System Design Document
for
NER Text Annotation Manager

Clients
Torsten Hahmann
Umayer Reza

SEWDO Developers
Sam Waggoner
Wilder Baldwin
Darien Orethun
Owen Bellew

Table of Contents

Name	**Date**	**Reason for Changes**	**Version**
All	11/06/23	Original creation	1.0
All	TBD	Incorporating peer feedback	1.1

1 Introduction

This section introduces the project background, motivation, goals, and desired outcome.

Umayer Reza, a graduate student at the University of Maine, is working with faculty professor Torsten Hahmann. Reza's Ph.D project involves generating cellulosic terms from scientific publications for populating and refining an ontology-based knowledge graph in a domain within forestry and agriculture. Reza is using a language model that reviews published literature on the subject of forestry and agriculture, which is then able to learn the states, and the relationship between them. However, the raw sentences in the literature are not enough to train this model–to learn the content, it needs labels indicating what words are the states and actions. To get these labels, experts in the field must manually label the text, sentence by sentence. However, the tool that they use to create these labels is time-consuming to use in several key ways. This is where Umayer Reza and Dr. Hahmann requested the help of SEWDO. The NER Text Annotation Manager is an existing software tool that allows users to add annotations of text, which can then be used for Named Entity Recognition (NER) tasks. Text annotations are labels of words or sections of a sentence. For example, 'sulfuric acid' and 'chlorine' could both be labeled as 'chemical', which would help the NER model learn what chemicals are. This project aims to add or improve three primary features: annotation review, annotation editing, and importing and exporting provenance data in a single JSON file. The project aims to include these features with a user interface that enables quick, simple, and intuitive annotation reviews. By adding these features, human annotators will be able to produce labels much faster and much easier, thus creating more data and improving the model that Reza is creating. In addition, the field experts' time is valuable. These improvements will mean that it will be easier to train new annotators and contributions will demand less of experts' time. The addition of these features is the capstone project for the members of our team (Sam Waggoner, Wilder Baldwin, Darien Orethun and Owen Bellew), in partial fulfillment of the Computer Science BS degree for the University of Maine. This project was requested by Torsten Hahmann and Umayer Reza.

1.1 Purpose of This Document

This section describes why this document was created, and what purpose it serves.

The System Design Document (SDD) for the NER Text Annotation Manager project serves as a comprehensive guide for both the customer and clients involved in the project, outlining the system's architecture, features, and intended improvements. It provides a clear understanding of the responsibilities of the project team and the technical details of the system implementation. The primary purpose of the SDD is to provide a detailed description of the NER Text Annotation Manager system, ensuring the project's design is aligned with the objectives and technical requirements as specified in the SRS. It serves as a reference document for the project team, ensuring goals, systems and development remain consistent and organized. The document also plays a crucial

role in communication between the client and the customer about the project's current scope and realistic design functionality. It opens room for discussion, which in turn provides valuable feedback to ensure the projects meet the specified needs. It also provides several detailed visual representations of how the system operates through design diagrams, allowing all parties to evaluate the functionality and applications of the system. Detailed documentation regarding existing functionality of files and file operations within the system as well as modified functionality enhance clarity within the scope of the project. This document is primarily for planning and clarification of the application design for SEWDO team members. Secondarily, it is for the clients, Torsten Hahmann and Umayer Reza, our peer QA team IMSG, and COS 397 graders so that they can understand, approve, or critique our intended design.

1.2 References

This section includes citations for the external references that were mentioned or utilized in this document. It is presented in two ways for the reader's ease of accessing links and information.

Fowler, M. (2004). *UML distilled: A brief guide to the standard object modeling language*. Addison-Wesley.

Arunmozhi, ad, Khan, A., Kunert, L., & Göller, D. (2021, December 30). Ner-AnnotatorVersion (1.3.0). *ner-annotator*. Retrieved 2023, from https://github.com/tecoholic/ner-annotator.

Waggoner, S., Orethun, D., Baldwin, W., & Bellew, O. (2023). *Software Requirements Specification for NER Text Annotation Manager* (pp. 1–27). Retrieved 2023, from https://docs.google.com/document/d/1stujhTD_k6uMkUDNC4R1CHnKlx_UubsRXVFx0pY2gHc/edit

Bibliography

Fowler, M. (2003). *UML Distilled*. https://martinfowler.com/books/uml.html

Waggoner, S. Baldwin, W. Orethun, D. Bellew, O.(2023) *SEWDO System Requirements Specifications(SRS) for NER annotator.*

Arunmozhi. (2023). *NER Annotator for Spacy* [Vue]. https://github.com/tecoholic/ner-annotator (Original work published 2020)

Fowler, Martin. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. 3rd ed, Addison-Wesley, 2004

2 System Architecture

This section details the systems architecture of our project. We include a high level architectural design, the current system architecture we are working from and a decomposition diagram detailing the classes, functions and their relationships throughout the system we are building.

2.1 Architectural Design

This section goes over the logical architecture of our system, explaining verbally and with diagrams. It will list hardware and software dependencies. It will also show a class diagram of the existing system.

Our UML diagram depicts the different states the software will be able to enter with our extension. The homepage will have similar functionality to how it is currently. However, there will be UI elements that allow the user to enter Annotation Mode and Review Mode. Each mode changes the functionality available to the user. In Annotation Mode, the user will be able to Add Annotations and Delete Annotations, which are already present functionalities, but the user will newly be able to Undo annotation changes, as well as Export Annotations to a JSON file in a specified format.

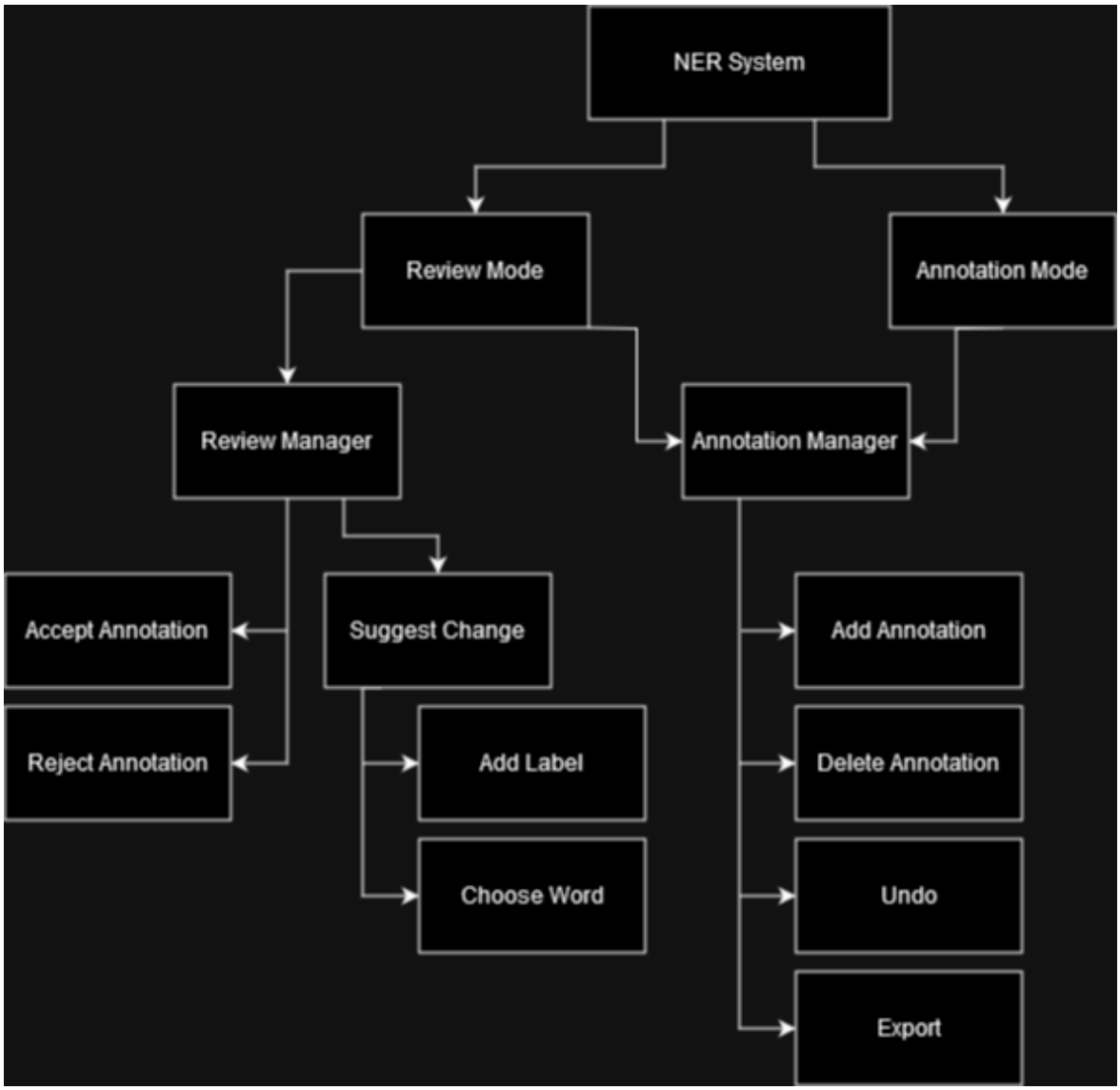

Fig. 2.1.1: Logical architecture diagram

An entirely new feature as seen above in Fig. 2.1.1 will be the Review Mode state. While this state maintains the functionality of the Annotation Mode, it will also feature several elements of annotation review such as Accept Annotation and Reject Annotation which allow the annotations to be marked for any changes needed. Additionally, the user will be able to use Suggest Change for a recommended annotation instead of the current one.

There is no hardware involved in our additions to the project. The only hardware involved is the server on which the website is running, and the client laptops used to access the website. We do not know how or where the existing server is being run, but if our features are added into the main branch of the GitHub project, then the repository owner (Tecoholic) can integrate our changes.

This project uses the following software libraries/dependencies:

- Tauri
- Yarn
- Vue.js
- Rust

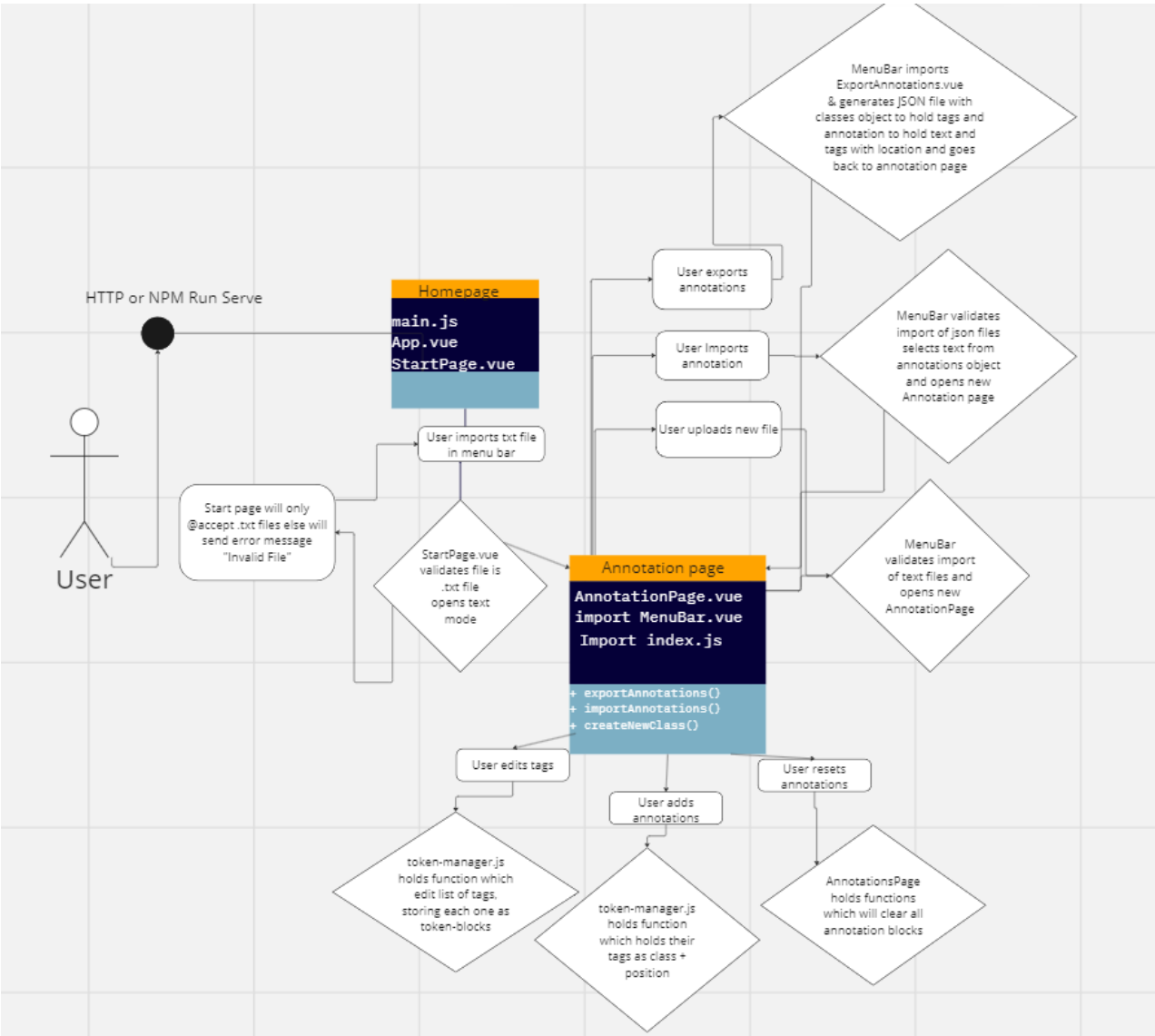

Fig. 2.1.2: Existing software class diagram

The legacy software of our project allows for a user to enter the homepage and upload a text file which will bring them into annotation mode. Currently this annotation mode allows users to add their own named entity recognition labels and attach them to words within their uploaded text file.

Once the user uploads a file the home page will verify it is a text file and open the annotation page with the text loaded for annotation. In the annotation page the user has the ability to edit the current array of tags, add new annotation and reset annotations. From the annotation page the user can upload new, import lists of annotations and export the annotations in json format. This is shown in detail of functionality above in Fig. 2.1.2.

66% of the existing application is written in Vue JS. Our team members are in the process of learning this framework. Thus, our understanding of how this application works is not rock-solid. However, to our best understanding Vue is a library integrated into Javascript that functions by creating instances of Vue. Vue is a component centered framework, so components can be made to have their own data, methods, and template. This is a great addition for our project, as extending features is easier in a component based model.

Yarn, Tauri and Rust serve a much more secondary purpose in the software. Yarn is the package manager for the project. Tauri is a toolkit with a Rust base that aids in the deployment of web apps for major platforms by leveraging Node.js. While Tauri works well by default for React, Svelte, and Yew it can be easily configured for use with Vue JS. Rust is necessary for building desktop versions of the project.

2.2 Decomposition Description

This section elaborates on the previous section, visualizing and describing a more detailed class diagram of our modified system.

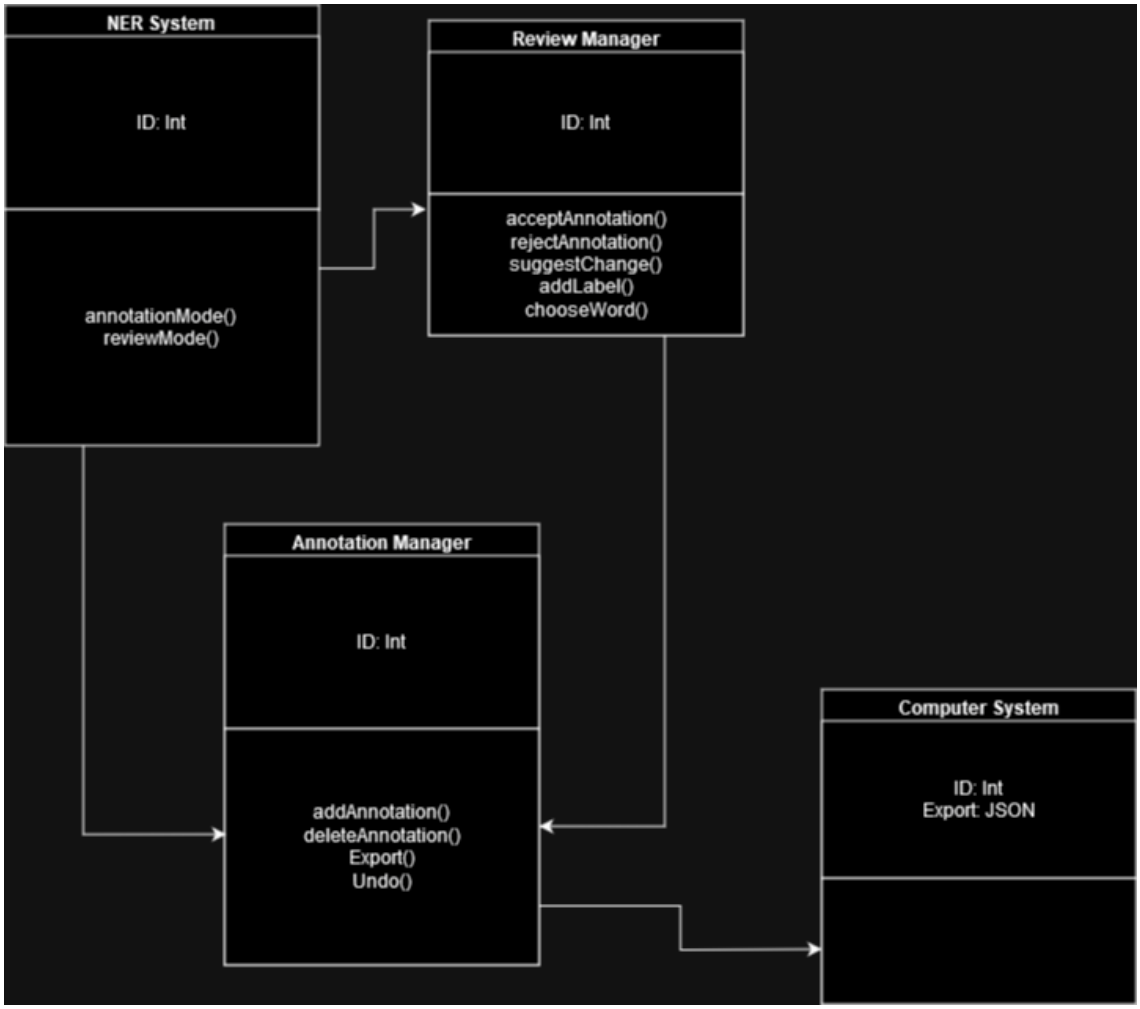

Fig. 2.2.1: Modified software class diagram

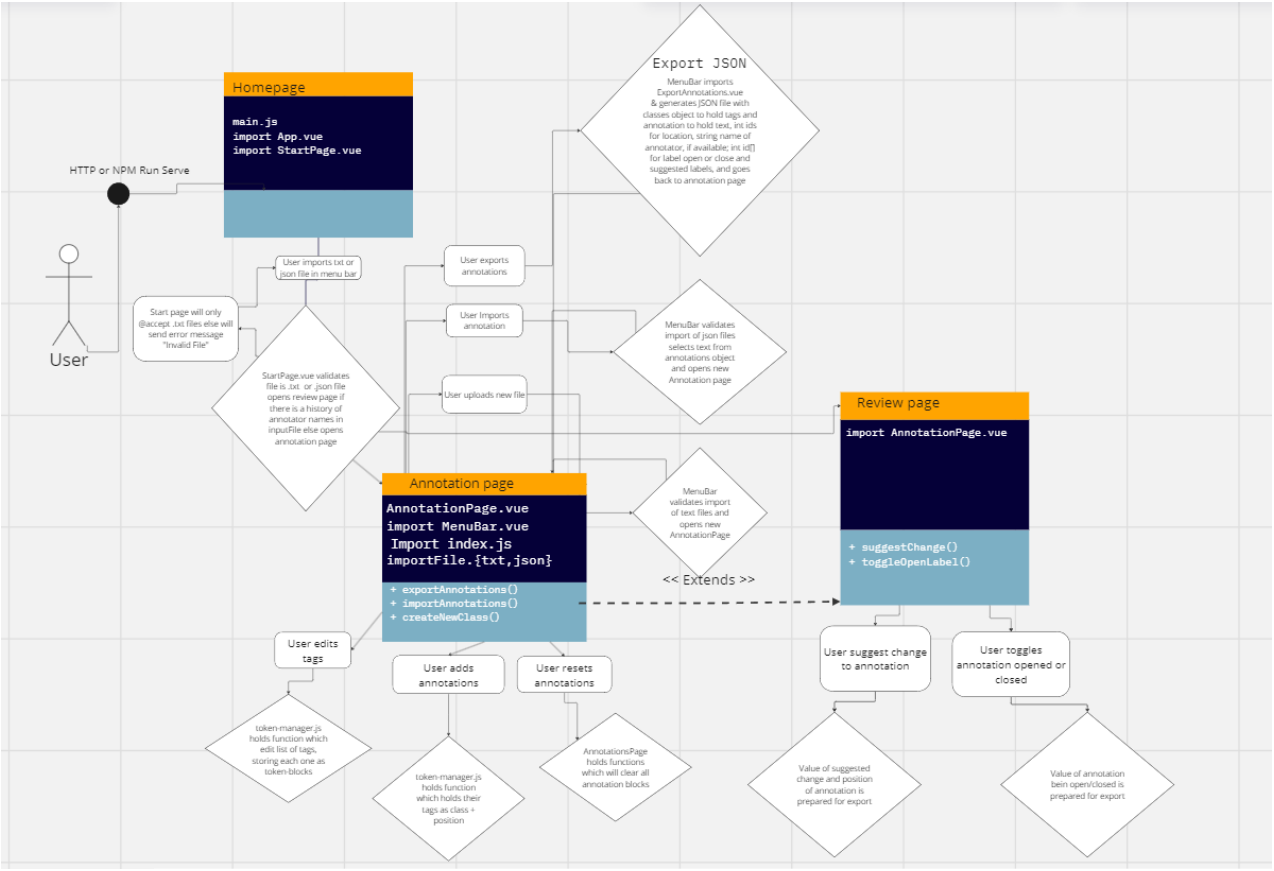

Fig. 2.2.2: Decomposition diagram of systems architecture

Our diagram (Fig. 2.2.1) uses a simplified state machine diagram predefined pattern. Our diagram displays the different states the program can be in and what features are available then. Notably, the Review Manager is able to access the features of the Annotation manager but the reverse is not true, which is an important feature to display.

Our diagram is also displayed by the interactions with the underlying NER system and the user's computer system. The Annotation Manager has the function to export a .txt file, which writes a file to the user's chosen directory. The switch between different state objects, called managers, is facilitated by the NER system, the underlying code for the states invoked by the manager objects,

Our diagram (Fig 2.2.2) illustrates the functional decomposition of our system and the pages which the user will interact with. From the already existing home page the user will input a txt or json file which will trigger their being sent to annotation page or review page depending on the format and contents of the file. The review mode extends the annotation page of the existing software architecture document and adds new features. These features are the ability to toggle a label open or closed and suggest changes to labels. Additionally new functionality has been added to export json allowing for the user to add their name for a historic record.

3 Persistent Data Design

This section details the related database and files involved with the system. Examples of the required files and formats are included and described.

3.1 Database

The existing application does not use any databases, since files are uploaded and exported, and there is no need to store large quantities of information beyond a browser session. Our added features will not require a database.

3.2 File Descriptions

This section details with examples the types of files which will be used within our application. These imported and exported files are critical to the system's functionality.

3.2.1 Imported Text Files

This section describes the required format of an imported text file.

In order to create annotations from scratch, the user will upload a text file. This file is "standard" .txt format (meaning it can use ANSI, OEM, Unicode, or UTF-8 character encoding). It cannot be .docx, pages, or any other format other than .txt. The file should contain characters with paragraphs separated by newline characters (\n). The file name is irrelevant. Below is an example of a text file that a user could upload.

```
Cellulose is an organic compound with the formula (C6H10O5)n, a
polysaccharide consisting of a linear chain of several hundred
to many thousands of linked D-glucose units.

It is an important structural component of the primary cell wall
of green plants, many forms of algae, and the oomycetes, and is
Earth's most abundant organic polymer.
Source: Wikipedia
```

Fig. 3.2.1.1: Example of a viable text file for upload

3.2.2 Annotation Files (Existing Functionality)

This section describes the required format of annotation files that are imported and exported in the **current** system. The next section, 3.2.3, will show our changes to these requirements.

NER models are created to add annotations to text. Thus, it can be quicker to create ground truth files by reviewing the annotations of an NER model and making corrections, rather than creating annotations from scratch. It is also a common case for multiple domain experts to review the same file to ensure that annotations are correct. In both of these situations, a user would want to upload and export an existing annotated body of text for review. These files are in JSON format.
In the existing application, to import annotations the user must:
1. upload a text document, as if they were going to annotate from scratch
2. upload the annotations JSON file (Annotations -> Import -> OK -> Open)
3. upload the tags JSON file (Tags -> Import -> Open)

The text document is the .txt file that was originally used to create the annotations. This example builds off Fig. 3.2.1.1 discussing cellulose.

An example of an annotations JSON file with labels is included below in Fig. 3.2.2.1. The empty '\r' with no entities shows up because of the whitespace newline between the two paragraphs. Please note that the annotations are only for demonstration purposes and are not correct.


```
{
    "classes": [
        "COMPOUND",
        "PROCESS_OR_TECHNIQUE",
        "MEASUREMENT"
    ],
    "annotations": [
        [
            "Cellulose is an organic compound with the formula
(C6H10O5)n, a polysaccharide consisting of a linear chain of several
hundred to many thousands of linked D-glucose units.\r",
            {
                "entities": [
                    [
                        0,
                        9,
                        "COMPOUND"
                    ],
                    [
                        16,
                        32,
                        "PROCESS_OR_TECHNIQUE"
                    ],
                    [
                        64,
                        78,
```

```
                        "PROCESS_OR_TECHNIQUE"
                    ],
                    [
                        148,
                        170,
                        "PROCESS_OR_TECHNIQUE"
                    ]
                ]
            }
        ],
        [
            "\r",
            {
                "entities": []
            }
        ],
        [
            "It is an important structural component of the primary
cell wall of green plants, many forms of algae, and the oomycetes,
and is Earth’s most abundant organic polymer.\r",
            {
                "entities": [
                    [
                        0,
                        2,
                        "COMPOUND"
                    ]
                ]
            }
        ],
        [
            "Source: Wikipedia\r",
            {
                "entities": []
            }
        ]
    ]
}
```


Fig. 3.2.2.1: Existing format for annotations JSON file

The entire object in Fig. 3.2.2.1 contains ‘classes’ and ‘annotations’. ‘classes’ is a list containing strings of all of the possible labels. ‘annotations’ is a list of lists. Each list within ‘annotations’ has a first element containing the paragraph string, and a second element containing the associated labels object. This labels object contains a list called ‘entities’. Each element of ‘entities’ contains two numbers and a tag name. The two numbers are the indexes of the starting and ending characters in the paragraph for the given label. For example, in the second paragraph, ‘0, 2, “COMPOUND”’ indicates that the first and second characters, ‘It’, are labeled as a compound. The first number is inclusive, the second number is exclusive.

Then, if the user wanted to upload the tags, they would upload a file similar to Fig. 3.2.2.2 below.

```
[
    {
        "id": 1,
        "name": "COMPOUND",
        "color": "red-11"
    },
    {
        "id": 2,
        "name": "PROCESS_OR_TECHNIQUE",
        "color": "blue-11"
    },
    {
        "id": 3,
        "name": "MEASUREMENT",
        "color": "light-green-11"
    }
]
```

Fig. 3.2.2.2: Existing format for tags JSON file

This file (Fig. 3.2.2.2) is simpler, and contains a list of objects, where each object contains an ‘id’ integer, ‘name’ string, and ‘color’ string. After the user uploads this tags JSON, they would have the text, annotations, and label colors loaded in the application. After this point, they can annotate as they wish. The exported annotations file format is the same as the formats described in this section.

3.2.3 Annotation Files (Modified Functionality)

This section will show the required format for annotation files that are imported and exported **after our modifications** of the system. This section contrasts with the previous section, 3.2.2.

SEWDO's contribution is to conglomerate the text file, the annotations JSON file, and the tags JSON file into one. In this way, the annotator's process is expedited. Instead of the three files as shown in the previous section, there will only be one file that the user has to upload. In addition, each annotation will be associated with a change history that lists the dates at which it was at each action, the name of the person who performed the action, and the suggested change (if applicable).

Action can be one of the following:
- *Created*: When the annotation was first made. This could be on a text file or an annotations file that already had labels. This adds an extra field containing the label name.
- *Accepted:* (Optional) When the annotation was accepted by an annotator.
- *Rejected:* (Optional) When the annotation was rejected by an annotator.
- *Suggested:* (Optional) When an annotator suggested another label for the annotation. This adds an extra field that contains the name of the suggested label. This updates the actual label to be the new suggested label.

Sticking with the previous example, the file will follow the form of the file below, Fig. 3.2.3.1. Note that this version only includes the first paragraph, since all of the demonstrated changes can be viewed in this shorter version.

```
{
    "classes": [
        {"id":1, "name":"COMPOUND", "color":"red-11"},
        {"id":2, "name":"PROCESS_OR_TECHNIQUE", "color":"blue-11"},
        {"id":3, "name":"MEASUREMENT", "color":"light-green-11"}
    ],
    "annotations": [
        [
            "Cellulose is an organic compound with the formula
(C6H10O5)n, a polysaccharide consisting of a linear chain of several
hundred to many thousands of linked D-glucose units.\r",
            {
                "entities": [
                    [
                        0,
                        9,
                        "COMPOUND",
                        [["Created", "2023-10-01", "Mohammad"],
                         ["Accepted","2023-10-08", "Andrew"]]
                    ],
                    [
                        16,
                        32,
                        "PROCESS_OR_TECHNIQUE",
```

```
                        [["Created", "2023-10-01", "Mohammad"],
                         ["Rejected","2023-10-08", "Andrew"]]
                    ],
                    [
                        64,
                        78,
                        "PROCESS_OR_TECHNIQUE",
                        [["Created", "2023-09-25", "NER Annotator"],
                         ["Rejected","2023-10-01", "Mohammad"],
                         ["Rejected","2023-10-08", "Andrew"]]
                    ],
                    [
                        148,
                        170,
                        "PROCESS_OR_TECHNIQUE",
                        [["Created", "2023-09-25", "NER Annotator"],
                         ["Rejected","2023-10-01", "Mohammad"],
                         ["Accepted","2023-10-08", "Andrew"],
                         ["Suggested","2023-10-16", "Henry",
"COMPOUND"]]
                    ]
                ]
            }
        ]
    ]
}
```

Fig. 3.2.3.1: Modified format for combined annotation and tags JSON file

In our single JSON file (Fig. 3.2.3.1), the ‘classes’ have been combined, such that instead of just strings, it contains objects, each containing the label’s id, name, and color, just like in the tags file. The other change is the addition of the history, which is a 2D list after every element in ‘entities’. This list will always contain a first list containing the ‘Created’ action, followed by the date (yyyy-mm–dd) and a string containing the name of the person who created the annotations. There will be additional lists within the 2D list if other users have reviewed the annotation since its creation. Every inner list will contain the Action string (as described before Fig. 3.2.3.1) followed by the date (in the same format) and name. Note that the name can contain spaces, if the user entered their whole name. The name input will be sanitized such that it is <50 characters (including spaces) and contains only UTF-8 characters. The exported format is the same format as the format described above for Fig. 3.2.3.1.

Notice that in the stored history, actions such as Undo and Reopen are not present. This is because they are scoped to the session, and their histories are discarded upon

exporting. Also note that Annotation Mode actions other than create (namely Replace and Delete) are not actions, and are not recorded in history. This is because Replace and Delete are final, and directly change or delete the annotation.

It is important that Accepting, Rejecting, and Suggesting an annotation do not change the actual label. When a label is marked as rejected, it is not desired functionality to delete it from the annotations. This is because although one annotator could reject it, another annotator could accept it. If it is deleted, no other annotator will be able to view it. Thus, any previously rejected label will still show up like any other non-rejected label when it is imported. (This situation is demonstrated in the third and fourth labels of the first paragraph in Fig. 3.2.3.1.) If Suggesting changed the actual label, then it would be no different than Replace. Accepting does not change the label at all.

This leads to why SEWDO needs to add an additional button that actually utilizes the histories. As it stands, history is just a recording that has no bearing on the labels. Thus, a button that resolves all Accepts, Rejects, and Suggestions into (accepting) no action, deletion, and replacements is necessary. The user must also specify what action to take in the case of a tie. For example, a user may prioritize Rejected reviews over Suggested reviews over Accepted reviews over Creation if they want to create a model that has high recall, since this will produce a dataset with fewer labels. This will go through each annotation and act upon the review action that takes the majority. When finding the majority, the Creation label is considered as an Accept.

Several examples of deciding the majority will now be presented. For a given annotation, if there is one Creation (as there is with all annotations), two Accepts, and two Rejects, then the label will not be changed, since Accepts/Creation takes the majority (3-2-0). For further examples, Creation will be implied. For another annotation, if there are two Accepts, two Rejects, and four Suggestions for the same label, then the suggested label will become the new label (3-2-4). If the suggestions had been split between two different labels, then the label would not be changed to any of the suggested labels–it would be accepted and not change (3-2-2). If there is one Reject and one Suggestion, then the user's predefined preference would dictate the result, since it is a three-way tie (1-1-1). This button has not been included in this document or the previous document as of yet, since its inclusion is pending discussion with the client on 11/8/23.

4 Requirements Matrix

The previous sections have described how the system works. The first table (Fig. 4.1) in this section will show where each functional requirement is satisfied in our architecture. The second table (Fig. 4.2) will visualize each function and where it is documented in our functional requirements.

FR	Functions/Methods
FR-1 Import JSON Annotation	{Web Interface} -> Upload file for Review Mode

	/ -> Upload file for Annotate Mode
FR-2 Export JSON Annotations	{Review Mode/ Annotation Mode} -> Backend -> Export Annotation
FR-3 Undo Action	{Review Mode/Annotation Mode} -> Undo
FR-4 Replace label in Annotation Mode	{Review Mode/Annotation Mode} -> Replace Annotation
FR-5 Accept or reject label in Review Mode	{Review Mode} -> Accept Annotation / -> Reject Annotation
FR-6 Suggest label in Review Mode	{Review Mode} -> Suggest Change
FR-7 Reopen label in Review Mode	{Review Mode} -> Reopen Label

Fig. 4.1: Correlating functional requirements and functions/methods

System Components	FR-1	FR-2	FR-3	FR-4	FR-5	FR-6	FR-7
Import Function	**X**			**X**			
Export Function		**X**					
Undo Function			**X**				
Replace Label Function				**X**			
Accept/Reject Function					**X**		
Suggest Label Function						**X**	
Reopen Function							**X**
Review Mode		**X**	**X**	**X**	**X**	**X**	**X**
Annotation Mode		**X**	**X**	**X**			
Web Interface	**X**						

Fig. 4.2: Visualizing which components satisfy which functional requirements

Appendix A – Agreement Between Customer and Contractor

This document signifies a mutual agreement between the customer (NER Text Annotation Manager) and the project team (SEWDO) regarding the project requirements as described in the SDD document. Both parties acknowledge that the project aims to enhance the NER Text Annotation Manager by adding features such as annotation review, annotation editing, and importing and exporting provenance data in a single file in JSON format. All parties agree that these requirements have been discussed and understood and to be executed as described.

In the event that changes are to be made in the future to this document, a formal change request procedure will be followed. The customer or any member of the project team may propose modifications by submitting a change request in writing. The request will be reviewed and evaluated by the project team, and if approved, the document will be updated accordingly. All changes will be documented and communicated to all relevant parties.

Signature (Client)

x_____	Name (Typed)_____	Date:_____
x_____	Name (Typed)_____	Date:_____

Signature(Project Team)

x*Owen Bellew*	Name (Typed) Owen Bellew	Date:11/7/23
x*Darien Orethun*	Name (Typed) Darien Orethun	Date:11/7/23
x*Samuel Waggoner*	Name (Typed) Samuel Waggoner	Date:11/7/23
x *Wilder Baldwin*	Name (Typed) Wilder Baldwin	Date:11/7/23

Appendix B – Team Review Sign-off

This document attests that all members of the SEWDO project team have reviewed the entirety of the “System Design Document " (SDD), and are in unanimous agreement regarding its content, format and specifications. The team members concur that the document accurately represents the system architecture and design according to the specified requirements of the project. This includes Architectural Design, Decomposition Descriptions, File Descriptions, and the Requirements Matrix. The team also agrees that there are no points of contention throughout the entirety of the document, but in the event they do arise, the provided space below will be used as a common ground open for discussion and suggestions. This will ensure that even the smallest constructive criticisms don’t go unnoticed and such comments are resolved in a timely manner.

The following have read, reviewed and understand the specified requirements of this document and the SDD.

Signature(Project Team)		
x*Owen Bellew*	Name (Typed) Owen Bellew	Date:11/7/23
x*Darien Orethun*	Name (Typed) Darien Orethun	Date:11/7/23
x*Samuel Waggoner*	Name (Typed) Samuel Waggoner	Date:11/7/23
x *Wilder Baldwin*	Name (Typed) Wilder Baldwin	Date:11/7/23

Appendix C – Document Contributions

This section describes the contributions of each team member to this document.

Sam Waggoner 25%
- Wrote Introduction and References
- Contributed to Purpose
- Wrote one paragraph for Architectural Design
- Wrote File Descriptions
- Wrote introduction sentence for all sections

Darien Orethun 25%
- Created logical architecture diagram
- Created class diagram
- Wrote Architectural design

Wilder Baldwin 25%
- Created existing software architecture diagram & wrote description paragraphs
- Created the first table for Requirements Matrix

Owen Bellew 25%
- Wrote Purpose
- Appendix A
- Appendix B
- Created the second table for Requirements Matrix

Questions for client:
- OK? Single upload location for the user, the user doesn't know about Review vs. Annotation Mode, that is just behind the scenes. txt file -> annotation mode, JSON file -> review mode?
    -
- OK? To get the name to include in the histories, just have a popup when the user clicks 'Export'. No login system.
    - Correct
- OK? Changes to history instead of the way Umayer did it in his email
    - No
- OK? Notice that in the stored history, actions such as Undo and Reopen are not present. This is because they are scoped to the session, and their histories are discarded upon exporting. Also note that Annotation Mode actions other than create (namely Replace and Delete) are not actions, and are not recorded in history. This is because Replace and Delete are final, and directly change or delete the annotation.
    - No, partially. Undo is not recorded. Reopen is recorded.
- OK? Purpose of review mode actions? Button to take majority rules and act on the majority?
    - No button. Every review is uploaded to their database. Do not stack reviews.