerStateEvent>() {
```

```
                @Override
                public void handle(WorkerStateEvent event) {
                    //show loading dialog
                    startUpController.showLoadingDialog();
                }
            });
            new Thread(task).start();
        } else {
            System.out.println(TAG + "Dont Process Files and set them in the Timeline");
        }
    }
}

/**
 * Called when the About MenuItem is pressed. Called from either the ListViewController
     or the StartUpController.
 */
@Override
public void showAbout() {
    System.out.println(TAG + "show about information");
    new AboutDialog().getAboutDialog().showAndWait();
}

/**
 * Called when the Close MenuItem is pressed. Should close the programs (root) window.
 * Called from either the ListViewController or the StartUpController.
 */
@Override
public void close() {
    System.out.println(TAG + "close program");
    primaryStage.close();
}

/**
 * Called when the Timeline MenuItem is pressed. Called from either the
     ListViewController or the StartUpController.
```

```
     * But it is only available from the ListView layout, as in the StartUp layout it is
         disabled.
     */
    @Override
    public void timeline() {
        System.out.println(TAG + "timeline options");
    }

    /**
     * Called when the Preferences MenuItem is pressed in the File Menu.
     */
    @Override
    public void preferences() {
        System.out.println(TAG + "Preferences pressed");
        try {
            Dialog<Settings> settingsDialog = new SettingsDialog().settingsDialog();//show
                 a settings dialog
            Optional<Settings> response = settingsDialog.showAndWait();
            response.ifPresent(new Consumer<Settings>() {
                @Override
                public void accept(Settings settings) {
                    if (settings != null) {//then the user decided to save the settings
                        BackEndSystem.getInstance().setSettings(settings);
                    }//else dont apply the Settings to the System.
                }
            });
        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
        }
    }

    /**
     * Called by the StartUpController, to indicate to Main, to allow the User to pick
         Files (from the File Chooser),
     * process them, and add them to the timeline (instead of setting them like in the
         previous method). The layout is
```

```
     * not changed, as we are already in the timeline layout.
     */
    @Override
    public void loadDocuments() {
        List<File> files = loadFiles(primaryStage);
        if (files != null) {
            List<FileData> fileDatas = getFileData(files);
            Alert fileConfirmationDialog = new
                 FileConfirmationDialog().getConfirmationFileDialog(fileDatas);
            Optional<ButtonType> response = fileConfirmationDialog.showAndWait();
            if (response.isPresent() && response.get() == ButtonType.OK) {
                System.out.println(TAG + "Process Files and Add them to the Timeline");
                Task<List<Result>> task = prepareTask(files, fileDatas);
                task.setOnSucceeded(new EventHandler<WorkerStateEvent>() {
                    @Override
                    public void handle(WorkerStateEvent event) {
                        List<Result> results = task.getValue();
                        if (listViewController != null) {
                            listViewController.addToTimelineListView(results, fileDatas);
                            //stop showing loading dialog as we have added the results
                            listViewController.removeLoadingDialog();
                        }
                    }
                });
                task.setOnRunning(new EventHandler<WorkerStateEvent>() {
                    @Override
                    public void handle(WorkerStateEvent event) {
                        //show loading dialog
                        listViewController.showLoadingDialog();
                    }
                });
                new Thread(task).start();
            } else {
                System.out.println(TAG + "Don't process Files");
            }
        }
```

```
}



/**
 * Used to show the Alert the dialog to allow the User to pick in what format to save
     the List of Results (JSON or
 * PDF). Depending on the option selected, the FileChooser is shown, and then the User
     picks the location to save
 * the file. An Alert is shown if the User is trying to overwrite a File in use when
     saving.
 *
 * @param results the List of Results to Save.
 */
@Override
public void saveTo(List<Result> results) {
    //show alert to know what to save as
    Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
    alert.setTitle("Save to...");
    alert.setHeaderText(null);
    alert.setContentText("Do you wish to save as PDF or JSON?");
    ButtonType buttonTypePDF = new ButtonType("Save to PDF");
    ButtonType buttonTypeJSON = new ButtonType("Save to JSON");
    ButtonType buttonTypeCancel = new ButtonType("Cancel",
         ButtonBar.ButtonData.CANCEL_CLOSE);
    alert.getDialogPane().getButtonTypes().setAll(buttonTypePDF, buttonTypeJSON,
         buttonTypeCancel);

    Optional<ButtonType> response = alert.showAndWait();
    if (response.isPresent() && response.get() == buttonTypePDF) {
        saveToPDF(results);
    } else if (response.isPresent() && response.get() == buttonTypeJSON) {
        saveToJSON(results);
    }
}

/**
```

```
     * Called when the Timeline being displayed (described by the List of Result objects)
         needs to be saved as a JSON.
     * The System should allow the user to pick a location to save the PDF, if it is
         overwriting a File in use by
     * another process the user is informed and given the option to select another location
         or close the process using
     * the File that they wish to overwrite.
     *
     * @param results the given List of Result objects.
     */
    private void saveToJSON(List<Result> results) {
        String json = ToJSON.toJSON(results);
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("Save Timeline As...");
        fileChooser.getExtensionFilters().addAll(new FileChooser.ExtensionFilter("JSON
            File", "*.json"));
        File file = fileChooser.showSaveDialog(primaryStage);
        if (file != null && listViewController != null) {
            try {
                PrintWriter printWriter = new PrintWriter(file);
                printWriter.write(json);
                printWriter.close();
            } catch (Exception e) {
                Alert alert = new Alert(Alert.AlertType.INFORMATION);//then inform the user,
                alert.setTitle("File In Use");
                alert.setHeaderText(null);
                alert.setContentText("The file: " + file.getName() + " is in use by another
                    process.");
                alert.getDialogPane().getButtonTypes().setAll(ButtonType.OK);
                Optional<ButtonType> response = alert.showAndWait();
                if (response.isPresent() && response.get() == ButtonType.OK) {//and if they
                    press OK, ie want to save
                    saveToJSON(results);//show them the file chooser to let them pick a
                        different (or same location, if
                }
            }
```

```
    }
}

/**
 * Called when the Timeline being displayed (described by the List of Result objects)
     needs to be saved as a PDF.
 * The System should allow the user to pick a location to save the PDF, if it is
     overwriting a File in use by
 * another process the user is informed and given the option to select another location
     or close the process using
 * the File that they wish to overwrite.
 *
 * @param results the given List of Result objects.
 */
private void saveToPDF(List<Result> results) {
    System.out.println(TAG + "Save To PDF pressed");
    FileChooser fileChooser = new FileChooser();
    fileChooser.setTitle("Save Timeline As...");
    fileChooser.getExtensionFilters().addAll(new FileChooser.ExtensionFilter("PDF
        File", "*.pdf"));
    File file = fileChooser.showSaveDialog(primaryStage);
    if (file != null && listViewController != null) {
        try {
            new ToPDF().saveToPDF(results, file);
        } catch (IOException e) {//if cant save the file, because it is most probably
            in use or it has been deleted
            Alert alert = new Alert(Alert.AlertType.INFORMATION);//then inform the user,
            alert.setTitle("File In Use");
            alert.setHeaderText(null);
            alert.setContentText("The file: " + file.getName() + " is in use by another
                process.");
            alert.getDialogPane().getButtonTypes().setAll(ButtonType.OK);
            Optional<ButtonType> response = alert.showAndWait();
            if (response.isPresent() && response.get() == ButtonType.OK) {//and if they
                press OK, ie want to save
```

```
                saveToPDF(results);//show them the file chooser to let them pick a
                    different (or same location, if
            }                           //they closed the process
        }
    }
}
```

C.6 src/main/resources/frontend

C.6.1 controllers

customErrorFields.css

```
.text-field:error {
  -fx-text-box-border: red ;
  -fx-focus-color: red ;
}

.text-area:error{
  -fx-text-box-border: red ;
  -fx-focus-color: red ;
}
```

customResultRow.fxml

```
<?xml version="1.0" encoding="UTF-8"?>


<?import javafx.geometry.Insets?>
<?import javafx.scene.Group?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.Separator?>
<?import javafx.scene.layout.ColumnConstraints?>
<?import javafx.scene.layout.GridPane?>
<?import javafx.scene.layout.HBox?>
```

```
<?import javafx.scene.layout.RowConstraints?>
<?import javafx.scene.layout.VBox?>

<VBox fx:id="rootVBox" maxHeight="1.7976931348623157E308"
    maxWidth="1.7976931348623157E308" minHeight="-Infinity" minWidth="-Infinity"
    xmlns="http://javafx.com/javafx/8.0.111" xmlns:fx="http://javafx.com/fxml/1">
  <children>
    <Group>
      <children>
        <GridPane vgap="5.0">
          <columnConstraints>
            <ColumnConstraints hgrow="NEVER" minWidth="10.0" />
            <ColumnConstraints hgrow="NEVER" minWidth="10.0" />
          </columnConstraints>
          <rowConstraints>
            <RowConstraints minHeight="10.0" vgrow="SOMETIMES" />
            <RowConstraints minHeight="10.0" vgrow="ALWAYS" />
              <RowConstraints prefHeight="30.0" vgrow="ALWAYS" />
          </rowConstraints>
           <children>
            <Label text="Subjects: " GridPane.halignment="LEFT"
                GridPane.valignment="TOP" />
            <Label text="Event: " GridPane.halignment="LEFT" GridPane.rowIndex="1"
                GridPane.valignment="TOP" />
            <Label fx:id="subjectsLabel" maxHeight="-Infinity" maxWidth="-Infinity"
                minHeight="-Infinity" minWidth="-Infinity" prefWidth="230.0"
                text="[subjects]" wrapText="true" GridPane.columnIndex="1"
                GridPane.halignment="LEFT" GridPane.valignment="TOP">
              <GridPane.margin>
                <Insets right="5.0" />
              </GridPane.margin>
            </Label>
            <Label fx:id="eventLabel" maxHeight="-Infinity" maxWidth="-Infinity"
                minHeight="-Infinity" minWidth="-Infinity" prefWidth="230.0"
                text="[event]" wrapText="true" GridPane.columnIndex="1"
                GridPane.halignment="LEFT" GridPane.rowIndex="1"
```

```
                    GridPane.valignment="TOP" GridPane.vgrow="ALWAYS">
                  <GridPane.margin>
                     <Insets right="5.0" />
                  </GridPane.margin>
               </Label>
               <Label text="From: " GridPane.halignment="LEFT" GridPane.rowIndex="2"
                    GridPane.valignment="TOP" />
               <Label fx:id="fromLabel" maxHeight="-Infinity" maxWidth="-Infinity"
                    minHeight="-Infinity" minWidth="-Infinity" prefWidth="230.0"
                    text="[from]" wrapText="true" GridPane.columnIndex="1"
                    GridPane.halignment="LEFT" GridPane.rowIndex="2"
                    GridPane.valignment="TOP" GridPane.vgrow="NEVER" />
            </children>
         </GridPane>
      </children>
   </Group>
   <HBox alignment="TOP_RIGHT">
      <children>
         <Button fx:id="editButton" mnemonicParsing="false" text="Edit">
            <HBox.margin>
               <Insets bottom="5.0" left="5.0" right="5.0" top="5.0" />
            </HBox.margin>
         </Button>
         <Button fx:id="viewButton" mnemonicParsing="false" text="View">
            <HBox.margin>
               <Insets bottom="5.0" left="5.0" right="5.0" top="5.0" />
            </HBox.margin>
         </Button>
      </children>
   </HBox>
   <Separator prefWidth="200.0" />
</children>
<padding>
   <Insets bottom="5.0" left="5.0" right="5.0" top="5.0" />
</padding>
</VBox>
```

documentLoadedRow.fxml

```
<?xml version="1.0" encoding="UTF-8"?>


<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.image.Image?>
<?import javafx.scene.image.ImageView?>
<?import javafx.scene.layout.ColumnConstraints?>
<?import javafx.scene.layout.GridPane?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.layout.RowConstraints?>
<?import javafx.scene.text.Font?>


<GridPane fx:id="gridPane" maxHeight="-Infinity" maxWidth="-Infinity"
    minHeight="-Infinity" minWidth="-Infinity" xmlns="http://javafx.com/javafx/8.0.111"
    xmlns:fx="http://javafx.com/fxml/1">
  <columnConstraints>
    <ColumnConstraints hgrow="SOMETIMES" maxWidth="211.0" minWidth="10.0"
        prefWidth="211.0" />
    <ColumnConstraints hgrow="SOMETIMES" maxWidth="94.0" minWidth="10.0" prefWidth="39.0"
        />
  </columnConstraints>
  <rowConstraints>
    <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
  </rowConstraints>
   <children>
      <ImageView fx:id="removingImageView" pickOnBounds="true" preserveRatio="true"
          GridPane.columnIndex="1" GridPane.halignment="RIGHT"
          GridPane.valignment="CENTER">
        <image>
          <Image url="@iccloseblack.png" />
        </image>
      </ImageView>
      <HBox alignment="CENTER_LEFT" prefHeight="100.0" prefWidth="200.0">
        <children>
```

```
            <Label text="Âů">
               <font>
                  <Font size="36.0" />
               </font>
            </Label>
            <Label fx:id="documentLabel" text="Document1.pdf">
               <padding>
                  <Insets left="5.0" />
               </padding>
            </Label>
         </children>
      </HBox>
   </children>
   <padding>
      <Insets bottom="10.0" left="10.0" right="10.0" top="10.0" />
   </padding>
</GridPane>
```

documentReader.fxml

```
<?xml version="1.0" encoding="UTF-8"?>


<?import javafx.geometry.Insets?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.BorderPane?>
<?import javafx.scene.text.Font?>
<?import org.fxmisc.richtext.InlineCssTextArea?>
<BorderPane fx:id="rootBorderPane" maxHeight="-Infinity" maxWidth="-Infinity"
     minHeight="-Infinity" minWidth="-Infinity" prefHeight="400.0" prefWidth="600.0"
     xmlns="http://javafx.com/javafx/8.0.111" xmlns:fx="http://javafx.com/fxml/1">
   <center>
         <InlineCssTextArea fx:id="documentInlineCssTextArea" editable="false"
              wrapText="true" BorderPane.alignment="TOP_LEFT">
            <BorderPane.margin>
               <Insets bottom="5.0" left="5.0" right="5.0" top="5.0" />
            </BorderPane.margin>
```

```
            <padding>
               <Insets bottom="2.0" left="2.0" right="2.0" top="2.0" />
            </padding>
         <font>
            <Font size="18.0" />
         </font>
         </InlineCssTextArea>
   </center>
   <top>
      <MenuBar BorderPane.alignment="CENTER">
        <menus>
          <Menu mnemonicParsing="false" text="File">
            <items>
              <MenuItem fx:id="closeMenuItem" mnemonicParsing="false" text="Close" />
            </items>
          </Menu>
          <Menu mnemonicParsing="false" text="Edit">
            <items>
              <MenuItem fx:id="copyMenuItem" mnemonicParsing="false" text="Copy" />
            </items>
          </Menu>
        </menus>
      </MenuBar>
   </top>
</BorderPane>
```

editEventDialog.fxml

```
<?xml version="1.0" encoding="UTF-8"?>


<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.TextArea?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.image.Image?>
<?import javafx.scene.image.ImageView?>
```

```
<?import javafx.scene.layout.BorderPane?>
<?import javafx.scene.layout.ColumnConstraints?>
<?import javafx.scene.layout.GridPane?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.layout.RowConstraints?>
<?import javafx.scene.layout.VBox?>

<GridPane fx:id="rootGridPane" alignment="CENTER" hgap="10.0" maxHeight="-Infinity"
    maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="400.0"
    prefWidth="600.0" vgap="10.0" xmlns="http://javafx.com/javafx/8.0.111"
    xmlns:fx="http://javafx.com/fxml/1">
  <columnConstraints>
    <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />
    <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />
  </columnConstraints>
  <rowConstraints>
    <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
    <RowConstraints maxHeight="46.0" minHeight="10.0" prefHeight="46.0" vgrow="SOMETIMES"
        />
    <RowConstraints maxHeight="305.0" minHeight="10.0" prefHeight="294.0"
        vgrow="SOMETIMES" />
  </rowConstraints>
   <children>
      <Label text="Date: " GridPane.hgrow="NEVER" GridPane.vgrow="NEVER" />
      <Label text="Subjects: " GridPane.hgrow="NEVER" GridPane.rowIndex="1"
          GridPane.valignment="CENTER" GridPane.vgrow="NEVER" />
      <Label text="Event: " GridPane.hgrow="NEVER" GridPane.rowIndex="2"
          GridPane.valignment="TOP" GridPane.vgrow="NEVER">
        <padding>
          <Insets bottom="5.0" right="5.0" top="5.0" />
        </padding>
      </Label>
      <BorderPane GridPane.columnIndex="1" GridPane.halignment="CENTER"
          GridPane.hgrow="ALWAYS" GridPane.valignment="CENTER" GridPane.vgrow="NEVER">
        <left>
```

```
      <TextField fx:id="firstDateTextField" prefHeight="31.0" prefWidth="214.0"
          promptText="dd-MM-yyyy" text="dd-MM-yyyy" BorderPane.alignment="CENTER" />
   </left>
   <center>
      <Label alignment="CENTER" text="-&gt;" BorderPane.alignment="CENTER" />
   </center>
   <right>
      <TextField fx:id="secondDateTextField" prefWidth="214.0"
          promptText="dd-MM-yyyy" BorderPane.alignment="CENTER" />
   </right>
   <padding>
      <Insets bottom="10.0" left="10.0" right="10.0" top="10.0" />
   </padding>
</BorderPane>
<VBox spacing="5.0" GridPane.columnIndex="1" GridPane.rowIndex="1">
   <children>
      <BorderPane prefHeight="200.0" prefWidth="200.0" VBox.vgrow="ALWAYS">
         <center>
            <TextField fx:id="subjectTextField" BorderPane.alignment="CENTER">
               <BorderPane.margin>
                  <Insets right="5.0" />
               </BorderPane.margin>
            </TextField>
         </center>
         <right>
            <ImageView fx:id="addImageView" pickOnBounds="true" preserveRatio="true"
                BorderPane.alignment="CENTER">
               <image>
                  <Image url="@icaddcircleoutline.png" />
               </image>
            </ImageView>
         </right>
         <padding>
            <Insets top="10.0" />
         </padding>
      </BorderPane>
```

Figure C.1: PNG Image used to indicate the adding of subjects to an event.

Figure C.2: PNG Image used for the action of deleting documents from the timeline.

```
            <HBox fx:id="subjectsHBox" prefHeight="100.0" prefWidth="200.0" spacing="5.0" />
         </children>
         <padding>
            <Insets bottom="10.0" left="10.0" right="10.0" top="10.0" />
         </padding>
      </VBox>
      <VBox alignment="TOP_RIGHT" prefHeight="200.0" prefWidth="100.0" spacing="2.0"
          GridPane.columnIndex="1" GridPane.rowIndex="2">
         <children>
            <TextArea fx:id="eventTextArea" prefHeight="200.0" prefWidth="200.0"
                wrapText="true" VBox.vgrow="ALWAYS" />
            <Label fx:id="maxCharacterLabel" text="Maximum 100 characters"
                textFill="#797979" />
         </children>
         <padding>
            <Insets bottom="10.0" left="10.0" right="10.0" top="10.0" />
         </padding>
      </VBox>
   </children>
   <padding>
      <Insets bottom="5.0" left="5.0" right="5.0" top="5.0" />
   </padding>
</GridPane>
```

icaddcircleoutline.png

icclose black.png

listViewTheme.css

```
.list-cell:even {
```

```
    -fx-background-color: white;
    -fx-text-fill: black;
    -fx-padding:0px 0px 0px 10px;
}

.list-cell:odd{
    -fx-background-color: white;
    -fx-text-fill: black;
    -fx-padding:0px 10px 0px 0px;
}

.list-cell:filled:selected:focused, .list-cell:filled:selected {
    -fx-focus-color: transparent;
}

.label{
    -fx-text-fill:black;
}
```

listViewThemeTimeline.css

```
.list-cell {
    -fx-background-color: white;
    -fx-text-fill: black;
}

.list-cell:filled:selected:focused, .list-cell:filled:selected {
    -fx-focus-color: transparent;
}

.label{
    -fx-text-fill:black;
}
```

rangeDataLayout.fxml

```
<?xml version="1.0" encoding="UTF-8"?>


<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.ListView?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.text.Font?>


<VBox fx:id="rootVBox" maxHeight="-Infinity" maxWidth="-Infinity"
    xmlns="http://javafx.com/javafx/8.0.111" xmlns:fx="http://javafx.com/fxml/1">
   <children>
      <Label fx:id="dateLabel" text="DATE">
         <font>
            <Font name="System Bold" size="19.0" />
         </font>
      </Label>
      <ListView fx:id="resultsListView" maxHeight="-Infinity" maxWidth="-Infinity"
          prefHeight="160.0" prefWidth="350.0" stylesheets="@listViewThemeTimeline.css"
          VBox.vgrow="ALWAYS">
         <VBox.margin>
            <Insets bottom="5.0" left="15.0" right="5.0" top="5.0" />
         </VBox.margin></ListView>
   </children>
   <padding>
      <Insets bottom="5.0" left="5.0" right="5.0" top="5.0" />
   </padding>
</VBox>
```

timelineRowEven.fxml

```
<?xml version="1.0" encoding="UTF-8"?>


<?import javafx.geometry.Insets?>
<?import javafx.scene.Group?>
<?import javafx.scene.control.Button?>
```

```
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.BorderPane?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.layout.Region?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.shape.Line?>

<Group fx:id="group" xmlns="http://javafx.com/javafx/8.0.111"
    xmlns:fx="http://javafx.com/fxml/1">
  <children>
    <HBox alignment="CENTER_LEFT" layoutX="10.0" layoutY="10.0" maxHeight="-Infinity"
        maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity">
      <children>
        <BorderPane fx:id="borderPane" maxHeight="-Infinity" maxWidth="-Infinity"
            minHeight="-Infinity" minWidth="-Infinity" prefWidth="370.0">
          <padding>
            <Insets bottom="15.0" left="15.0" right="15.0" top="15.0" />
          </padding>
          <bottom>
            <BorderPane>
              <left>
                <Button fx:id="editButton" mnemonicParsing="false" text="Edit Event"
                    BorderPane.alignment="CENTER" />
              </left>
              <right>
                <Button fx:id="viewButton" mnemonicParsing="false" text="Go to
                    Document" BorderPane.alignment="CENTER" />
              </right>
              <padding>
                <Insets bottom="5.0" top="5.0" />
              </padding>
            </BorderPane>
          </bottom>
          <left>
            <Group BorderPane.alignment="CENTER">
              <children>
```

```
                <VBox prefWidth="370.0">
                   <children>
                      <Label fx:id="eventNumberLabel" text="Event ##" />
                      <Label fx:id="dateLabel" maxHeight="-Infinity"
                           maxWidth="-Infinity" minHeight="-Infinity"
                           minWidth="-Infinity" prefWidth="340.0" text="Date:
                           dd-MM-yyy -&gt; dd-MM-yyyy" wrapText="true" />
                      <Label fx:id="subjectsLabel" maxHeight="-Infinity"
                           maxWidth="-Infinity" minHeight="-Infinity"
                           minWidth="-Infinity" prefWidth="340.0" text="Subjects:
                           Football, Gifts, London" wrapText="true" />
                      <Label fx:id="eventLabel" maxHeight="-Infinity"
                           maxWidth="-Infinity" minHeight="-Infinity"
                           minWidth="-Infinity" prefWidth="340.0" text="Event: played
                           football with friends" wrapText="true" />
                      <Label fx:id="fromLabel" maxHeight="-Infinity"
                           maxWidth="-Infinity" minHeight="-Infinity"
                           minWidth="-Infinity" prefWidth="340.0" text="From:
                           filename.ext" wrapText="true" />
                   </children>
                   <padding>
                      <Insets bottom="5.0" top="5.0" />
                   </padding>
                </VBox>
             </children>
          </Group>
       </left>
    </BorderPane>
    <Line startX="-100.0" strokeWidth="3.0" />
    <Line endY="250.0" nodeOrientation="LEFT_TO_RIGHT" strokeWidth="3.0" />
    <Region maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity"
        minWidth="-Infinity" prefHeight="200.0" prefWidth="483.0" />
</children>
<padding>
    <Insets right="10.0" />
</padding>
```

```
    </HBox>
  </children>
</Group>
```

timelineRowOdd.fxml

```
<?xml version="1.0" encoding="UTF-8"?>


<?import javafx.geometry.Insets?>
<?import javafx.scene.Group?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.BorderPane?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.layout.Region?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.shape.Line?>


<Group fx:id="group" xmlns="http://javafx.com/javafx/8.0.111"
    xmlns:fx="http://javafx.com/fxml/1">
   <children>
      <HBox alignment="CENTER_LEFT" layoutX="10.0" layoutY="10.0" maxHeight="-Infinity"
          maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity">
         <children>
         <Region maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity"
             minWidth="-Infinity" prefHeight="200.0" prefWidth="483.0" />
            <Line endY="250.0" nodeOrientation="LEFT_TO_RIGHT" strokeWidth="3.0" />
            <Line startX="-100.0" strokeWidth="3.0" />
         <BorderPane fx:id="borderPane" maxHeight="-Infinity" maxWidth="-Infinity"
             minHeight="-Infinity" minWidth="-Infinity" prefWidth="370.0">
           <padding>
             <Insets bottom="15.0" left="15.0" right="15.0" top="15.0" />
           </padding>
           <bottom>
              <BorderPane>
                 <left>
```

```
            <Button fx:id="editButton" mnemonicParsing="false" text="Edit Event"
                BorderPane.alignment="CENTER" />
         </left>
         <right>
            <Button fx:id="viewButton" mnemonicParsing="false" text="Go to
                Document" BorderPane.alignment="CENTER" />
         </right>
         <padding>
            <Insets bottom="5.0" top="5.0" />
         </padding>
      </BorderPane>
   </bottom>
   <left>
      <Group BorderPane.alignment="CENTER">
         <children>
            <VBox prefWidth="370.0">
               <children>
                  <Label fx:id="eventNumberLabel" text="Event ##" />
                  <Label fx:id="dateLabel" maxHeight="-Infinity"
                      maxWidth="-Infinity" minHeight="-Infinity"
                      minWidth="-Infinity" prefWidth="340.0" text="Date:
                      dd-MM-yyy -&gt; dd-MM-yyyy" wrapText="true" />
                  <Label fx:id="subjectsLabel" maxHeight="-Infinity"
                      maxWidth="-Infinity" minHeight="-Infinity"
                      minWidth="-Infinity" prefWidth="340.0" text="Subjects:
                      Football, Gifts, London" wrapText="true" />
                  <Label fx:id="eventLabel" maxHeight="-Infinity"
                      maxWidth="-Infinity" minHeight="-Infinity"
                      minWidth="-Infinity" prefWidth="340.0" text="Event: played
                      football with friends" wrapText="true" />
                   <Label fx:id="fromLabel" maxHeight="-Infinity"
                       maxWidth="-Infinity" minHeight="-Infinity"
                       minWidth="-Infinity" prefWidth="340.0" text="From:
                       filename.ext" wrapText="true" />
               </children>
               <padding>
```

```
                        <Insets bottom="5.0" top="5.0" />
                    </padding>
                </VBox>
            </children>
        </Group>
    </left>
</BorderPane>
</children>
<padding>
    <Insets right="10.0" />
</padding>
</HBox>
</children>
</Group>
```

C.6.2 dialogs

customErrorFields.css

```
.text-field:error {
  -fx-text-box-border: red ;
  -fx-focus-color: red ;
}

.text-area:error{
  -fx-text-box-border: red ;
  -fx-focus-color: red ;
}
```

loadingDialog.css

```
.please-wait-text {
    -fx-font-weight: bold;
    -fx-font-size: 18px;
}
```

```
.processing-text {
    -fx-font-size: 16px;
}

.loading-dialog {
    -fx-background-color: #ffffff;
    -fx-border-color: -fx-text-box-border;
    -fx-border-width: 2;
}
```

C.6.3 documentListViewTheme.css

```
.list-cell {
    -fx-background-color: white;
    -fx-text-fill: black;
}



.list-cell:filled:selected:focused, .list-cell:filled:selected {
    -fx-focus-color: transparent;

}

.label{
    -fx-text-fill:black;
}
```

C.6.4 listView.fxml

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.ListView?>
```

```
<?import javafx.scene.control.Menu?>
<?import javafx.scene.control.MenuBar?>
<?import javafx.scene.control.MenuItem?>
<?import javafx.scene.control.RadioMenuItem?>
<?import javafx.scene.control.ScrollPane?>
<?import javafx.scene.control.SeparatorMenuItem?>
<?import javafx.scene.layout.BorderPane?>
<?import javafx.scene.layout.ColumnConstraints?>
<?import javafx.scene.layout.GridPane?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.layout.RowConstraints?>
<?import javafx.scene.layout.StackPane?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.text.Font?>

<StackPane fx:id="stackPane" maxHeight="-Infinity" maxWidth="-Infinity"
    minHeight="-Infinity" minWidth="-Infinity" xmlns="http://javafx.com/javafx/8.0.111"
    xmlns:fx="http://javafx.com/fxml/1"
    fx:controller="frontend.controllers.ListViewController">
  <children>
    <VBox fx:id="vBox" maxHeight="1.7976931348623157E308"
        maxWidth="1.7976931348623157E308" minHeight="-Infinity" minWidth="-Infinity"
        prefHeight="800.0" prefWidth="1024.0" StackPane.alignment="CENTER">
      <children>
        <MenuBar>
          <menus>
            <Menu mnemonicParsing="false" text="File">
              <items>
                <MenuItem mnemonicParsing="false" onAction="#close" text="Close" />
                <MenuItem mnemonicParsing="false" onAction="#about" text="About" />
                <SeparatorMenuItem mnemonicParsing="false" />
                <MenuItem mnemonicParsing="false" onAction="#preferences"
                    text="Preferences..." />
              </items>
            </Menu>
            <Menu mnemonicParsing="false" text="Timeline">
```

```
            <items>
               <RadioMenuItem fx:id="dateView" mnemonicParsing="false" text="Date
                   View" />
               <RadioMenuItem fx:id="rangeView" mnemonicParsing="false" text="Range
                   View" />
            </items>
         </Menu>
      </menus>
   </MenuBar>
   <GridPane VBox.vgrow="ALWAYS">
      <columnConstraints>
         <ColumnConstraints hgrow="SOMETIMES" maxWidth="370.0" minWidth="10.0" />
         <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" />
         <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />
         <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />
         <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />
      </columnConstraints>
      <rowConstraints>
         <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
      </rowConstraints>
      <children>
         <BorderPane GridPane.rowSpan="2147483647">
            <top>
               <VBox prefWidth="100.0" BorderPane.alignment="CENTER">
                  <children>
                     <Label text="Documents Loaded:">
                        <padding>
                           <Insets bottom="5.0" right="5.0" top="5.0" />
                        </padding>
                     </Label>
                     <ListView fx:id="documentListView" maxHeight="-Infinity"
                         maxWidth="-Infinity" prefHeight="280.0" prefWidth="310.0"
                         stylesheets="@documentListViewTheme.css" />
                  </children>
                  <padding>
                     <Insets bottom="10.0" left="20.0" right="20.0" top="10.0" />
```

```
      </padding>
   </VBox>
</top>
<bottom>
   <HBox alignment="CENTER" prefHeight="100.0" prefWidth="200.0"
       BorderPane.alignment="CENTER">
      <children>
         <BorderPane prefHeight="200.0" prefWidth="200.0"
             HBox.hgrow="ALWAYS">
            <left>
               <Button fx:id="loadDocumentsButton" maxWidth="-Infinity"
                    minWidth="-Infinity" mnemonicParsing="false"
                    prefWidth="120.0" text="Load Documents"
                    textAlignment="CENTER" wrapText="true"
                    BorderPane.alignment="CENTER">
                  <font>
                     <Font size="18.0" />
                  </font>
               </Button>
            </left>
            <right>
               <Button fx:id="saveToButton" maxWidth="-Infinity"
                    minWidth="-Infinity" mnemonicParsing="false"
                    prefWidth="120.0" text="Save to... "
                    textAlignment="CENTER" wrapText="true"
                    BorderPane.alignment="CENTER">
                  <font>
                     <Font size="18.0" />
                  </font>
               </Button>
            </right>
            <padding>
               <Insets bottom="10.0" left="20.0" right="20.0" top="10.0"
                    />
            </padding>
         </BorderPane>
```

```
                    </children>
                </HBox>
            </bottom>
        </BorderPane>
        <ListView fx:id="timelineListView" GridPane.columnIndex="1"
            GridPane.columnSpan="2147483647" GridPane.halignment="RIGHT"
            GridPane.hgrow="ALWAYS" GridPane.rowSpan="2147483647">
            <padding>
                <Insets bottom="10.0" left="10.0" right="10.0" top="10.0" />
            </padding>
        </ListView>
        <ScrollPane fx:id="scrollPane" prefHeight="200.0" prefWidth="200.0"
            GridPane.columnIndex="1" GridPane.columnSpan="2147483647"
            GridPane.rowSpan="2147483647" GridPane.valignment="TOP" />
      </children>
    </GridPane>
  </children>
 </VBox>
</children>
</StackPane>
```

C.6.5 startup.fxml

```
<?xml version="1.0" encoding="UTF-8"?>


<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Menu?>
<?import javafx.scene.control.MenuBar?>
<?import javafx.scene.control.MenuItem?>
<?import javafx.scene.control.SeparatorMenuItem?>
<?import javafx.scene.layout.BorderPane?>
<?import javafx.scene.layout.StackPane?>
<?import javafx.scene.layout.VBox?>
```

```
<StackPane fx:id="stackPane" maxHeight="-Infinity" maxWidth="-Infinity"
    minHeight="-Infinity" minWidth="-Infinity" prefHeight="400.0" prefWidth="600.0"
    xmlns="http://javafx.com/javafx/8.0.111" xmlns:fx="http://javafx.com/fxml/1"
    fx:controller="frontend.controllers.StartUpController">
  <children>
    <VBox fx:id="vBox" maxHeight="1.7976931348623157E308"
        maxWidth="1.7976931348623157E308" StackPane.alignment="CENTER">
      <children>
        <MenuBar>
          <menus>
            <Menu mnemonicParsing="false" text="File">
              <items>
                <MenuItem mnemonicParsing="false" onAction="#close" text="Close" />
                <MenuItem mnemonicParsing="false" onAction="#about" text="About" />
                <SeparatorMenuItem mnemonicParsing="false" />
                <MenuItem mnemonicParsing="false" onAction="#preferences"
                    text="Preferences..." />
              </items>
            </Menu>
            <Menu disable="true" mnemonicParsing="false" text="Timeline">
              <items>
                <MenuItem mnemonicParsing="false" onAction="#timeline" text="Delete"
                    />
              </items>
            </Menu>
          </menus>
        </MenuBar>
        <BorderPane VBox.vgrow="ALWAYS">
          <center>
            <Button fx:id="loadDocumentsButton" mnemonicParsing="false"
                onAction="#loadDocuments" prefHeight="111.0" prefWidth="283.0"
                text="Get started by loading in documents!"
                BorderPane.alignment="CENTER" />
          </center>
          <padding>
            <Insets bottom="10.0" left="10.0" right="10.0" top="10.0" />
```

```
            </padding>
            <VBox.margin>
               <Insets />
            </VBox.margin>
         </BorderPane>
      </children>
   </VBox>
</children>
</StackPane>
```

C.7 src/test/backend

EngineTest.java

```
package backend;

import backend.process.Engine;
import backend.process.Result;
import backend.process.TimelineDate;
import org.junit.Assert;
import org.junit.Test;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;

/**
 * Test class for the backend.process.Engine. Gives in a sample text, and compares that
     the generated and predicted backend.process.Result objects are equal
 */
public class EngineTest {
    private final static SimpleDateFormat simpleDateFormat = new
        SimpleDateFormat("yyyy-MM-dd");

    /**
```

```
     * Gives backend.process.Engine sample text and compares the expected and generated
         output.
     * Only compares the output by the date (checking that it is picking out the right
         sentence, not looking at sentence trimming).
     *
     * @throws ParseException thrown when we create the Dates for the expected
         TimelineDates and Results.
     */
    @Test
    public void testDates() throws ParseException {
        String sampleText = "On the 12th of December I ran tests on my final year project.
            The tests did not go so well. " +
               "Yesterday I played games. " + "It was fun playing games! Tomorrow I am
                   going to study. Last week I went to watch a football match.";
        ArrayList<Result> expectedResults = new ArrayList<>();

        Result result1 = new Result();
        TimelineDate timelineDate1 = new TimelineDate();
        timelineDate1.setDate1(simpleDateFormat.parse("2016-12-12"));
        result1.setTimelineDate(timelineDate1);
        expectedResults.add(result1);

        Result result2 = new Result();
        TimelineDate timelineDate2 = new TimelineDate();
        timelineDate2.setDate1(simpleDateFormat.parse("2016-12-22"));
        result2.setTimelineDate(timelineDate2);
        expectedResults.add(result2);

        Result result3 = new Result();
        TimelineDate timelineDate3 = new TimelineDate();
        timelineDate3.setDate1(simpleDateFormat.parse("2016-12-24"));
        result3.setTimelineDate(timelineDate3);
        expectedResults.add(result3);

        Result result4 = new Result();
        TimelineDate timelineDate4 = new TimelineDate();
```

```
        timelineDate4.setDate1(simpleDateFormat.parse("2016-12-12"));
        timelineDate4.setDate2(simpleDateFormat.parse("2016-12-18"));
        result4.setTimelineDate(timelineDate4);
        expectedResults.add(result4);

        Engine engine = new Engine();
        ArrayList<Result> results = engine.getResults(sampleText, "2016-12-23");
        compareExpectedToActualDate(results, expectedResults);
    }

    /**
     * Tests the Engines ability to pick out Subjects from a given Sample text, and checks
         it to the expected Subjects
     * for that text.
     */
    @Test
    public void testSubjects() {
        //picks up last 16 as a date (issue with casual language used)
        String sampleText = "Manchester City face Monaco in the Champions League, with the
            first leg on 21 February. In their last season, Liverpool narrowly missed out
            on the 2013/14 Premier League title to Manchester City, while Leicester won
            last year's title only playing domestically. ";
        ArrayList<Result> expectedResults = new ArrayList<>();//just looking at subjects,
            not dates
        Result result1 = new Result();//two sentences, both with dates, so two results made
        Result result2 = new Result();//should pick out the team names: Monaco, Liverpool,
            Manchester City, Leicester, and Competitions: Champions League, Premier League
        //for the first sentence, the teams mentioned
        result1.addSubject("Champions League");
        result1.addSubject("Monaco");
        result1.addSubject("Manchester City");

        result2.addSubject("Liverpool");
        result2.addSubject("Manchester City");
        result2.addSubject("Premier League");
        result2.addSubject("Leicester");
```

```
    expectedResults.add(result1);
    expectedResults.add(result2);

    Engine engine = new Engine();
    ArrayList<Result> actualResults = engine.getResults(sampleText, "26-12-2016");

    compareExpectedToActualSubject(actualResults, expectedResults);
}



/**
 * Compares an actual list of Results with a expected one, by just looking at the dates
     picked out. Checking that the backend.process.Engine
 * is picking out the right sentence and producing the right dates, not caring about
     the sentence trimming or subject picking.
 *
 * @param actualResults the list of produced backend.process.Result objects
 * @param expectedResults the list of expected backend.process.Result objects
 */
private void compareExpectedToActualDate(ArrayList<Result> actualResults,
     ArrayList<Result> expectedResults) {
    System.out.println("Actual Results: " + actualResults);
    Assert.assertEquals(expectedResults.size(), actualResults.size());//if this does
         not hold then the test fails
    for (int i = 0; i < actualResults.size(); i++) {
        Assert.assertEquals(expectedResults.get(i).getTimelineDate(),
            actualResults.get(i).getTimelineDate());
    }
}

/**
 * Compares the Subjects (only) in the list of Expected backend.process.Result objects
     to the list of Actual backend.process.Result objects.
 * Compares each actual backend.process.Result object individually to its expected
     counterpart.
```

```
     *
     * @param actualResults the list of produced backend.process.Result objects
     * @param expectedResults the list of expected backend.process.Result objects
     */
    private void compareExpectedToActualSubject(ArrayList<Result> actualResults,
         ArrayList<Result> expectedResults) {
        System.out.println("Actual Results: " + actualResults);
        System.out.println("Expected Results: " + expectedResults);
        Assert.assertEquals(expectedResults.size(), actualResults.size());
        for (int i = 0; i < actualResults.size(); i++) {
            Assert.assertEquals(expectedResults.get(i).getSubjects(),
                 actualResults.get(i).getSubjects());
        }
    }
}
```

ProcessFileTest.java

```
package backend;

import backend.process.*;
import javafx.concurrent.Task;
import org.junit.Assert;
import org.junit.Test;

import java.io.File;
import java.net.URISyntaxException;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.List;

/**
 * Tests the the extracting of text in files and passing it to the backend.process.Engine,
     and the fact that Threads are being created
 * to process the Files separately.
```

```
 */

public class ProcessFileTest {
    private SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd");
    private List<Result> actualResults;

    /**
     * Chaining tests (all in same class running one after the other) for ProcessFile will
         overlap, as when the next
     * test runs, the other has not finished.
     * <p>
     * Test for processing a test file with sample text, and comparing its output to
         expected backend.process.Result objects.
     *
     * @throws ParseException    for the Dates that we create for the expected Results.
     * @throws InterruptedException as we are putting the Thread that runs this test to
         sleep to let ProcessFile finish running.
     */
    @Test
    public void testSampleFileProcessTXT() throws ParseException, InterruptedException,
        URISyntaxException {
        List<Result> expectedResults = new ArrayList<>();
        actualResults = null;

        Result result1 = new Result();
        TimelineDate timelineDate1 = new TimelineDate();
        timelineDate1.setDate1(simpleDateFormat.parse("2016-12-12"));
        result1.setTimelineDate(timelineDate1);
        expectedResults.add(result1);

        Result result2 = new Result();
        TimelineDate timelineDate2 = new TimelineDate();
        timelineDate2.setDate1(simpleDateFormat.parse("2017-01-27"));
        result2.setTimelineDate(timelineDate2);
        expectedResults.add(result2);
```

```
        Result result3 = new Result();
        TimelineDate timelineDate3 = new TimelineDate();
        timelineDate3.setDate1(simpleDateFormat.parse("2017-01-29"));
        result3.setTimelineDate(timelineDate3);
        expectedResults.add(result3);

        Result result4 = new Result();
        TimelineDate timelineDate4 = new TimelineDate();
        timelineDate4.setDate1(simpleDateFormat.parse("2017-01-16"));
        timelineDate4.setDate2(simpleDateFormat.parse("2017-01-22"));
        result4.setTimelineDate(timelineDate4);
        expectedResults.add(result4);

        File testFile = new File(getClass().getResource("testfile1.txt").toURI());
        ArrayList<File> files = new ArrayList<>();
        files.add(testFile);

        ArrayList<FileData> fileDatas = new ArrayList<>();
        FileData fileData = new FileData("testfile1.txt",
            getClass().getResource("testfile1.txt").toString());
        fileData.setCreationDate("28-01-2017");
        fileDatas.add(fileData);

        ProcessFiles processFiles = new ProcessFiles();
        actualResults = processFiles.processFiles(files, fileDatas);
        compareExpectedToActual(actualResults, expectedResults);
    }

    /**
     * Tests that more than one Thread is running after calling ProcessFile (as a minimum
         it will create one extra Thread
     * to process the passed in files).
     *
     * @throws InterruptedException as we put the Thread that runs this test to sleep, so
         that ProcessFile can finish running.
     */
```

```
@Test
public void testMultiThread() throws InterruptedException, URISyntaxException {//check
     that processfile makes two more threads run
    // at least 2 Threads should be running (main and the one created by process file).
    actualResults = null;
    File testFile1 = new File(getClass().getResource("testfile1.txt").toURI());
    File testFile2 = new File(getClass().getResource("testfile2.txt").toURI());
    File testFile3 = new File(getClass().getResource("testfile3.txt").toURI());

    ArrayList<File> files = new ArrayList<>();
    files.add(testFile1);
    files.add(testFile2);
    files.add(testFile3);

    ArrayList<FileData> fileDatas = new ArrayList<>();
    FileData fileData = new FileData("testfile1.txt",
        getClass().getResource("testfile1.txt").toString());
    fileData.setCreationDate("28-01-2017");
    fileDatas.add(fileData);
    fileData = new FileData("testfile2.txt",
        getClass().getResource("testfile2.txt").toString());
    fileData.setCreationDate("28-01-2017");
    fileDatas.add(fileData);
    fileData = new FileData("testfile3.txt",
        getClass().getResource("testfile3.txt").toString());
    fileData.setCreationDate("28-01-2017");
    fileDatas.add(fileData);

    ProcessFiles processFiles = new ProcessFiles();
    Task<Boolean> task = new Task<Boolean>() {
        @Override
        protected Boolean call() throws Exception {
            processFiles.processFiles(files, fileDatas);
            return null;
        }
    };
```

```
        new Thread(task).start();
        System.out.println("Actual Threads running" + Thread.activeCount());
        Assert.assertEquals(true, Thread.activeCount() >= 2);
        Thread.sleep(10000);//give the backend.process.Engine time to finish running
        Assert.assertEquals(true, true);//seems to be an error that if you finish with a
            thread sleep it will stop running the other operations on other threads (i.e.
            releasing semaphores and setting state to FINISHED).
    }

    /**
     * Tests the processing of a sample docx file, by checking the processed Result against
         expected Results. Only looks
     * at dates picked out (checking only it is processing the File correctly, i.e. picking
         out the correct sentences, and
     * their dates).
     *
     * @throws InterruptedException as we are putting the Thread that runs this test to
         sleep to let ProcessFile finish running.
     * @throws ParseException    for the Dates that we create for the expected Results.
     */
    @Test
    public void testSampleFileProcessDocx() throws InterruptedException, ParseException,
         URISyntaxException {
        actualResults = null;
        ArrayList<Result> expectedResults = new ArrayList<>();
        //base date is 29/12/2016 (creation date of file)
        Result result1 = new Result();
        TimelineDate timelineDate1 = new TimelineDate();
        timelineDate1.setDate1(simpleDateFormat.parse("2017-01-27"));
        result1.setTimelineDate(timelineDate1);
        expectedResults.add(result1);

        Result result2 = new Result();
        TimelineDate timelineDate2 = new TimelineDate();
        timelineDate2.setDate1(simpleDateFormat.parse("2008-01-01"));//second sentence
            talks about donations between 2008 and spring of this year (2016)
```

```
        timelineDate2.setDate2(simpleDateFormat.parse("2017-05-31"));//last day of spring
        result2.setTimelineDate(timelineDate2);
        expectedResults.add(result2);

        Result result3 = new Result();
        TimelineDate timelineDate3 = new TimelineDate();
        timelineDate3.setDate1(simpleDateFormat.parse("1980-01-01"));//donated since the
            early 1980s (range so 1980 ->1989)
        timelineDate3.setDate2(simpleDateFormat.parse("1989-12-31"));//last day of the 1980s
        result3.setTimelineDate(timelineDate3);
        expectedResults.add(result3);

        Result result4 = new Result();
        TimelineDate timelineDate4 = new TimelineDate();
        timelineDate4.setDate1(simpleDateFormat.parse("1996-01-01"));//in 1996 an event
            happened (doesnt specify day or month, just year)
        result4.setTimelineDate(timelineDate4);
        expectedResults.add(result4);

        File testFile = new File(getClass().getResource("testfile4.docx").toURI());
        ArrayList<File> files = new ArrayList<>();
        files.add(testFile);

        ArrayList<FileData> fileDatas = new ArrayList<>();
        FileData fileData = new FileData("testfile4.docx",
            getClass().getResource("testfile4.docx").toString());
        fileData.setCreationDate("28-01-2017");
        fileDatas.add(fileData);

        ProcessFiles processFiles = new ProcessFiles();
        actualResults = processFiles.processFiles(files, fileDatas);
        compareExpectedToActual(actualResults, expectedResults);
    }

    /**
```

```
     * Test for processing PDF Files, by checking the processed Results to the expected
         Results. Checks only their dates
     * as we are interested in it picking out the right sentences and their dates in the
         documents parsed in.
     *
     * @throws InterruptedException as we are putting the Thread that runs this test to
         sleep to allow time for ProcessFile to finish (runs on separate Thread).
     * @throws ParseException    for the Dates that we create for the expected Results.
     */
    @Test
    public void testSampleFilePDF() throws InterruptedException, ParseException,
         URISyntaxException {
        actualResults = null;
        ArrayList<Result> expectedResults = new ArrayList<>();
        //base date 2016-12-29

        //on Friday = 2016-12-30
        //1967 = 1967-01-01
        Result result1 = new Result();
        TimelineDate timelineDate1 = new TimelineDate();
        timelineDate1.setDate1(simpleDateFormat.parse("2017-01-27"));
        result1.setTimelineDate(timelineDate1);
        expectedResults.add(result1);

        Result result2 = new Result();
        TimelineDate timelineDate2 = new TimelineDate();
        timelineDate2.setDate1(simpleDateFormat.parse("1967-01-01"));
        result2.setTimelineDate(timelineDate2);
        expectedResults.add(result2);

        File testFile = new File(getClass().getResource("testfile5.pdf").toURI());//bbc
            article (http://www.bbc.com/news/world-us-canada-38451258)
        ArrayList<File> files = new ArrayList<>();
        files.add(testFile);

        ArrayList<FileData> fileDatas = new ArrayList<>();
```

```
        FileData fileData = new FileData("testfile5.pdf",
             getClass().getResource("testfile5.pdf").toString());
        fileData.setCreationDate("28-01-2017");
        fileDatas.add(fileData);

        ProcessFiles processFiles = new ProcessFiles();
        actualResults = processFiles.processFiles(files, fileDatas);
        System.out.println(actualResults);
        compareExpectedToActual(actualResults, expectedResults);
    }

    /**
     * Compares an actual list of Results with a expected one, by just looking at the dates
         picked out. Checking that the backend.process.Engine
     * is picking out the right sentence and producing the right dates, not caring about
         the sentence trimming or subject picking.
     *
     * @param actualResults the list of produced backend.process.Result objects
     * @param expectedResults the list of expected backend.process.Result objects
     */
    private void compareExpectedToActual(List<Result> actualResults, List<Result>
         expectedResults) {
        System.out.println("Actual Results: " + actualResults);
        Assert.assertEquals(actualResults.size(), expectedResults.size());//if this does
             not hold then the test fails
        for (int i = 0; i < actualResults.size(); i++) {
            Assert.assertEquals(actualResults.get(i).getTimelineDate(),
                 expectedResults.get(i).getTimelineDate());
        }
    }
}
```

ProduceRangesTest.java

```
package backend;
```

```
import backend.process.Result;
import backend.process.TimelineDate;
import backend.ranges.ProduceRanges;
import backend.ranges.Range;
import org.junit.Assert;
import org.junit.Test;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.List;

/**
 * Test class for the processing of Result Lists into Range Trees.
 */
public class ProduceRangesTest {

    /**
     * Tests a simple Range Tree formed by processing the fake List of Results, by matching
          it to an expected Range Tree.
     *
     * @throws ParseException when setting the Dates for the fake Results.
     */
    @Test
    public void testSingleTree() throws ParseException {
        List<Result> resultList = new ArrayList<>();

        SimpleDateFormat simpleDateFormat = new SimpleDateFormat("dd-MM-yyyy G");

        List<Range> expectedTrees = new ArrayList<>();
        Range range = new Range(simpleDateFormat.parse("01-01-0001 AD"),
             simpleDateFormat.parse("31-12-9999 AD"));
        Range range1980 = new Range(simpleDateFormat.parse("01-01-1980 AD"),
             simpleDateFormat.parse("31-12-1989 AD"));
        Range range2008 = new Range(simpleDateFormat.parse("01-01-2008 AD"),
             simpleDateFormat.parse("31-05-2016 AD"));
```

```
Range range2015 = new Range(simpleDateFormat.parse("01-01-2015 AD"), null);
range2008.addChild(range2015);
range.addChild(range1980);
range.addChild(range2008);
expectedTrees.add(range);

Result result1;
TimelineDate timelineDate1;
result1 = new Result();
timelineDate1 = new TimelineDate();
timelineDate1.setDate1(simpleDateFormat.parse("01-01-0001 AD"));
timelineDate1.setDate2(simpleDateFormat.parse("31-12-9999 AD"));
timelineDate1.getRange();
result1.setTimelineDate(timelineDate1);
resultList.add(result1);
range.add(result1);
result1 = new Result();
timelineDate1 = new TimelineDate();
timelineDate1.setDate1(simpleDateFormat.parse("01-01-1980 AD"));
timelineDate1.setDate2(simpleDateFormat.parse("31-12-1989 AD"));
timelineDate1.getRange();
result1.setTimelineDate(timelineDate1);
resultList.add(result1);
range1980.add(result1);
result1 = new Result();
timelineDate1 = new TimelineDate();
timelineDate1.setDate1(simpleDateFormat.parse("01-01-2008 AD"));
timelineDate1.setDate2(simpleDateFormat.parse("31-05-2016 AD"));
timelineDate1.getRange();
result1.setTimelineDate(timelineDate1);
resultList.add(result1);
range2008.add(result1);
result1 = new Result();
timelineDate1 = new TimelineDate();
timelineDate1.setDate1(simpleDateFormat.parse("01-01-2015 AD"));
timelineDate1.getRange();
```

```
        result1.setTimelineDate(timelineDate1);
        resultList.add(result1);
        range2015.add(result1);

        ProduceRanges produceRanges = new ProduceRanges();
        produceRanges.produceRanges(resultList);
        List<Range> actualTrees = produceRanges.getTrees();

        checkTrees(expectedTrees, actualTrees);

    }

    /**
     * Tests that multiple Trees are formed when the List of fake Results contains at least
         2 disjoint Results (based on
     * their Dates), and then checks that the Trees formed match the expected List of Trees.
     *
     * @throws ParseException when setting the Dates for the fake Results.
     */
    @Test
    public void testSeparateTrees() throws ParseException {
        List<Result> resultList = new ArrayList<>();

        SimpleDateFormat simpleDateFormat = new SimpleDateFormat("dd-MM-yyyy G");

        List<Range> expectedTrees = new ArrayList<>();
        Range range = new Range(simpleDateFormat.parse("01-01-0001 AD"),
            simpleDateFormat.parse("31-12-9999 AD"));
        Range range1980 = new Range(simpleDateFormat.parse("01-01-1980 AD"),
            simpleDateFormat.parse("31-12-1989 AD"));
        Range range2008 = new Range(simpleDateFormat.parse("01-01-2008 AD"),
            simpleDateFormat.parse("31-05-2016 AD"));
        Range range2015 = new Range(simpleDateFormat.parse("01-01-2015 AD"), null);
        range2008.addChild(range2015);
        range.addChild(range1980);
        range.addChild(range2008);
```

```
Range range400BC = new Range(simpleDateFormat.parse("01-01-499 BC"),
    simpleDateFormat.parse("31-12-400 BC"));
expectedTrees.add(range400BC);
expectedTrees.add(range);

Result result1;
TimelineDate timelineDate1;
result1 = new Result();
timelineDate1 = new TimelineDate();
timelineDate1.setDate1(simpleDateFormat.parse("01-01-0001 AD"));
timelineDate1.setDate2(simpleDateFormat.parse("31-12-9999 AD"));
timelineDate1.getRange();
result1.setTimelineDate(timelineDate1);
resultList.add(result1);
range.add(result1);
result1 = new Result();
timelineDate1 = new TimelineDate();
timelineDate1.setDate1(simpleDateFormat.parse("01-01-1980 AD"));
timelineDate1.setDate2(simpleDateFormat.parse("31-12-1989 AD"));
timelineDate1.getRange();
result1.setTimelineDate(timelineDate1);
resultList.add(result1);
range1980.add(result1);
result1 = new Result();
timelineDate1 = new TimelineDate();
timelineDate1.setDate1(simpleDateFormat.parse("01-01-2008 AD"));
timelineDate1.setDate2(simpleDateFormat.parse("31-05-2016 AD"));
timelineDate1.getRange();
result1.setTimelineDate(timelineDate1);
resultList.add(result1);
range2008.add(result1);
result1 = new Result();
timelineDate1 = new TimelineDate();
timelineDate1.setDate1(simpleDateFormat.parse("01-01-2015 AD"));
timelineDate1.getRange();
result1.setTimelineDate(timelineDate1);
```

```
        resultList.add(result1);
        range2015.add(result1);
        result1 = new Result();
        timelineDate1 = new TimelineDate();
        timelineDate1.setDate1(simpleDateFormat.parse("01-01-0499 BC"));
        timelineDate1.setDate2(simpleDateFormat.parse("31-12-0400 BC"));
        timelineDate1.getRange();
        result1.setTimelineDate(timelineDate1);
        resultList.add(result1);
        range400BC.add(result1);


        ProduceRanges produceRanges = new ProduceRanges();
        produceRanges.produceRanges(resultList);
        List<Range> actualTrees = produceRanges.getTrees();

        checkTrees(expectedTrees, actualTrees);
    }

    /**
     * Tests the two Trees formed based on a list of fake Results. One of the Trees has a
         depth of 2, such that it contains
     * a Range with a sub-Range with a sub-sub-Range (more complex than previous tests).
         Checks that the Results produce
     * the expected two Range Trees (one in BC and one in AD), with the AD Range Tree
         containing 10 results.
     *
     * @throws ParseException when producing the Dates for the fake Results.
     */
    @Test
    public void testComplexSeparateTrees() throws ParseException {
        SimpleDateFormat simpleDateFormat = new SimpleDateFormat("dd-MM-yyyy G");
        List<Range> expectedTrees = new ArrayList<>();

        Range range01To99 = new Range(simpleDateFormat.parse("01-01-0001 AD"),
            simpleDateFormat.parse("31-12-9999 AD"));
```

```
Range range1980 = new Range(simpleDateFormat.parse("01-01-1980 AD"),
    simpleDateFormat.parse("31-12-1989 AD"));
Range range2008 = new Range(simpleDateFormat.parse("01-01-2008 AD"),
    simpleDateFormat.parse("31-05-2016 AD"));
Range range2015 = new Range(simpleDateFormat.parse("01-01-2015 AD"), null);
Range range201612To20 = new Range(simpleDateFormat.parse("12-12-2016 AD"),
    simpleDateFormat.parse("20-12-2016 AD"));
Range range201612To18 = new Range(simpleDateFormat.parse("12-12-2016 AD"),
    simpleDateFormat.parse("18-12-2016 AD"));
Range range201615To20 = new Range(simpleDateFormat.parse("15-12-2016 AD"),
    simpleDateFormat.parse("20-12-2016 AD"));
Range range201601 = new Range(simpleDateFormat.parse("01-11-2016 AD"), null);
Range range1996 = new Range(simpleDateFormat.parse("01-01-1996 AD"), null);
Range range201623 = new Range(simpleDateFormat.parse("23-12-2016 AD"), null);
Range range0499 = new Range(simpleDateFormat.parse("01-01-0499 BC"),
    simpleDateFormat.parse("31-12-0400 BC"));



range01To99.getChildren().add(range1980);
range01To99.getChildren().add(range1996);
range01To99.getChildren().add(range2008);
range01To99.getChildren().add(range201601);
range01To99.getChildren().add(range201612To20);
range01To99.getChildren().add(range201623);
range2008.getChildren().add(range2015);
range201612To20.getChildren().add(range201612To18);
range201612To20.getChildren().add(range201615To20);


expectedTrees.add(range0499);
expectedTrees.add(range01To99);


List<Result> results = new ArrayList<>();
Result result;
TimelineDate timelineDate;
result = new Result();
timelineDate = new TimelineDate();
```

```
        timelineDate.setDate1(simpleDateFormat.parse("01-01-0001 AD"));
        timelineDate.setDate2(simpleDateFormat.parse("31-12-9999 AD"));
        timelineDate.getRange();
        result.setTimelineDate(timelineDate);
        range01To99.add(result);
        range01To99.add(result);
        results.add(result);
        results.add(result);
        result = new Result();
        timelineDate = new TimelineDate();
        timelineDate.setDate1(simpleDateFormat.parse("01-01-1980 AD"));
        timelineDate.setDate2(simpleDateFormat.parse("31-12-1989 AD"));
        timelineDate.getRange();
        result.setTimelineDate(timelineDate);
        range1980.add(result);
        results.add(result);
        result = new Result();
        timelineDate = new TimelineDate();
        timelineDate.setDate1(simpleDateFormat.parse("01-01-2008 AD"));
        timelineDate.setDate2(simpleDateFormat.parse("31-05-2016 AD"));
        timelineDate.getRange();
        result.setTimelineDate(timelineDate);
        range2008.add(result);
        results.add(result);
        result = new Result();
        timelineDate = new TimelineDate();
        timelineDate.setDate1(simpleDateFormat.parse("01-01-2015 AD"));
        timelineDate.getRange();
        result.setTimelineDate(timelineDate);
        range2015.add(result);
        results.add(result);
        result = new Result();
        timelineDate = new TimelineDate();
        timelineDate.setDate1(simpleDateFormat.parse("12-12-2016 AD"));
        timelineDate.setDate2(simpleDateFormat.parse("18-12-2016 AD"));
        timelineDate.getRange();
```

```
        result.setTimelineDate(timelineDate);
        range201612To18.add(result);
        results.add(result);
        result = new Result();
        timelineDate = new TimelineDate();
        timelineDate.setDate1(simpleDateFormat.parse("15-12-2016 AD"));
        timelineDate.setDate2(simpleDateFormat.parse("20-12-2016 AD"));
        timelineDate.getRange();
        result.setTimelineDate(timelineDate);
        range201615To20.add(result);
        results.add(result);
        result = new Result();
        timelineDate = new TimelineDate();
        timelineDate.setDate1(simpleDateFormat.parse("01-11-2016 AD"));
        timelineDate.getRange();
        result.setTimelineDate(timelineDate);
        range201601.add(result);
        results.add(result);
        result = new Result();
        timelineDate = new TimelineDate();
        timelineDate.setDate1(simpleDateFormat.parse("01-01-1996 AD"));
        timelineDate.getRange();
        result.setTimelineDate(timelineDate);
        range1996.add(result);
        results.add(result);
        result = new Result();
        timelineDate = new TimelineDate();
        timelineDate.setDate1(simpleDateFormat.parse("23-12-2016 AD"));
        timelineDate.getRange();
        result.setTimelineDate(timelineDate);
        range201623.add(result);
        results.add(result);
        result = new Result();
        timelineDate = new TimelineDate();
        timelineDate.setDate1(simpleDateFormat.parse("01-01-0499 BC"));
        timelineDate.setDate2(simpleDateFormat.parse("31-12-0400 BC"));
```

```
        timelineDate.getRange();
        result.setTimelineDate(timelineDate);
        range0499.add(result);
        results.add(result);

        ProduceRanges produceRanges = new ProduceRanges();
        produceRanges.produceRanges(results);

        List<Range> actual = produceRanges.getTrees();
        checkTrees(expectedTrees, actual);
    }

    /**
     * Asserts that both lists of Ranges, the expected and actual, are equal, ie they are
         the same size, and that each
     * contain the same Range objects at the same indices.
     *
     * @param expected the expected list of Ranges.
     * @param actual  the actual list of Ranges.
     */
    private void checkTrees(List<Range> expected, List<Range> actual) {
        Assert.assertEquals(expected.size(), actual.size());
        for (int i = 0; i < expected.size(); i++) {
            Assert.assertEquals(expected.get(i), actual.get(i));
        }
    }
}
```

SystemStateTest.java

```
package backend;

import backend.process.FileData;
import backend.process.ProcessFiles;
import backend.system.BackEndSystem;
import backend.system.SystemState;
```

```java
import org.junit.Assert;
import org.junit.Test;

import java.io.File;
import java.net.URISyntaxException;
import java.util.ArrayList;

/**
 * Test class to test the change of System States as files are being Processed. Checks for
     STARTED, PROCESSING, and PROCESSED
 * System States, as these are the three main different states.
 */
public class SystemStateTest {

    /**
     * Checks the initial state, then when processing, and then when it should be finished.
     *
     * @throws InterruptedException as we make the Test thread wait for ProcessFile to
         finish processing the Files.
     */
    @Test
    public void testSystemStatePROCESSING() throws InterruptedException,
         URISyntaxException {
        BackEndSystem.getInstance().setSystemState(SystemState.STARTED);
        Assert.assertEquals(SystemState.STARTED,
             BackEndSystem.getInstance().getSystemState());

        ProcessFiles processFiles = new ProcessFiles();
        File testFile = new File(getClass().getResource("testfile1.txt").toURI());
        ArrayList<File> files = new ArrayList<>();
        files.add(testFile);

        ArrayList<FileData> fileDatas = new ArrayList<>();
        for(File file: files){
            fileDatas.add(new FileData(file));
        }
```

```
    Runnable newRunnable = new Runnable() {
        @Override
        public void run() {
            processFiles.processFiles(files, fileDatas);
        }
    };

    Thread thread = new Thread(newRunnable);
    thread.start();
    Thread.sleep(100);
    System.out.println("System state: " + BackEndSystem.getInstance().getSystemState());
    Assert.assertEquals(SystemState.PROCESSING,
         BackEndSystem.getInstance().getSystemState());

    Thread.sleep(5000);

}



/**
 * Checks that the state is set to FINISHED when files have been processed, and the
     result has been returned.
 */
@Test
public void testSystemStateFINISHED() throws URISyntaxException {
    BackEndSystem.getInstance().setSystemState(SystemState.STARTED);
    Assert.assertEquals(SystemState.STARTED,
         BackEndSystem.getInstance().getSystemState());

    ProcessFiles processFiles = new ProcessFiles();
    File testFile = new File(getClass().getResource("testfile1.txt").toURI());
    ArrayList<File> files = new ArrayList<>();
    files.add(testFile);

    ArrayList<FileData> fileDatas = new ArrayList<>();
```

```
        for(File file: files){
            fileDatas.add(new FileData(file));
        }

        processFiles.processFiles(files, fileDatas);

        Assert.assertEquals(SystemState.FINISHED,
             BackEndSystem.getInstance().getSystemState());

    }
}
```

TimelineDateTest.java

```
package backend;

import backend.process.TimelineDate;
import edu.stanford.nlp.time.SUTime;
import org.junit.Assert;
import org.junit.Test;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

/**
 * Test for the parsing of Normalized Entity Tag in backend.process.TimelineDate, and how
     they are processed.
 */
public class TimelineDateTest {
    private static final String PRESENT_REF = "PRESENT_REF";
    private static final String PAST_REF = "PAST_REF";
    private static final String FUTURE_REF = "FUTURE_REF";
    private SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd");
    private String baseDate = "2016-12-30";//doesn't affect the output of most tests, but
         has to be still passed in.
```

```
/**
 * Passes in a simple date of the format yyyy-MM-dd into backend.process.TimelineDate,
     and checks the output date.
 *
 * @throws ParseException when comparing the expected Date to check.
 */
@Test
public void testTimelineDateProcessSimple() throws ParseException {
    String dateStr = "2016-12-24";
    Date expectedDate = simpleDateFormat.parse(dateStr);

    TimelineDate timelineDate = new TimelineDate();
    timelineDate.parse(dateStr, baseDate);//should make a Date object of the same type,
        not much processing.

    Assert.assertEquals(expectedDate, timelineDate.getDate1());
    Assert.assertEquals(null, timelineDate.getDate2());//should have only picked up one
        date
}

/**
 * Passes in a normalized entity tag for a year range, 198X, and checks the output to
     the expected start and end Date
 * of the year range.
 *
 * @throws ParseException when creating the expected Dates to check.
 */
@Test
public void testTimelineDateProcessYearRange() throws ParseException {
    String yearRange = "198X";
    String expectedStartDate = "1980-01-01";
    String expectedEndDate = "1989-12-31";

    Date expectedStart = simpleDateFormat.parse(expectedStartDate);
    Date expectedEnd = simpleDateFormat.parse(expectedEndDate);
```

```
    TimelineDate timelineDate = new TimelineDate();
    timelineDate.parse(yearRange, baseDate);

    Assert.assertEquals(expectedStart, timelineDate.getDate1());
    Assert.assertEquals(expectedEnd, timelineDate.getDate2());

}

/**
 * Passes in a normalized entity tag for an actual year range, 1980-01-01/2016-10-25,
     and checks the output to the
 * expected start and end Date.
 *
 * @throws ParseException when creating the expected Dates.
 */
@Test
public void testTimelineDateProcessActualDateRange() throws ParseException {
    String input = "1980-01-01/2016-10-25";
    String expectedStartDate = "1980-01-01";
    String expectedEndDate = "2016-10-25";

    Date expectedStart = simpleDateFormat.parse(expectedStartDate);
    Date expectedEnd = simpleDateFormat.parse(expectedEndDate);

    TimelineDate timelineDate = new TimelineDate();
    timelineDate.parse(input, baseDate);

    System.out.println(timelineDate);
    Assert.assertEquals(expectedStart, timelineDate.getDate1());
    Assert.assertEquals(expectedEnd, timelineDate.getDate2());
}

/**
 * Passes in a year and a season to backend.process.TimelineDate, and compares the
     output to an expected start and end Dates.
```

```
     *
     * @throws ParseException when creating the expected Dates.
     */
    @Test
    public void testTimelineDateProcessSeason() throws ParseException {
        String input = "1980-SP";
        String expectedStartDate = "1980-03-01";
        String expectedEndDate = "1980-05-31";

        Date expectedStart = simpleDateFormat.parse(expectedStartDate);
        Date expectedEnd = simpleDateFormat.parse(expectedEndDate);

        TimelineDate timelineDate = new TimelineDate();
        timelineDate.parse(input, baseDate);

        System.out.println(timelineDate);

        Assert.assertEquals(expectedStart, timelineDate.getDate1());
        Assert.assertEquals(expectedEnd, timelineDate.getDate2());
    }

    /**
     * Passes in a year with a week number to backend.process.TimelineDate, and compares
         the output to an expected start and end Dates.
     *
     * @throws ParseException when creating the expected Dates.
     */
    @Test
    public void testTimelineDateProcessWeekNumber() throws ParseException {
        String input = "2016-W47";
        String expectedStartDate = "2016-11-21";
        String expectedEndDate = "2016-11-27";

        Date expectedStart = simpleDateFormat.parse(expectedStartDate);
        Date expectedEnd = simpleDateFormat.parse(expectedEndDate);
```

```
    TimelineDate timelineDate = new TimelineDate();
    timelineDate.parse(input, baseDate);

    System.out.println(timelineDate);

    Assert.assertEquals(expectedStart, timelineDate.getDate1());
    Assert.assertEquals(expectedEnd, timelineDate.getDate2());
}

/**
 * Passes in a year and a month, and checks the output to an expected start Date.
 *
 * @throws ParseException when creating the expected Date.
 */
@Test
public void testTimelineDateProcessMonth() throws ParseException {
    String input = "2016-10";
    String expectedStartDate = "2016-10-01";

    Date expectedStart = simpleDateFormat.parse(expectedStartDate);

    TimelineDate timelineDate = new TimelineDate();
    timelineDate.parse(input, baseDate);

    System.out.println(timelineDate);

    Assert.assertEquals(expectedStart, timelineDate.getDate1());
}

/**
 * Tests the resulting date produced when a normalized entity tag for now (PRESENT_REF)
     is passed into TimelineDate.
 *
 * @throws ParseException when creating the expected Date.
 */
@Test
```

```
public void testTimelineDateProcessNow() throws ParseException {
    //the base date passed in should be the date used when it processes REFERENCE_NOW
    //base date being
    Date expectedDate = simpleDateFormat.parse(baseDate);

    TimelineDate timelineDate = new TimelineDate();
    timelineDate.parse(PRESENT_REF, baseDate);

    Assert.assertEquals(expectedDate, timelineDate.getDate1());
    Assert.assertEquals(null, timelineDate.getDate2());//shouldnt have a second date
}

/**
 * Tests the processing of the INTERSECT data (duration) that can come with a
     normalized entity tag for Dates.
 * Tests for a simple INTERSECT (4 Years).
 *
 * @throws ParseException when creating the expected Date.
 */
@Test
public void testTimelineDateProcessINTERSECT() throws ParseException {
    String expectedResultDuration = "Period: 4 Year(s)";
    Date expectedResultDate = simpleDateFormat.parse(baseDate);
    String input = baseDate + " INTERSECT P4Y";

    TimelineDate timelineDate = new TimelineDate();
    timelineDate.parse(input, baseDate);

    Assert.assertEquals(expectedResultDate, timelineDate.getDate1());
    Assert.assertEquals(null, timelineDate.getDate2());
    Assert.assertEquals(expectedResultDuration, timelineDate.getDurationData());
}

/**
 * Tests the processing of a more complex INTERSECT data (duration), including time.
 *
```

```
     * @throws ParseException when creating the expected Date.
     */
    @Test
    public void testTimelineDateProcessINTERSECTInclTime() throws ParseException {
        String expectedResultDuration = "Period: 3 Year(s) 6 Month(s) 4 Day(s) Time: 12
            Hour(s) 30 Minute(s) 5 Second(s)";
        Date expectedResultDate = simpleDateFormat.parse(baseDate);
        String input = baseDate + " INTERSECT P3Y6M4DT12H30M5S";

        TimelineDate timelineDate = new TimelineDate();
        timelineDate.parse(input, baseDate);

        Assert.assertEquals(expectedResultDate, timelineDate.getDate1());
        Assert.assertEquals(null, timelineDate.getDate2());
        Assert.assertEquals(expectedResultDuration, timelineDate.getDurationData());
    }

    /**
     * Tests the auto correct for Dates, where the day value has been over-estimated (that
         day value for that month
     * does not exist).
     *
     * @throws ParseException when creating the expected Date.
     */
    @Test
    public void testTimelineDateProcessAutoCorrectDates() throws ParseException {
        String inputWrongDate = "2016-12-32";
        String expectedDateStr = "2016-12-31";
        Date expectedDate = simpleDateFormat.parse(expectedDateStr);

        TimelineDate timelineDate = new TimelineDate();
        timelineDate.parse(inputWrongDate, baseDate);

        Assert.assertEquals(expectedDate, timelineDate.getDate1());
        Assert.assertEquals(null, timelineDate.getDate2());
    }
```

```
/**
 * Tests the auto correct for Dates, where the day value has been over-estimated, such
     that the month value has to
 * also be updated.
 *
 * @throws ParseException when creating the expected Date.
 */
@Test
public void testTimelineDateProcessAutoCorrectDatesAndMonths() throws ParseException {
    String inputWrongDate = "2016-12-00";
    String expectedDateStr = "2016-11-30";
    Date expectedDate = simpleDateFormat.parse(expectedDateStr);

    TimelineDate timelineDate = new TimelineDate();
    timelineDate.parse(inputWrongDate, baseDate);

    Assert.assertEquals(expectedDate, timelineDate.getDate1());
    Assert.assertEquals(null, timelineDate.getDate2());
}

/**
 * Tests the processing of a full BC normalized entity Date, of the format -yyyy-MM-dd.
 *
 * @throws ParseException when creating the expected Date.
 */
@Test
public void testTimelineDateBCFullDate() throws ParseException {
    SimpleDateFormat simpleDateFormatBC = new SimpleDateFormat("yyyy-MM-dd G");
    String input = "-0004-12-31";
    String expectedDateStr = "0004-12-31 BC";
    Date expectedDate = simpleDateFormatBC.parse(expectedDateStr);

    TimelineDate timelineDate = new TimelineDate();
    timelineDate.parse(input, baseDate);
```

```
    Assert.assertEquals(expectedDate, timelineDate.getDate1());
    Assert.assertNull(timelineDate.getDate2());
}

/**
 * Tests the processing of only a BC year normalized entity Date, of the format -yyyy,
     to see how TimelineDate gives
 * it a Date value (i.e. assumes it means the start of the year, so month 01 and day 01)
 *
 * @throws ParseException when creating the expected Date.
 */
@Test
public void testTimelineDateBCYearDate() throws ParseException {
    SimpleDateFormat simpleDateFormatBC = new SimpleDateFormat("yyyy-MM-dd G");
    String input = "-0004";
    String expectedDateStr = "0004-01-01 BC";
    Date expectedDate = simpleDateFormatBC.parse(expectedDateStr);

    TimelineDate timelineDate = new TimelineDate();
    timelineDate.parse(input, baseDate);

    Assert.assertEquals(expectedDate, timelineDate.getDate1());
    Assert.assertNull(timelineDate.getDate2());
}

/**
 * Tests the processing of a BC Date, that has a day value that is wrong for that
     month. Looks at how this Date is
 * reduced until it correct (assuming we over-estimated the day value).
 *
 * @throws ParseException when creating the expected Date.
 */
@Test
public void testTimelineDateBCWrongDate() throws ParseException {
    SimpleDateFormat simpleDateFormatBC = new SimpleDateFormat("yyyy-MM-dd G");
    String input = "-0004-12-32";
```

```
    String expectedDateStr = "0004-12-31 BC";
    Date expectedDate = simpleDateFormatBC.parse(expectedDateStr);

    TimelineDate timelineDate = new TimelineDate();
    timelineDate.parse(input, baseDate);

    Assert.assertEquals(expectedDate, timelineDate.getDate1());
    Assert.assertNull(timelineDate.getDate2());
}

/**
 * Tests the building of a range of Dates, for a BC date. Testing how the system infers
     the start and end Dates for
 * BC dates. (Should be the same as AD dates, just that the year values to towards 0 as
     it is B.C.)
 *
 * @throws ParseException when creating the expected Date.
 */
@Test
public void testTimelineDateBCDateRange() throws ParseException {
    SimpleDateFormat simpleDateFormatBC = new SimpleDateFormat("yyyy-MM-dd G");
    String input = "-04XX-12-32";
    String expectedDateStr1 = "0499-12-31 BC";
    String expectedDateStr2 = "0400-12-31 BC";
    Date expectedDate1 = simpleDateFormatBC.parse(expectedDateStr1);
    Date expectedDate2 = simpleDateFormatBC.parse(expectedDateStr2);

    TimelineDate timelineDate = new TimelineDate();
    timelineDate.parse(input, baseDate);

    Assert.assertEquals(expectedDate1, timelineDate.getDate1());
    Assert.assertEquals(expectedDate2, timelineDate.getDate2());
}

/**
 * Tests the case when the input is a week number with the day of the week.
```

```
     *
     * @throws ParseException when the expected date is created.
     */
    @Test
    public void testWeekDate() throws ParseException {
        String input = "2016-W47-7";//the seventh day in week 47 in iso 8601 is the sunday
        String expectedDateStr = "2016-11-27";//this the actual date for sunday
        Date expectedDate = simpleDateFormat.parse(expectedDateStr);
        TimelineDate timelineDate = new TimelineDate();
        timelineDate.parse(input, baseDate);
        Assert.assertEquals(expectedDate, timelineDate.getDate1());
        Assert.assertEquals(null, timelineDate.getDate2());
    }

    /**
     * Tests that when a Past reference is passed, a range of dates from the start of AD
         years until the base date is
     * created.
     *
     * @throws ParseException when the expected dates are created.
     */
    @Test
    public void testPastRef() throws ParseException {
        String input = PAST_REF;
        String expectedDateStr1 = "0001-01-01";
        String expectedDateStr2 = baseDate;//past ref points from start of year in AD to
            base date
        Date expectedDate1 = simpleDateFormat.parse(expectedDateStr1);
        Date expectedDate2 = simpleDateFormat.parse(expectedDateStr2);
        TimelineDate timelineDate = new TimelineDate();
        timelineDate.parse(input, baseDate);
        Assert.assertEquals(expectedDate1, timelineDate.getDate1());
        Assert.assertEquals(expectedDate2, timelineDate.getDate2());
    }

    /**
```

```
     * Tests that when a Future reference is passed, a range of dates from the base date
         until the end of times is
     * created.
     *
     * @throws ParseException when creating the expected dates.
     */
    @Test
    public void testFutureRef() throws ParseException {
        String input = FUTURE_REF;
        String expectedDateStr1 = baseDate;
        String expectedDateStr2 = "9999-12-31";
        Date expectedDate1 = simpleDateFormat.parse(expectedDateStr1);
        Date expectedDate2 = simpleDateFormat.parse(expectedDateStr2);
        TimelineDate timelineDate = new TimelineDate();
        timelineDate.parse(input, baseDate);
        Assert.assertEquals(expectedDate1, timelineDate.getDate1());
        Assert.assertEquals(expectedDate2, timelineDate.getDate2());
    }

    /**
     * Tests that TimelineDate enforces the Rules of expanding the Range of dates.
     * I.e. when pass a date which is earlier than the current date1, then date1 should be
         updated, or if a date is
     * passed in that is further away in the future than date2, then date2 should be
         updated. However, if a date is
     * passed in that is within the range, then the date1, and date2 values should not be
         changed.
     *
     * @throws ParseException when creating the expected Dates.
     */
    @Test
    public void testEnforceRule() throws ParseException {
        String input = baseDate;
        Date expectedDate1 = simpleDateFormat.parse(input);
        Date expectedDate2 = null;
        TimelineDate timelineDate = new TimelineDate();
```

```
        timelineDate.parse(input, baseDate);
        Assert.assertEquals(expectedDate1, timelineDate.getDate1());
        Assert.assertNull(timelineDate.getDate2());
        input = "2015-12-15";
        timelineDate.parse(input, baseDate);
        expectedDate1 = simpleDateFormat.parse(input);
        Assert.assertEquals(expectedDate1, timelineDate.getDate1());
        Assert.assertNull(timelineDate.getDate2());
        input = "2077-01-05";
        timelineDate.parse(input, baseDate);
        expectedDate2 = simpleDateFormat.parse(input);
        Assert.assertEquals(expectedDate1, timelineDate.getDate1());
        Assert.assertEquals(expectedDate2, timelineDate.getDate2());
        input = "2055-12-24";
        timelineDate.parse(input, baseDate);//dates shouldnt have changed as range hasnt
            expanded
        Assert.assertEquals(expectedDate1, timelineDate.getDate1());
        Assert.assertEquals(expectedDate2, timelineDate.getDate2());
    }

}
```

ToJSONTest.java

```
package backend;

import backend.helpers.ToJSON;
import backend.process.FileData;
import backend.process.Result;
import backend.process.TimelineDate;
import org.junit.Assert;
import org.junit.Test;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
```

```
/**
 * Test Class to test the toJSON class (ie taking a list of Results and producing JSON
     Strings).
 */
public class ToJSONTest {
    private static final SimpleDateFormat simpleDateFormat = new
        SimpleDateFormat("yyyy-MM-dd G");

    /**
     * Tests the JSON String produced for a List of one Result object which has all the
         data initialised (i.e. all the
     * fields in the JSON should be populated).
     *
     * @throws ParseException when producing the dates (date1, date2 from TimelineDate) for
         the test Result object.
     */
    @Test
    public void testCompleteJSON() throws ParseException {
        Result result = new Result();
        TimelineDate timelineDate = new TimelineDate();
        timelineDate.setDate1(simpleDateFormat.parse(returnDate("2016", "02", "14",
            false)));
        timelineDate.setDate2(simpleDateFormat.parse(returnDate("2016", "02", "15",
            false)));
        result.setTimelineDate(timelineDate);
        result.addSubject("Valentines Day");
        result.addSubject("Party");
        result.setEvent("On Valentines Day we had a huge party!");
        FileData fileData = new FileData("party.txt", "FAKEPATH");
        fileData.setCreationDate("02-02-2017");
        result.setFileData(fileData);

        ArrayList<Result> results = new ArrayList<>();
        results.add(result);
        //process the full result
```

```
    String actualResultJson = ToJSON.toJSON(results);
    String expectedJson = "[{\"date1\":\"14-02-2016 AD\",\"date2\":\"15-02-2016
        AD\",\"subjects\":[\"Valentines Day\",\"Party\"],\"event\":\"On Valentines Day
        we had a huge
        party!\",\"from\":{\"filename\":\"party.txt\",\"baseDate\":\"02-02-2017\"}}]";
    Assert.assertEquals(expectedJson, actualResultJson);
}

/**
 * Tests the JSON String produced for a List of one Result object, where only some of
     its data fields have been
 * populated (date1, event, and from fields should have data).
 *
 * @throws ParseException when producing the date1 in TimelineDate for the test Result
     object.
 */
@Test
public void testPartialJSON() throws ParseException {
    Result result = new Result();
    TimelineDate timelineDate = new TimelineDate();
    timelineDate.setDate1(simpleDateFormat.parse(returnDate("2017", "02", "14",
        false)));
    result.setTimelineDate(timelineDate);
    result.setEvent("On Valentines Day we had a huge party!");
    FileData fileData = new FileData("party.txt", "FAKEPATH");
    fileData.setCreationDate("02-02-2017");
    result.setFileData(fileData);

    ArrayList<Result> results = new ArrayList<>();
    results.add(result);

    String actualResultJson = ToJSON.toJSON(results);
    String expectedJson = "[{\"date1\":\"14-02-2017 AD\",\"subjects\":[],\"event\":\"On
        Valentines Day we had a huge
        party!\",\"from\":{\"filename\":\"party.txt\",\"baseDate\":\"02-02-2017\"}}]";
```

```
    Assert.assertEquals(expectedJson, actualResultJson);
}

/**
 * Tests the JSON String produced for a List of two Result objects, where one is fully
     populated (i.e. all fields
 * have non null values) and the other is partially populated (i.e. only date1, event
     and from fields are populated).
 *
 * @throws ParseException when producing the date1 and/or date2 in TimelineDate for the
     test Result objects (since
 *                     its a list).
 */
@Test
public void testMultipleResultJSON() throws ParseException {
    ArrayList<Result> results = new ArrayList<>();

    Result result = new Result();
    TimelineDate timelineDate = new TimelineDate();
    timelineDate.setDate1(simpleDateFormat.parse(returnDate("2016", "02", "14",
        false)));
    timelineDate.setDate2(simpleDateFormat.parse(returnDate("2016", "02", "15",
        false)));
    result.setTimelineDate(timelineDate);
    result.addSubject("Valentines Day");
    result.addSubject("Party");
    result.setEvent("On Valentines Day we had a huge party!");
    FileData fileData = new FileData("party.txt", "FAKEPATH");
    fileData.setCreationDate("02-02-2017");
    result.setFileData(fileData);
    results.add(result);

    result = new Result();
    timelineDate = new TimelineDate();
    timelineDate.setDate1(simpleDateFormat.parse(returnDate("2017", "02", "14",
        false)));
```

```
        result.setTimelineDate(timelineDate);
        result.setEvent("On Valentines Day we had a huge party!");
        fileData = new FileData("party.txt", "FAKEPATH");
        fileData.setCreationDate("02-02-2017");
        result.setFileData(fileData);
        results.add(result);

        String actualResultsJson = ToJSON.toJSON(results);
        String expectedJson = "[{\"date1\":\"14-02-2016 AD\",\"date2\":\"15-02-2016
            AD\",\"subjects\":[\"Valentines Day\",\"Party\"],\"event\":\"On Valentines Day
            we had a huge
            party!\",\"from\":{\"filename\":\"party.txt\",\"baseDate\":\"02-02-2017\"}},
         {\"date1\":\"14-02-2017 AD\",\"subjects\":[],\"event\":\"On Valentines Day we had
              a huge
              party!\",\"from\":{\"filename\":\"party.txt\",\"baseDate\":\"02-02-2017\"}}]";
        Assert.assertEquals(expectedJson, actualResultsJson);
    }

    /**
     * Tests the JSON String produced for a List of one Result object, that has just been
         initialised, and had no data
     * set to it.
     */
    @Test
    public void testBareResult() {
        Result result = new Result();
        ArrayList<Result> results = new ArrayList<>();
        results.add(result);

        String actualResultJson = ToJSON.toJSON(results);
        String expectedJson = "[{\"subjects\":[],\"event\":\"\",\"from\":{}}]";
        Assert.assertEquals(expectedJson, actualResultJson);
    }

    /**
     * Tests the JSON String produced for an empty List.
```

```
     */
    @Test
    public void testNoResults() {
        ArrayList<Result> results = new ArrayList<>();
        String actualResultJson = ToJSON.toJSON(results);
        String expectedJson = "[]";
        Assert.assertEquals(expectedJson, actualResultJson);
    }

    /**
     * Produces a date String of the format yyyy-MM-dd, for the given input.
     *
     * @param year the year of the date
     * @param month the month of the date
     * @param day  the day of the date
     * @return a String of the format yyyy-MM-dd
     */
    private String returnDate(String year, String month, String day, boolean isBC) {
        if (isBC) {
            return String.format("\%s-\%s-\%s BC", year, month, day);
        }
        return String.format("\%s-\%s-\%s AD", year, month, day);
    }

}
```

C.8 src/test/resources/backend

C.8.1 testfile1.txt, textfile2.txt, and textfile3.txt

```
On the 12th of December I ran tests on my final year project. The tests did not go so
    well. Yesterday I played games. It was fun playing games! Tomorrow I am going to
    study. Last week I went to watch a football match.
```

C.8.2 testfile4.docx

```
On Friday, the Washington Post came out with the latest from its long-running
    investigation into Trump's charitable donations.
In its latest story, the paper called 420-plus charities with some connection to Trump but
    found only one personal gift from him between 2008 and the spring of this year.
But the Post did find nearly $8m that Trump has donated from his own pocket since the
    early 1980s.
One of the bizarre episodes the paper recounts is that in 1996, Trump showed up without an
    invitation to a charity for the Association to Benefit Children where he took a seat
    on the stage that had been reserved for a major donor, despite not being a donor
    himself.
In response to the piece, the Trump campaign told the Post that he "has personally donated
    tens of millions of dollars... to charitable causes".
```

C.8.3 testfile5.pdf

```
On Friday, the US chose not to veto a UN Security Council resolution calling for an end to
    Israeli settlement construction, leading to an angry response from Israel. The issue
    of Jewish settlements is one of the most contentious between Israel and the
    Palestinians, who see them as an obstacle to peace and the creation of a viable
    Palestinian state. More than 500,000 Jews live in about 140 settlements built since
    Israel's 1967 occupation of the West Bank and East Jerusalem. The settlements are
    considered illegal under international law, though Israel disputes this.
```