Universidade de Aveiro
2021

André
Sousa Neves

# Ferramenta Colaborativa de Anotação e Mapeamento de Conceitos Clínicos

# Collaborative Annotation and Mapping Tool for Clinical Concepts

Universidade de Aveiro
2021

André
Sousa Neves

# Ferramenta Colaborativa de Anotação e Mapeamento de Conceitos Clínicos

# Collaborative Annotation and Mapping Tool for Clinical Concepts

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor Sérgio Matos, Professor Auxiliar do  da Universidade de Aveiro.

Texto Apoio financeiro do POCTI no âmbito do III Quadro Comunitário de Apoio.

Texto Apoio financeiro da FCT e do FSE no âmbito do III Quadro Comunitário de Apoio.

Dedico este trabalho aos meus pais e irmão por sempre me apoiarem durante o desenvolvimento da dissertação.

## agradecimentos / acknowledgements

Gostaria de agradecer em primeiro lugar ao Prof. Doutor Sérgio Matos e ao Investigador João Rafael Almeida pela oportunidade de desenvolver este trabalho, por todo o apoio prestado e principalmente por me motivarem em alturas menos produtivas.
Agradeço também aos meus pais e irmão por sempre me apoiarem nesta jornada e por me fornecerem todas as condições necessárias para concluir esta etapa.
Dedico um agradecimento especial a todos os meus colegas de curso que me acompanharam nestes 5 anos e me permitiram desenvolver tanto a nível social como a nível académico, por sempre se preocuparem e ajudarem incondicionalmente, sem eles não seria possível.
Por último, gostaria de agradecer aos amigos que passaram comigo horas em chamadas enquanto fomos supreendidos por uma pandemia, pois foram um suporte muito importante para ultrapassar a mesma, no sentido em que a sensação de confinamento deixava de existir e de alguma forma continuávamos ligados.

**Palavras Chave**

Mineração de Texto Biomédico, Reconhecimento de Entidades Mencionadas, Processamento de Linguagem Natural, Recolha de Informação, Extração de Informação, Mapeamento, Conceitos de Vocabulário Padrão.

**Resumo**

Todos os dias são publicadas novas informações biomédicas sob a forma de artigos de investigação, livros e relatórios, mas dada a sua forma não-estruturada não é útil para a aquisição de conhecimento para além da pesquisa por palavras-chave. Ao longo dos anos tem surgido um interesse significativo na mineração de texto e a produção de dados estruturados, utilizando técnicas de recuperação de informação e extração de informação, nomeadamente o reconhecimento de entidades mencionadas. Foram desenvolvidas várias ferramentas de processamento de linguagem natural com o objetivo principal de auxiliar a tarefa manual intensiva realizada por curadores especialistas, implementando pipelines automáticos de pré-processamento que anotam entidades biomédicas e as relações entre si na literatura, juntamente com interfaces interativas para as rever e validar. Além disso, é essencial que os dados sejam harmonizados num padrão comum que todos possam compreender, independentemente da língua, formato ou codificação em que foram originalmente registados, a fim de proporcionar um esforço colaborativo entre os investigadores. Algumas ferramentas proporcionam capacidades eficientes de indexação e pesquisa para mapear conceitos de vários domínios em conceitos de vocabulários padrão, ou por outras palavras, são capazes de padronizar os dados num formato comum que, por sua vez, permite a realização de estudos colaborativos. No entanto, ferramentas que permitem realizar tanto a anotação como o mapeamento são escassas. Esta dissertação apresenta uma ferramenta web-based com a intenção de preencher esta lacuna, permitindo aos especialistas realizar cada tarefa individualmente, mas também formar um pipeline e utilizar as anotações resultantes como input para o processo de mapeamento. Como resultado, a ferramenta fornece uma interface interativa que permite aos utilizadores carregar documentos de texto e anotar entidades biomédicas presentes nos mesmos, quer manualmente selecionando porções de texto ou palavras com duplo clique, quer automaticamente com os serviços web do Neji e gerir as anotações geradas. Para mapeamento, os utilizadores podem carregar documentos CSV contendo termos para serem mapeados para conceitos de vocabulário padrão, utilizando o código open-source do Usagi. Além disso, os utilizadores podem rever e validar os mapeamentos sugeridos com base na pontuação dos mesmos.

**Keywords** Biomedical Text Mining, Named Entity Recognition, Natural Language Processing, Information Retrieval, Information Extraction, Mapping, Standard Vocabulary Concepts.

**Abstract**

Every day new biomedical information is published in the form of research articles, books and reports, but given its unstructured form it is not useful for knowledge acquisition apart from keyword search. Over the years significant interest has been generated towards text mining and the production of structured data using information retrieval and information extraction techniques, namely named entity recognition. Several natural language processing tools were developed with the main purpose of aiding the manual labor-intensive task conducted by expert curators by implementing automatic pre-processing pipelines that annotate biomedical entities and their relationships in literature, along with interactive interfaces to review and validate them. Moreover, it is essential that the data is harmonized into a common standard that everyone can understand no matter what language, format or encoding it was originally recorded in, in order to provide a collaborative effort among researchers. Some tools provide efficient indexing and searching capabilities to map concepts from various domains into standard vocabulary concepts, or in other words are capable of standardize data into a common format which in turn allow collaborative studies to be conducted. Nevertheless, there is a lack of tools that allow to perform both annotation and mapping. This dissertation presents a web-based tool with the intent to fill this gap by allowing experts to still perform each task individually, but also to form a pipeline and use the output annotations as input for the mapping process. As a result, the tool provides an interactive interface that allows the users to upload text documents and annotate biomedical entities present in them, either manually by selecting portions of text or double clicking words, or automatically with Neji's web services and manage those generated annotations. For mapping, the users can upload CSV documents containing terms to be mapped to standard vocabulary concepts, using Usagi's open-source code. Moreover, the users can review and validate suggested mappings based on match score.

# Contents

# List of Figures

# List of Tables

# Acronyms

| | |
|---|---|
| **AO** | Annotation Ontology |
| **API** | Application Programming Interface |
| **BioTM** | Biomedical Text Mining |
| **CA** | Certificate Authority |
| **CSV** | Comma-separated Values |
| **CDM** | Common Data Model |
| **CRF** | Conditional Random Field |
| **CoNLL** | Conference on Natural Language Learning |
| **CRUD** | Create, Read, Update and Delete |
| **CORS** | Cross-Origin Resource Sharing |
| **DB** | Database |
| **DOM** | Document Object Model |
| **EHR** | Eletronic Health Record |
| **ETL** | Extract Transform Load |
| **GSC** | Gold Standard Corpora |
| **GUI** | Graphical User Interface |
| **HMM** | Hidden Markov Model |
| **HTML** | HyperText Markup Language |
| **HTTP** | Hypertext Transfer Protocol |
| **HTTPS** | Hypertext Transfer Protocol Secure |
| **IE** | Information Extraction |
| **IR** | Information Retrieval |
| **ICD** | International Classification of Disease |
| **JSON** | JavaScript Object Notation |
| **JSP** | Java Server Pages |
| **LOINC** | Logical Observation Identifiers Names and Codes |
| **ML** | Machine Learning |
| **ME** | Maximum Entropy |
| **NER** | Named Entity Recognition |
| **NLP** | Natural Language Processing |
| **NPM** | Node Package Manager |
| **ORM** | Object-Relational Mapping |
| **OHDSI** | Observational Health Data Sciences and Informatics |
| **OMOP** | Observational Medical Outcomes Partnership |
| **POS** | Part-of-Speech |
| **PDF** | Portable Document Format |
| **PPI** | Protein-Protein Interaction |
| **PMC** | PubMed Central |
| **REST** | Representational State Transfer |
| **RDF** | Resource Description Framework |
| **SSL** | Secure Sockets Layer |
| **SSC** | Silver Standard Corpora |
| **SNOMED** | Standard Nomenclature of Medicine |
| **SQL** | Structured Query Language |
| **SVM** | Support Vector Machine |
| **TM** | Text Mining |
| **URL** | Uniform Resource Locator |
| **WHO** | World Health Organization |
| **WWW** | World Wide Web |
| **XHTML** | eXtensible HyperText Markup Language |
| **XML** | eXtensible Markup Language |

CHAPTER 1

# Introduction

## 1.1 Motivation

With the advancement of technology, it is imperative that it becomes a resource. Over the years, sophisticated IT solutions have been exponentially integrated into the most diverse areas such as industry, education, and healthcare with several goals in mind: be it automating or streamlining processes, in order to make them more efficient while reducing man labor, which is prone to error. In this dissertation, I am interested in contributing to a learning healthcare system, as many have argued for more than a decade. Every healthcare-related entity, academic or professional (e.g. academic medical centers, regulatory agencies and medical product manufacturers, insurance companies and policy centers, etc), face a common challenge that, when overcome, contributes to a learning healthcare system: "how do we apply what we've learned from the past to make better decisions for the future?". When a patient undergoes medical appointments and/or examinations, the captured data is stored in databases. As such, the motivation and ambition to analyze patient-level data arises to produce real-world evidence, which in turn could be disseminated across the healthcare system to inform clinical practice [1].

## 1.2 Goals

The focus of this dissertation is to provide a tool that allows biomedical experts to contribute for a richer literature that in turn makes it possible to conduct further research studies, which may lead to innovative breakthroughs regarding clinical sciences. The experts should be able to rely on the tool to perform automatically generated annotations over text while indicating their domain, as well as manually curate biomedical documents. Moreover, in order to harmonize data towards collaborative clinical research, it should allow them to map non-standard concepts to standard vocabulary concepts and validate and rectify mappings to ensure quality.

The tool must support three use cases:

- Biomedical experts need text mining services for document annotation and curation;
- Biomedical experts need to harmonize data into standard concepts;
- Biomedical experts need to execute both tasks.

## 1.3 Document Structure

The remainder of this document is structured as follows. Chapter 2 describes the importance of mapping medical concepts into standard vocabulary concepts to the biomedical research and presents a successful approach that revolutionized this domain. In addition, it explores the state-of-the-art concerning how Named Entity Recognition (NER) has been included as a major role in Natural Language Processing (NLP) tools and NLP techniques. Chapter 3 provides the requirements needed to build a solution that fulfils the goals listed above and its architecture. Chapter 4 gives a detailed description of the implementation behind the proposed solution, followed by a walkthrough of how to use it in Chapter 5. Finally, this document is concluded in Chapter 6 where are enumerated relevant points for future work.

CHAPTER 2

# State of the Art

## 2.1 Mapping Clinical Concepts

Given that patient-level data is captured at several healthcare institutions, it is possible to conduct observational studies which are useful to clinical research. However, since this data is quite disparate, it can not be used as it is directly because it hinders a collaborative effort between healthcare communities that can produce analytics useful for several use cases, such as population's statistics about treatments, demographic information, track disease natural history and so on. A successful approach to this problem was achieved by an open-science collaborative: Observational Health Data Sciences and Informatics (OHDSI). The main goal is to improve health by collaboratively generate real evidence that promotes better health decisions and better care. This section describes their challenges, their data standards and the tools developed to aid manual processes.

### 2.1.1 Collecting and processing patient-level data

Clinical research demands a high volume of data from patients, which is captured at medical appointments and/or examinations and is used to perform observational studies. Observational, prospective cohort studies, also called registries, evaluate the experience of a group of people who share a defining characteristic [2], i.e. a group of subjects that experienced a common event in a selected time period, such as a disease diagnosis, clinical milestone, or initiation of medical or surgical treatment. These studies rely on regular measurements of clinical and patient-reported outcomes, since they are useful in evaluating a breadth of data in a timely fashion that may help predicting and diagnosing patients earlier. For that purpose, observational databases have been used extensively in clinical research with the main goal of accumulate and document a large, heterogeneous patient experience over time that may be gathered by different practitioners [3]. The collection of this kind of information not only allows to keep records of the patients and track their health, but also to compare treatments and seek for common characteristics that may improve clinical research studies.

The OHDSI community named this process as "the journey from data to evidence" which focused on overcome a common challenge in healthcare: "how do we apply what we've learned from the past to make better decisions for the future?". In other words, what they try to accomplish is to transform patient-level data into real-world evidence for observational studies. Even though tremendous progress has been made, they claim to be still far from their goals. The reason why they fall short is due to the heterogeneity of observational databases. There are different types of observational databases that capture several kinds of patient-level data, such as populations (i.e. pediatric vs elderly), care settings that depend on the patients (inpatient vs outpatient, primary vs secondary care), data capture processes and health systems. On the other side of the spectrum, there is also different types of evidence that could be useful to inform decision-making, among them clinical characterization, population-level effect estimation and patient-level prediction. Moreover, due to the breadth of clinical, scientific and technical competencies required to trek from data to evidence, this is an arduous task for a single individual given that is very unlikely to possess such a handful of skills. Thus, the journey requires collaboration across multiple individuals and organizations to produce the best evidence possible. It is required a thorough understanding of health informatics, epidemiologic principals and statistical methods, implementation and execution of computationally-efficient data science algorithms and clinical knowledge [1].

### 2.1.2 Data harmonization

Now that we established that observational studies are a collaborative effort from several institutions with different skilled individuals or, if we think through a higher dimensional viewpoint, several countries, we realize that the clinical registries, be it observational databases or Eletronic Health Record (EHR)s, will store data in different formats and encoding. Plus, different countries will store patient-level data in different languages: in Portugal, a patient's sex is defined as *homem/mulher*, in Spain as *hombre/mujer*, in England as *man/woman*. This presents itself as a problem to clinical research. Due to the large array of data sources and considering that none of them capture all clinical events equally well, it is necessary to analyze those data sources concurrently . In order to do that, data need to be harmonized into a common data standard, which transforms concepts into standard vocabulary concepts that everyone can understand no matter what language, format or encoding it was originally recorded in. Otherwise, a Structured Query Language (SQL) query would have issues obtaining results and would not be possible to extend studies to other countries [1]. This is the reason why mapping medical concepts to standard definitions is so important. Subsection 2.1.3 is about a data harmonization solution developed by OHDSI.

### 2.1.3 Common Data Model

OHDSI paved the way for collaboration by adopting an open-science approach. At first, its purpose was to monitor drug safety in a public-private partnership between the US Food and Drug administration, the Foundation for the National Institutes of Health and a consortium of pharmaceutical companies. Soon the Observational Medical Outcomes Partnership (OMOP) Common Data Model (CDM) was designed to standardize observational data and establish an institutional collaboration environment that could accommodate several data types, unlocking a wide array of research areas and computationally-efficient analytics. Such array of research areas allow: 1) to identify groups of patients that share certain healthcare interventions and outcomes (i.e. drug exposures, conditions, procedures etc) and 2) generate population's statistics about treatments, disease natural history, costs, demographic information and more, for predicting outcomes in individual patients [1].

In order to achieve those goals, this CDM follows a number of design principles:

*Suitability for purpose*
The data is organized not for the purpose of healthcare providers or payers, but in a analysis-friendly way.

*Data protection*
Patients' personal information is sensible, therefore protected and not presented. Exceptions may occur for research purposes.

*Design of domains*
The domains are modeled in a person-centric relational data model. This means that clinical events are linked to a person allowing a detailed visualization of a person's clinical events.

*Rationale for domains*
Domains are identified and separately defined in an entity-relationship model if they have an analysis use case (conditions, for example) and the domain has specific attributes that are not otherwise applicable.

*Standardized vocabularies*
Source of standardized content, containing all necessary concepts.

*Reuse of existing vocabularies*
Existing vocabularies from organizations or initiatives are reused.

*Maintaining source codes*
Source codes are stored after the standardization process, in order to avoid information loss.

*Technology neutrality*
The CDM is not restricted to a specific relational database.

*Scalability*

The CDM is capable of processing different sized data sources.

*Backwards compatibility*
All versions of CDM are available in GitHub.

### 2.1.4 Standardized Vocabularies

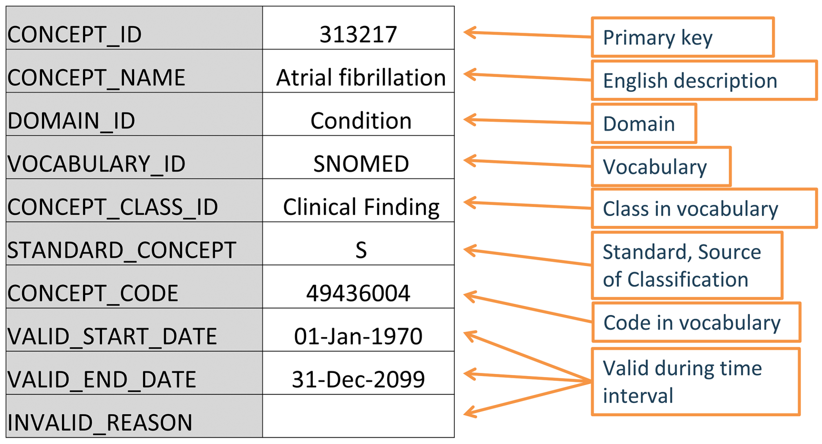
Vocabularies are the key for network research. They keep all sorts of information including terminologies, relations, hierarchies and ontologies spread across a variety of healthcare domains: Drug, Procedure, Condition etc. Each country or institution tends to have their own system, which they agree upon with the common goal of capturing and analyzing patient data. However, it exclusively suits their needs. One example is the World Health Organization (WHO)'s International Classification of Disease (ICD), which is the reference for country-specific versions, such as ICD10CM (USA) or ICD10GM (Germany). Hence, is not possible to work collaboratively if there are multiple data formats. Standardized Vocabularies are presented as the solution for patient data exchange and standardized research. Multiple standards have been created by organizations, such as Standard Nomenclature of Medicine (SNOMED), Logical Observation Identifiers Names and Codes (LOINC), RxNorm, etc [1].

Building and maintaining a standardized vocabulary is an arduous task due to its complexity and size. It must follow the same common format throughout its length to aid the researchers. This way, they do not encounter problems understanding and handling different formats. For this reason, it is infeasible for a healthcare community to build all standardized vocabularies from scratch, thus a common practice followed by OHDSI is to adopt existing vocabularies from dedicated organizations, such as those mentioned in the previous paragraph. Currently they support 111 vocabularies, of which 78 are adopted from external sources [1].

*Concepts*
The OMOP CDM represents all clinical events as Concepts and stores them in the CONCEPT table. Each concept is defined by the following fields:

- CONCEPT_ID
- CONCEPT_NAME
- DOMAIN_ID
- VOCABULARY_ID
- CONCEPT_CLASS_ID
- STANDARD_CONCEPT
- CONCEPT_CODE
- VALID_START_DATE
- VALID_END_DATE
- INVALID_REASON

Figure 2.1 shows an example of the standard representation of *Atrial fibrillation* following the OMOP CDM standards. The concept ID is used as primary key but is meaningless. On



| | |
|---|---|
| CONCEPT_ID | 313217 |
| CONCEPT_NAME | Atrial fibrillation |
| DOMAIN_ID | Condition |
| VOCABULARY_ID | SNOMED |
| CONCEPT_CLASS_ID | Clinical Finding |
| STANDARD_CONCEPT | S |
| CONCEPT_CODE | 49436004 |
| VALID_START_DATE | 01-Jan-1970 |
| VALID_END_DATE | 31-Dec-2099 |
| INVALID_REASON | |

**Figure 2.1:** Standard representation of Atrial fibrillation in the OMOP CDM [1]

the other hand, the concept code and the domain are what uniquely identify this concept in the specified vocabulary. For example, *Atrial fibrillation* is defined in several vocabularies: MESH, CIEL, SNOMED, ICD9CM and Read. However, only the SNOMED concept is standard and represents the condition in the data [1]. In this case, the standard concept field has value “S”. Only the standard concepts are used to represent clinical events in the CDM. Non-standard concepts or source concepts can be mapped to become standard. There are also classification concepts, which are not standard either and hence can not represent the data. The assignment of Standard, Source or Classification depends on the domain, which can be seen in table 2.1.

| **Domain** | **for Standard Concepts** | **for Source Concepts** | **for Classification Concepts** |
|---|---|---|---|
| Condition | SNOMED, ICDO3 | SNOMED Veterinary | MedDRA |
| Procedure | SNOMED, CPT4, HCPCS, ICD10PCS, ICD9Proc, OPCS4 | SNOMED Veterinary, HemOnc, NAACCR | None at this point |
| Measure | SNOMED, LOINC | SNOMED Veterinary, NAACCR, CPT4 HCPCS, OPCS4, PPI | None at this point |
| Drug | RxNorm, RxNorm Extension, CVX | HCPCS, CPT4, HemOnc, NAAACCR | ATC |
| Device | SNOMED | SNOMED & Others, currently not normalized | None at this point |
| Observation | SNOMED | Others | None at this point |
| Visit | CMS Place of Service, ABMT, NUCC | SNOMED, HCPCS, CPT4, UB04 | None at this point |

**Table 2.1:** Standard/Source/Classification concept assignments by domain [1]

*Domains*

The CDM has a total of 30 domains. The concept count distribution is illustrated in Figure 2.2:

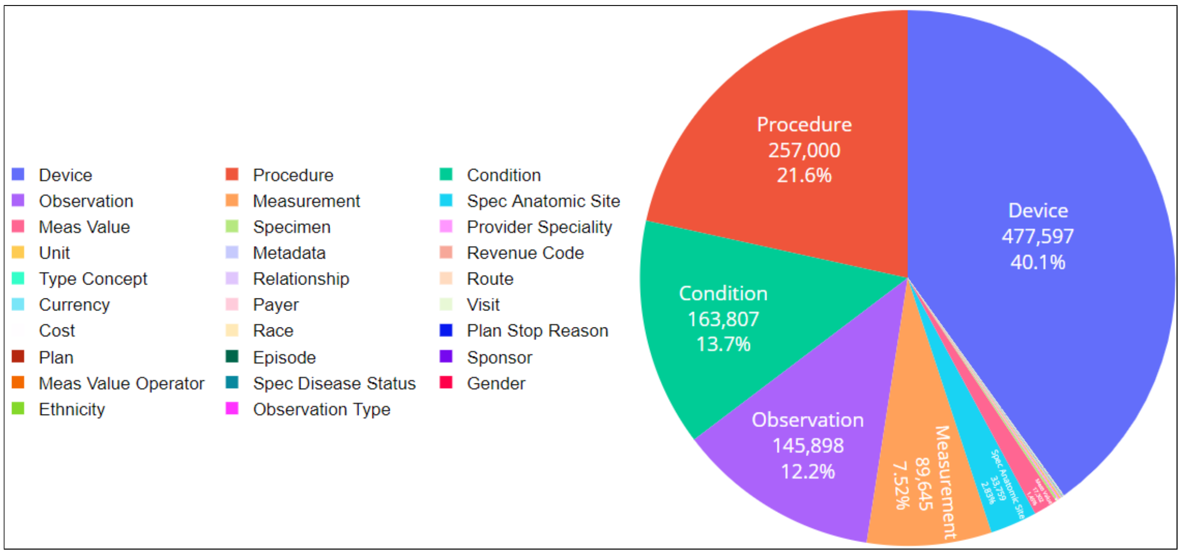


**Figure 2.2:** Concept count distribution by domain

### 2.1.5 Tools for Mapping Concepts

This section is dedicated to the process responsible for capturing observational data and converting it into the OMOP CDM. Moreover, is presented a mapping tool developed by the OHDSI community to automate the process and promote a collaborative effort.

*Extract Transform Load*

Extract Transform Load (ETL) processes perform a critical role in modern society as the amount of data sources and its generated data exponentially increases. They are responsible for extracting data from heterogeneous data sources, their cleansing and customization, their transformation to fit the businesses' schemes and their loading into a data warehouse [4]. Data warehousing first appeared in the late 80s as a concept intended for transferring data from services to decision support environments [5]. It is a collection of decision support technologies that allow intelligent businesses to make better and faster decisions. Since then, data warehousing technologies have been deployed on several industries, such as healthcare, manufacturing, financial services, telecommunications, etc [6]. A good example is in the retail space where the information gathered can be analysed to study sales flows, customer support and trends to improve their service, such as what and how much it is being sold so they can understand if any changes should be applied to meet higher competitive standards relatively to other retailers. This data is stored into a data warehouse, a non-volatile collection of data maintained separately from an organization's operational database. They store consolidated data from several sources over long periods of time, hence tend to be gigabytes to terabytes, or even petabytes, in size. Due to its proportions, querying through such amount of records is a very intensive process with mostly ad hoc, complex queries that can access millions of records and perform a handful of operations [6]. Traditionally, the refreshment of data warehouses had been performed in an off-line fashion, typically every 24 hours. However, this is not feasible to every decision support environment. Some businesses (e.g. stock markets) demand accurate and updated reports based on current data. Consequently, data warehouses evolved to “active” producers for their users, performing “near real time” transactions, improving from 24 hour updates to hourly updates [4] [7].

An ETL process is necessary to convert the raw captured data to the OMOP CDM. In other words, it adds new mappings to the Standardized Vocabularies and is totally automated. The creation of an ETL process is a difficult task, because it requires knowledge of both the source data and the desired output format. Its building can be divided in five steps: 1) understanding the source data, 2) go from source to target output format, 3) create code mappings, 4) implementing the ETL and 5) quality control and maintenance [1].

1) *Understanding the source data*:
The first step consists of scanning the source data, which can be in Comma-separated Values (CSV) files or databases, and get a detailed description of its structure. OHDSI has been using their own software, White Rabbit, which generates an excel report with a list of all tables, fields in each table, values from each field, data types, maximum length of the fields, the frequency of each value and more useful contents. These will be of great help for the next step.

2) *From source data to CDM tables*:
After scanning the source data and analyzing the White Rabbit report, a team can work collaboratively to decide how to connect it to the tables and columns of the CDM (see figure 2.3). Also, it is necessary to define the logic for standardizing the concepts. This is done with the help of Rabbit-In-a-Hat tools that comes with the White Rabbit software. For example, when converting patient data to the CDM Person table the logic is the following:

```
if $gender ==' M'$ then
    $GENDER\_CONCEPT\_ID = 8507$
else if $gender ==' F'$ then
    $GENDER\_CONCEPT\_ID = 8532$
else
    Drop row
end if
```

3) *Create code mappings*:
Before creating new code mappings, it is a good practice to check before-hand if the source codes to be mapped are already recorded as standard concepts. A SQL query is useful to do so. Otherwise, it is necessary to map the code, which can be executed using a tool to aid the process: **Usagi**.

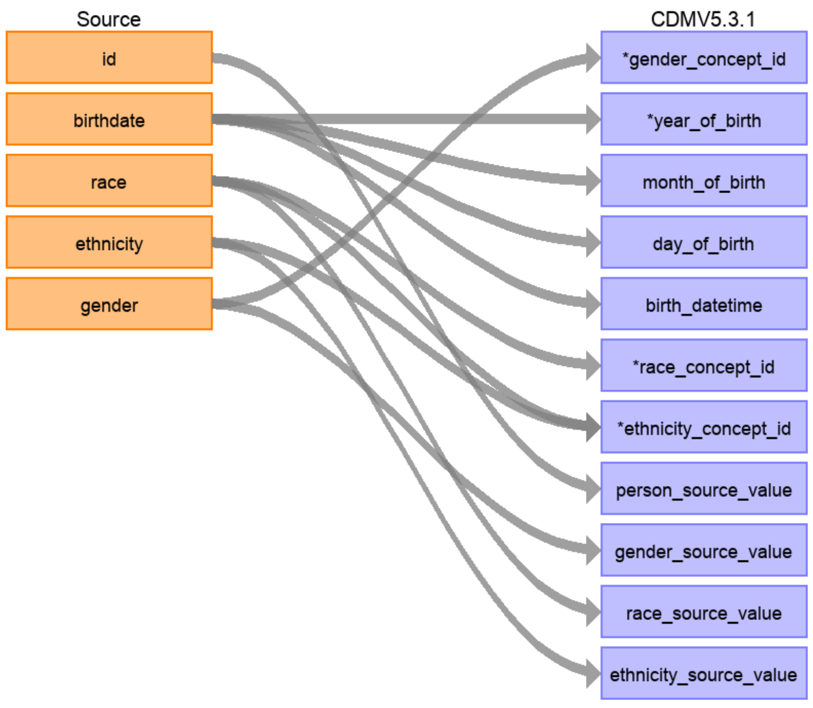


**Figure 2.3:** Mapping source patient data to CDM Person table [1]

4) *Implement the ETL*:
At this stage, technical individuals work alongside designers and knowledgeable people about the source and CDM with the common goal of implementing the logic necessary to accomplish the ETL correct behavior.

5) *Quality control and maintenance*:
Finally, during and after the ETL process creation, is crucial to ensure high standards of quality regarding every aspect: documentation, computer code, code mappings and testing. Also, being an extensive and complex procedure, the ETL requires constant maintenance.

*Usagi*

Usagi is a mapping tool developed by OHDSI to aid the manual process of creating code mappings based on concepts from different domains supported by the OMOP CDM. It allows loading source codes that need to be mapped to standard vocabulary concepts and automates the process by taking a term similarity approach. Some mappings may be wrong however, so Usagi offers an interface for human intervention for those situations, allowing a knowledgeable individual (not mandatory, but preferably) in the coding system and medical terminology to approve or unapprove mappings, replace or add target concepts, search for a more suitable concept, among other features. When the manual review meets the expectations, the user may export the code mappings into a CSV or Excel file [1].

### 2.2.1 Information Retrieval and Information Extraction

Every day new information is published in the form of research articles, books and reports, but given its unstructured form it is not useful for knowledge acquisition apart from keyword search. Therefore, since the development of the World Wide Web (WWW), significant interest has been generated towards the text mining problem and the production of useful structured data, which is then exploited in search, browsing and querying [8]. Text mining leverages computer power to efficiently and automatically extracting new information from different written resources [9]. Information Retrieval (IR) complemented by Information Extraction (IE) are techniques that allow to organize and structure raw data for research purposes. While IR techniques identify relevant documents from a larger collection based on a query, IE techniques, namely NER, process documents and seek to locate and classify pre-specified entities and relationships between them [10].

### 2.2.2 NER Approaches

The NER task is divided in two sub-tasks: 1) identification and 2) classification of entities. For a human, this is an intuitively natural and simple process, but for a machine it is very hard to solve. As of today, an overwhelming variety of named entity recognition systems have been developed with different approaches: rule-based, machine learning based and hybrid.

*Rule-based*

A rule-based approach is the most labor-intensive and exhausting of the three before mentioned. This type of system define a set of patterns to identify entities based on grammatical, syntactic and orthographic rules, such as nouns, verbs, locations, etc. Allied to the set of rules, typically they use dictionaries to include a wider span of terms resulting in better results. New nouns emerge as time passes, hence for this approach to work the dictionary needs to be manually updated. The best results may be achieved when used on restricted domains, where the named entities are well defined, unique and usually unambiguous [11].

*Machine Learning-based*

Machine learning approaches use algorithms that instead of identifying named entities following a rule set, classify them based on previous data and decisions. There are three types of machine learning models:

- **Supervised learning** is usually the chosen method and can use Support Vector Machine (SVM)s, Maximum Entropy (ME) models, Hidden Markov Model (HMM)s, decision trees, Conditional Random Field (CRF)s, etc [12]. However, it cannot achieve good performance without large training datasets [11].
- **Semi-supervised learning** is similar to supervised methods but use knowledge learned from a small training dataset.

- **Unsupervised learning** methods do not take feedback, thus are not very popular choices in NER systems. Generally, they are not domain specific unlike rube-based approaches [11].

*Hybrid*

Hybrid NER takes advantage of the strong features from the approaches above, combining rule-based human-made dictionaries and machine learning algorithms. Therefore, it experiences the same pitfalls that both have [11].

### 2.2.3 Biomedical NER

In the biomedical domain, the NER task is an essential step of IE. These systems are a valuable asset and their integration into larger biomedical IE pipelines is a key piece to accomplish other tasks. Whereas in domains such as newswire, typically the objective is to identify names of persons, locations, organizations, etc., the process becomes more challenging in the biomedical field. The reason for such is the added complexity introduced by the special naming conventions, as the authors of [13] report. Biomedical NER aims to recognize genes, proteins, conditions, disorders, among others, and relations between them (e.g. protein-protein). The identification of those entities is difficult due to several characteristics:

- Most entity names are 2 words long or more (e.g. “normal thymic epithelial cells”). The system must be capable of identifying those cases as one single entity instead of multiple.
- Some entities may share nouns, which can be split or joint. Example: “91 and 84 kDa proteins” consists of two entity names: “91 kDa proteins” and “84 kDa proteins”.
- Several spelling forms: “N-acetylcysteine”, “N-acetyl-cysteine”, and “NAcetylCysteine”.
- Abbreviations are frequently used in the biomedical domain and most times are irregular and/or ambiguous. “IL2” stands for “Interleukin 2”, “PAL” stands for “palate” and “TCF” may refer to “T cell Factor” or “Tissue Culture Fluid”.

Thus, Machine Learning (ML)-based approaches are often the selected method for this domain since they can adapt to the variety of entities, presenting several advantages over other methods and hence achieving better results.

### 2.2.4 Corpora

A critical part of ML-based NER systems is the dependency on annotated documents. Generally, these systems require two essential steps: train and annotate [14]. In order to be able to annotate a raw document, the ML model must learn how to perform that task. It trains with annotated documents typically related to the target domain but more structured than plain text, which are usually referred to as corpora. Building corpora is a very delicate process conducted by experts, since errors in the training corpus propagate to the final system [15].

There are two kinds of corpora:

*Gold Standard Corpora*

Gold Standard Corpora (GSC) are manually annotated collections of text belonging to a specific domain. It is a very laborious and time-consuming process conducted by experts to ensure that ML models train with the richest structured data possible. Its construction is a collaborative task because each expert annotates the same text independently, but in the end they compute an inter-annotator agreement ensuring the consensus of everyone and, most important, high quality corpora [15].

*Silver Standard Corpora*

On the other hand, since the construction of a GSC is a very costly process, in order to minimize costs some opt to attempt to determine the optimal size of a corpus (measured by the number of sentences) and automatically generate Silver Standard Corpora (SSC) to substitute gold standard [15]. Evidently, the quality compared to a gold standard is inferior. Nonetheless, it may be used in some situations.

Just as corpora, NLP tasks also depend on domain and intended results, thus each domain and task require a high quality and reasonable size corpora. Therefore, we quickly realize that a proper GSC does not exist for many NLP tasks [15]. Table 2.2 shows a list of GSC for each biomedical entity.

| **Entity** | **Corpus** | **Type** | **Size (in sentences)** |
|---|---|---|---|
| **Gene and Protein** | GENETAG | Sentences | 20000 |
| | JNLPBA (from GENIA) | Abstracts | 22402 |
| | FSUPRGE | Abstracts | 29447 |
| | PennBioIE | Abstracts | 22877 |
| **Species** | OrganismTagger Corpus | Full texts | 9863 |
| | Linnaeus Corpus | Full texts | 19491 |
| **Disorders** | SCAI Disease | Abstracts | 3640 |
| | EBI Disease | Sentences | 600 |
| | Arizona Disease (AZDC) | Sentences | 2500 |
| | BioText | Abstracts | 3655 |
| **Chemical** | SCAI IUPAC | Sentences | 20300 |
| | SCAI General | Sentences | 914 |
| **Anatomy** | AnEM | Sentences | 4700 |
| **Miscellaneous** | CellFinder | Full texts | 2100 |

**Table 2.2:** List of relevant Gold Standard Corpora (GSC) available for each biomedical entity, presenting the type of documents and its size [14].

## 2.3 NLP Techniques

The NLP phase is an utmost pre-processing stage in most NLP solutions. It is conducted before any NER techniques because it is infeasible for the recognition methods to process raw input data from the documents. In order to simplify the recognition of entities, there are some NLP techniques that can be applied, such as sentence splitting, tokenization, stop word removal, stemming, lemmatization, chunking and POS tagging.

### 2.3.1 Sentence Splitting

Typically, the first step of pre-processing is sentence splitting. As the name indicates, this task divides a text document into sentences by punctuation marks like periods, commas, semicolons and so on.

### 2.3.2 Tokenization

After splitting the text into sentences, an additional step named tokenization takes place consisting in further breaking. It is important to note that tokenization is not restricted to words, i.e. if we consider a token to be a phrase then it is applied to a full-text instead of split sentences. For consistency purposes, let us define a token as a word. This time, the goal is to split the sentences into words. In english, usually a whitespace regex rule is applied. In other words, the sentences are split by spaces [16].

### 2.3.3 Stop Word Removal

This technique is usually performed along with tokenization. Romance languages, i.e. Latin languages such as English, Spanish, Italian, Portuguese, French, etc. [17], use a large number of non-informative words such as articles, prepositions and conjunctions, called stop words [18]. There are different approaches for dealing with stop words: one can loop over the tokens and exclude those that are shorter than a number of characters, say three or four, for example. Although, usually a stop-list is built with words that should be removed, which can be prepositions like “and”, “or”, “the”, but also language and task dependent words [18]. This technique is very useful for dimension reduction allowing for better performance.

### 2.3.4 Stemming and Lemmatization

Another commonly used methods are stemming and lemmatization. A given token can have several morphological variants, which in turn may be equivalent. For instance, “computes”, “computing” and “computer” derive from the same term, thus can be analyzed as a single item. The basic idea of stemming is to map these variations to a stem, hence in the previous example they are mapped to “comput”. On the other hand, lemmatization is a more robust method because it returns the base or dictionary form of a word (e.g. the lemma of “was” is “be” and of “industrialized” is “industry” or “industrious”) [18] [14]. Both stemming and lemmatization increase relevance and recall capabilities of a retrieval system [19].

### 2.3.5 POS Tagging and Parsing

Part-of-Speech (POS) tagging essentially associates each token with a particular grammatical category, namely noun, verb, participle, article, pronoun, preposition, adverb and conjunction and assigns them the respective tag [20] providing lexical information. On the other hand, parsing produces syntactic information represented in a tree, providing richer structured information [16] (see figure 2.4).

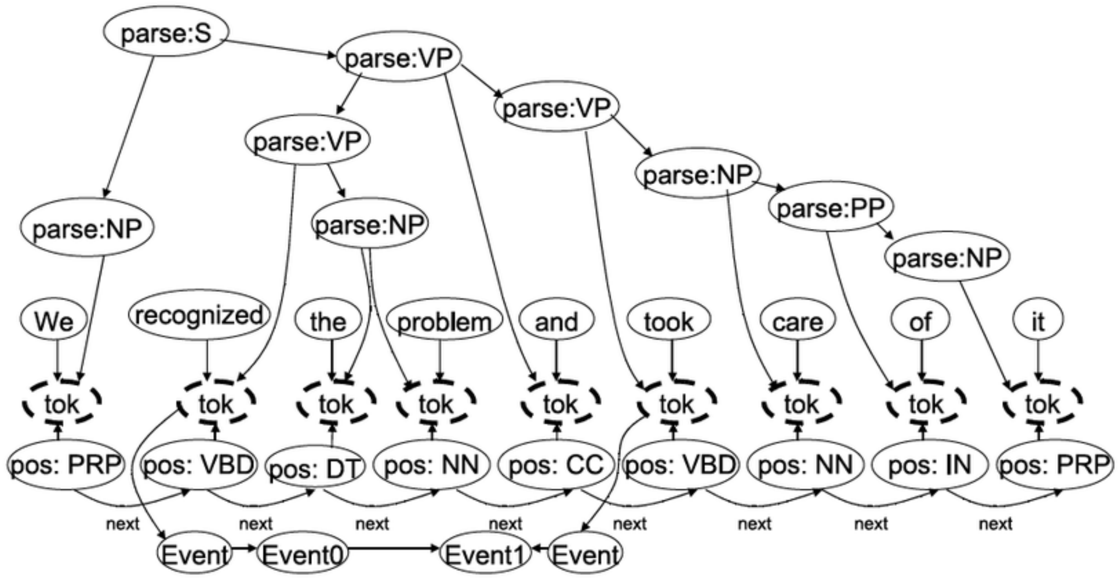


**Figure 2.4:** An example text graph encoding syntactic information [21]

### 2.3.6 Toolkits for NLP

In order to perform the above-mentioned techniques, several toolkits are commonly used. An overview can be seen in table 2.3.

| Toolkit | Language | Description |
|---|---|---|
| **NLTK** [22] | Python | Natural Language Toolkit (NLTK) is an open source platform for performing NLP tasks including tokenization, stemming, POS tagging, parsing, and semantic reasoning. It provides interfaces for many corpora and lexicons which are useful for opinion mining and sentiment analysis. http://www.nltk.org/ |
| **OpenNLP** | Java | The Apache OpenNLP is a JAVA library for the processing of natural language texts, which supports common tasks including tokenization, sentence segmentation, POS tagging, named entity recognition, parsing, and coreference resolution. https://opennlp.apache.org |
| **CoreNLP** [23] | Java | Stanford CoreNLP is a framework which supports not only basic NLP task, such as POS tagging, named entity recognization, parsing, coreference resolution, but also advanced sentiment analysis [24]. http://stanfordnlp.github.io/CoreNLP/ |
| **Gensim** [25] | Python | Gensim is an open source library for topic modeling which includes online Latent Semantic Analysis (LSA), Latent Dirichlet Allocation (LDA), Random Projection, Hierarchical Dirichlet Process and word2vec. All implemented algorithms support large scale corpora. LSA and LDA have distributed parallel versions. http://radimrehurek.com/gensim/ |
| **FudanNLP** [26] | Java | FudanNLP is an open source toolkit for Chinese NLP, which supports word segmentation, POS tagging, named entity recognition, dependency parsing, coreference resolution and so on. https://code.google.com/archive/p/fudannlp/ |
| **LTP** [27] | C++/Python | The Language Technology Platform (LTP) is an open source NLP system for Chinese, including lexical analysis (word segmentation, POS tagging, named entity recognition), syntactic parsing and semantic parsing (word sense disambiguation, semantic role labeling) modules. http://www.ltp-cloud.com/intro/en/ |
| **NiuParser** [28] | C++ | NiuParser is a Chinese Syntactic and Semantic Analysis Toolkit, which supports word segmentation, POS tagging, named entity recognition, constituent parsing, dependency parsing and semantic role labeling. http://www.niuparser.com/index.en.html |

**Table 2.3:** Available toolkits for NLP [16]

## 2.4 NLP Tools for Clinical Text

Bearing in mind that the creation of GSC is a very time-consuming task, it makes sense to support it by computational tools capable of a set of features [29]:

- Graphical interfaces for highlighting and tagging texts
- Management of text collections
- Assignment of texts to annotators
- Definition of annotation schemes
- Export results into various formats

As of today, multiple annotation tools have been developed to aid the creation of GSC, most being domain or task specific. This section is dedicated to explore and compare NLP tools developed, mostly, for annotating biomedical entities. At first, the research for these tools was only restricted to the annotation task, i.e. every annotation tool that could be useful for annotating texts, independently of their domain or task, would be of interest. For this purpose, [30] was used as a starting point for studying existing tools. Recently, the authors published an extensive review [31] following a set of criteria and comparing these tools. Essentially, [30] is a GitHub repository with an extensive list of annotation tools and also a web application [32] that allows to search through the list and filter those that fulfill specific criteria defined by the user, such as:

- Data format regarding annotations, documents and schema
- Functional aspects (e.g. highlighting, inter-annotator agreement, pre-annotations, ontologies, etc.)
- Type of instalation (i.e. desktop, web-based, plugin), if is available and workable
- Technical aspects (e.g. free or not, difficulty when installing, etc.)

However, we focused on tools capable of annotating biomedical documents semi or fully automatically or that are collaborative in some way. For this reason, from an initial list of 74 tools, only 15 fulfill one or both requirements.

### 2.4.1 Selected tools

The selected tools are capable of annotating biomedical documents, although some are general-purpose, and perform it in a semi-automated or automated fashion taking advantage of pre-processing NLP techniques (see section 2.3) and NER approaches (see subsection 2.2.2). The type of installation can be of three types: desktop, web-based or plugin. Concerning input and output there are variations, as some accept text, Portable Document Format (PDF), BioC, HyperText Markup Language (HTML), eXtensible HyperText Markup Language (XHTML), eXtensible Markup Language (XML) or Database (DB) formats and, as output, can generate the previous mentioned plus CSV, Resource Description Framework (RDF) Annotation Ontology (AO), A1, JavaScript Object Notation (JSON) and Conference on Natural Language Learning (CoNLL) [33]. In total, we identified 15 tools matching these criteria:

*@Note*

@Note [34] is a desktop Biomedical Text Mining (BioTM) platform developed to support three different roles: 1) biologists can use it for biomedical document retrieval, annotation and curation; 2) text miners may analyse biological text relying on text engineering techniques and 3) software developers contribute to BioTM research by including new modules and algorithms. Takes abstracts and full-texts in TXT and PDF formats as input, allows for PubMed search and journal crawling and converts PDF to text. @Note outputs the results in XML format.

*Argo*

Argo [35] is a web-based collaborative workbench with a Graphical User Interface (GUI) to ease manual and automatic annotations, and boost productivity. Two annotators work individually and their job is to verify the annotations' correctness. Also, they can identify interactions between words or phrases.

*BioNotate*

BioNotate [36] is a web-based annotation tool focused on extracting relationships between biomedical entities, such as interactions between genes and proteins or associations between genes and diseases. It was developed with the objective of representing a benchmark dataset built through disease studies data, i.e. a distributed creation of a large corpus.

*BioQRator*

BioQRator [37] is a general-purpose web-based user interface built for assisting manual literature curation. It is used for annotation of biomedical entities and Protein-Protein Interaction (PPI)s, capable of handling documents created by users or by querying the PubMed search engine, creating entities and relation types, and managing collections of documents. BioQRator also focuses on a document ranking task which has proven to be more efficient than PubMed's, when retrieving PPI information such as protein names and general topic queries.

*Brat*

Brat [38] is one of the most popular web-based general-purpose annotation tools. Supports high-quality annotation visualisation that helps users to understand complex annotations from different semantic types or sometimes overlapping text, as well as connections between annotations. Moreover, their intuitive annotation interface allows for annotation editing with mouse events (e.g. clicking or dragging). Given that it is general-purpose, users may accomplish most text annotation tasks with fully configurable options.

*Djangology*

Djangology [39] is also a web-based application for highly distributed annotation projects involving many participants. Thus, this tool has a project-oriented vision, in the sense that project administrators are in charge of uploading data (i.e. documents), defining schemas, assigning annotators to specific projects and monitor their progress. Finally, they are able to review inter-annotator agreement statistics. It has been used in the biomedical domain, namely in medical studies of trauma, shock and sepsis conditions.

*Domeo*

Domeo [40] is a web-based software framework for online semantic annotation of HTML, XHTML and XML documents. It takes advantage of using web services that provide annotation resources such as vocabularies and entity recognition services. Thus, it becomes a very flexible tool for different domains and can adapt to technology improvements over time.

*Egas*

Egas [41] is a web-based platform for BioTM and assisted curation that allows automatic in-line annotations of both biomedical entities and its relations. It is built on a project-oriented vision similar to Djangology, so annotators can only work in projects they have been assigned to. The concept recognition task is made through Representational State Transfer (REST) web-services, which can be implemented as long as they abide the input and output formats required: text and A1 or BioC, respectively. In turn, given that none existing PPI services available were fast enough for real-time usage, the authors report to have built their own service on top of other existing solutions. Finally, a key feature presented by Egas is real-time collaboration between annotators, enabling detail discussion.

*ezTag*

ezTag [42] is a web-based annotation tool for biological concepts. By standardizing text into BioC format, it supports PubMed abstracts and PubMed Central (PMC) articles. Like other tools aforementioned, allows for customized tagging modules from RESTful Application Programming Interface (API)s for annotating bio-entities. Offers multiple ways of annotation, among them pre-trained taggers, such as TaggerOne [43], GNormPlus [44] and tmVar [45], a dictionary-based tagger and customized taggers.

*GATE Teamware*

GATE Teamware [46] is a web-based, collaborative annotation framework that allows complex projects with distributed annotator teams. It is based on GATE [47], a very well known platform used for NLP tasks. This new version is meant to allow non-expert annotators to work on curation projects as it pre-annotates entities, which contributes to their training and involvement. For this reason, GATE Teamware is yet another tool with roles, i.e. there are project managers or administrators and annotators.

*MyMiner*

MyMiner [48] is a web-based tool designed for expert biologists without programming expertise, providing features such as document classification, classification efficiencies comparison, annotation of entities and binary relationships. Another interesting feature is the possibility to link articles to entities. For instance, for each gene or protein name, MyMiner suggests a ranked list of UniProt identifiers. UniProt is a free database of protein sequence and functional information derived from the research literature [49]. Thus, it provides a short description to help annotators on manual curation.

*OntoGene*

OntoGene [50] is a Text Mining (TM) system specialized in annotating biomedical entities and relationships between them. A web-based platform named OntoGene Document INspector (ODIN) was developed for assisted curation while also providing RESTful web services so other works can implement their system's text mining components. It accepts XML files based on the BioC specifications and the same format is applied at the output. In case an annotator wishes to export the results, he may choose between CSV or RDF.

*PubTator*

PubTator [51] is a web-based application developed for assisting biocurators who have limited TM experience. It allows to search for articles either by search queries or by a list of PubMed articles input by its users. Support for automatic annotation is achieved using a hybrid approach, i.e. for annotating genes, diseases, species and mutations they use ML-based entity recognition tools (GeneTUKit [52] and GenNorm [53] for gene mention and normalization, SR4GN [54] for species, DNorm [55] for diseases and tmVar [45] for mutations) and for chemicals a dictionary-based approach [56]. Apart from automatic and manual bio-entities annotation, PubTator provides document triage and relationship annotation such as PPIs or others.

*TeamTat*

TeamTat [57] is a web-based tool for wide annotation projects, due to its role architecture similar to some previously described tools. It was developed by the same team from ezTag, and provides features that were not found on it, mainly a collaborative environment for concurrent annotation. Moreover, supports full-text along with figures, integration with PubMed and PMC through BioC format and quality assessment performed by project administrators. TeamTat allows for configuring annotation schemas for entities and relations, thus is capable of a wide range of annotation tasks.

*Neji*

Neji [58] is an open source framework highly specialized for biomedical concept recognition, through a hybrid approach, i.e. combines dictionary matching and machine learning methods to achieve the best results possible. It was built on top of four ideas: 1) modularity (i.e. independent modules perform unique processing tasks), 2) allows for dictionary and ML models configuration which makes it scalable, 3) provides fast and multi-threaded data processing and 4) is developer-friendly in the sense that developers and researchers are capable of easily implement new modules or using pre-defined pipelines. It supports overlapped concept names and disambiguation techniques due to a concept tree implementation. Recently, a web services branch was developed so it is possible to take advantage of its RESTful API to annotate either text files or PubMed articles and export the results into well-known output formats (e.g. A1, BioC, CoNLL, JSON, XML, etc.).

### 2.4.2 Other tools

Manual annotation tools such as ANALEC [59], Anotatornia [60], Glozz [61], KAFnotator [62] and SALTO [63] were excluded. Nevertheless, they present features that the selected tools may not have. For instance, Glozz can represent annotations over the text and annotations as a graph, which can prove to be very useful. Some tools were not considered for being designed for specific tasks or different domains other than the biomedical. Examples of those are AWOCATo [64], designed for a sentiment annotation task; EasyRef [65] performs syntactic annotation by splitting sentences and identifying subjects, verbs, nouns, complements, etc; EULIA [66] provides an environment for consulting, visualizing and modifying documents processed by other tools; GraPAT [67] builds graphs over text like Glozz and its main use case is sentiment analysis; PDFAnno [68], as the name suggests, is an annotation tool for PDF documents.

CHAPTER 3

# Architecture

This chapter provides an overview of the challenges of developing a state-of-the-art tool. Section 3.1 starts by addressing the problem statement. Next, in section 3.2 we define the requirements necessary to ensure that the solution developed fulfills all the expectations and finally we propose a solution along with an overview of its architecture in section 3.3.

## 3.1 Problem statement

After reviewing the state of the art concerning the mapping and annotation of medical concepts, we understand that there are several tools that fulfill the requirements to perform each task. However, there is a lack of tools that implement both functionalities simultaneously. Usually, an expert's *modus operandi* takes, at least, six steps from start to finish: 1) automatically annotate a document using an NLP tool; 2) perform manual curation in order to fix errors; 3) export annotated results; 4) import those into a mapping tool; 5) review and validate mappings and, finally, 6) export mapped results. Therefore, this process sometimes may become tedious and more complex given that an user must learn how to use and possibly configure two tools in order to complete the full pipeline, starting at a document and ending with standard definitions.

For this reason, we believe that a complete tool, providing it is able to satisfy both ends of the spectrum, would be a great asset for biomedical experts. The next section presents the requirements necessary to create a state-of-the-art tool that fulfills them.

## 3.2 Requirements

### 3.2.1 Project Management

In chapter 2, the term "*collaborative*" is present repeatedly because it is a crucial aspect in the biomedical domain. On one hand, the task of mapping medical concepts requires collaboration between healthcare communities that can produce analytics useful for different use cases, which in turn promote better health decisions and better care. Also, it is important to harmonize health data into standard vocabularies that will allow to conduct broad studies involving multiple countries and organizations worldwide. On the other hand, NLP tools demand rich structured data so that the ML models learn how to annotate biomedical documents. This can only be done by providing a collaborative environment giving the ability for experts to combine efforts and share knowledge between each other.

Therefore, the tool should be built based on the idea of projects. A project should have at least a name and description at creation, as a way of identifying what task it is focused on. Concerning the project's structure, it consists of a team of experts that will work on one or multiple documents. The individual who creates the project is nominated project manager while the rest of the members, which should be invited, perform the role of annotators. Despite the project managers' skills, he/she should be able to contribute by annotating documents and/or mapping terms. If possible, develop a review system where project managers evaluate the project's progress, analyse inter-annotator agreement and other kind of statistics. In regard to annotation and mapping documents, the application must support file uploads from local storage, restricting its format and size.

### 3.2.2 Annotation Interface

The user interface dedicated for annotation should be user-friendly, simple and easy to use. Annotation projects usually deal with very extensive documents that despite being oriented to a single domain, may have a high number of entity types. In the biomedical domain, that number is specially large (e.g. disease, anatomy, species, cellular component, etc.). Thus, the user should be able to distinguish and identify entity types once they are annotated. State-of-the-art NLP tools such as those described in section 2.4 use highlighting techniques to facilitate the annotators' work.

As a state-of-the-art tool, automatic pre-annotation is a sought-after feature by experts as it spares them from manual annotation, which is a labor-expensive and time-consuming task. Consequently, manual curation is necessary due to the NER methods' limitations, i.e. ambiguity and unrecognized entities. Moreover, some kind of information should be made available to the user about a given annotation. Several NLP tools are also capable of annotating relationships between entities.

### 3.2.3 Mapping Interface

On the other side of the spectrum, a mapping user interface is required. It should be able to work with two data sources: 1) the resulting annotated terms previously worked on by the annotators, thus following a complete pipeline starting with a biomedical document and finishing with terms mapped to standard definitions and 2) terms that were uploaded by the user. In this scenario, the user previously annotated a document and only needs to map the terms, hence skipping the annotation stage. Allowing this behaviour implies that the tool must be in some way modular.

### 3.2.4 Database

Finally, the database is where all the project-related information will be stored.

## 3.3 Proposed solution

Taking into consideration the requirements described above, we propose a web-based annotation and mapping tool, based on the idea of projects. As a starting point, we followed a simple tutorial [1] from bezkoder on how to build a Create, Read, Update and Delete (CRUD) app with **React.js** [2], **Node.js** [3], **Express.js** [4] and **PostgreSQL** [5]. Figure 3.1 shows an overview of the system's architecture.

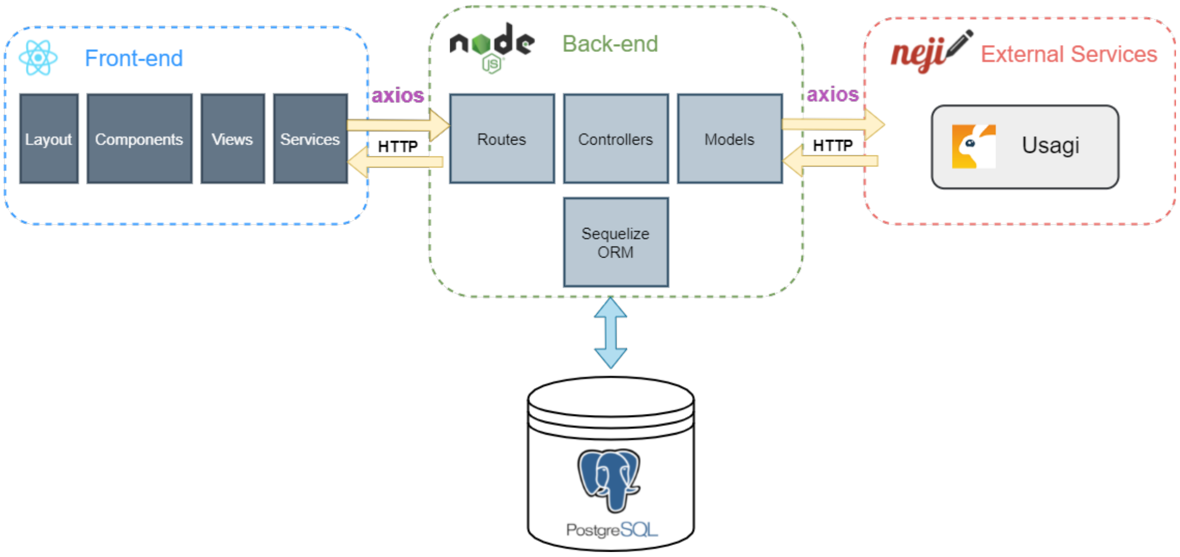


**Figure 3.1:** Overview of the system's architecture (adapted from bezkoder)

*Front-end*

The front-end is an user interface that will interact with the users while displaying all the information needed. We opted to develop this component using React.js, a well known and widely adopted JavaScript library for building user interfaces that benefits from a component-based architecture. In addition, it is a very efficient and light-weight library.

*Back-end*

The back-end is essentially the server-side of the application that handles the requests triggered by users' actions and returns information from the database or other services. It is based on Node.js, which is a JavaScript runtime together with Express.js, a framework that provides a set of features for web applications, which is responsible for configuring the server and defining endpoints.

*Database*

The database itself is responsible for storing important information including projects, members, documents and annotations and is a PostgreSQL relational database.

[1]https://www.bezkoder.com/react-node-express-postgresql/
[2]https://reactjs.org
[3]https://nodejs.org/en/
[4]https://expressjs.com
[5]https://www.postgresql.org

CHAPTER 4

# Implementation

This chapter describes the development that was undertaken with the purpose of implementing the proposed solution described in chapter 3, taking into consideration the requirements outlined. Section 4.1 starts by analysing database-related requirements, such as identifying relevant entities that will play a part in the overall process, understand the relationships between them and what they consist of. Section 4.2 describes in depth how the back-end was built in order to achieve the desired functionality, as well as how some challenges were overcome. Section 4.3 defines the front-end structure, in the sense that it provides an overview of what users might expect from the user interface.

## 4.1 Database Planning

Before we take any step towards the implementation itself, it is a good practice to draft a plan of what will be required. This section describes the database planning which usually takes at least three steps: requirement analysis, construction of an entity-relationship model and administration of the database. These steps are further described in the next subsections.

### 4.1.1 Requirement Analysis

The first step of planning is figuring out all the requirements that the tool demands, in other words identify entities and their attributes. As mentioned in section 3.3, our solution is based on the idea of projects. Therefore, a project is the main entity, so let us start by describing it. A **project** is identified in the database by an unique number that will match the order of creation, i.e. the first project will be number one, the second is number two and so forth. In order to distinguish projects and to promote a user-friendly interface, each project should have a name and description. At the time of its creation, the user who created the project is automatically its manager and the project's status becomes "Open". A **member** is identified by a first name and last name, but throughout the interface is presented to others by a username. Also, for registration purposes a member may also define an email and password. The database must store documents for both tasks (i.e. annotation and mapping) uploaded by users. A **document** is identified by a number which is auto-incremented following every upload, similar to projects, and a title that is the name of the document itself, including its extension. Additionally, there are four fields that will be better described later in this document, namely location, mapping, annotated and rows. Finally, annotations performed on documents must be stored as well. An **annotation** consists of a meaningless auto-incremental identification number, an annotation ID that follows a specific format hence stored as a string, the actual term that was annotated, its offset representing the starting position in the document and a biomedical type, for instance anatomy, disease, etc.

### 4.1.2 Entity-Relationship Model

With all the entities and their attributes identified, the next step is to build an entity-relationship model depicting an image of how the entities are connected, what is their relationship, key attributes and cardinalities. This step is very important because it allows for better understanding of how we need to configure the database tables, but most importantly it provides hidden information that otherwise we may not recognize. The attributes that we identified for each entity do not represent the complete tables in the database, as we are going to witness next.

First of all, for each entity we must define a key attribute that will represent the entity's primary key in the database. It must be an unique property that defines the entity, for instance for the entity Project we have chosen the project's ID, for Member its username, for Document the document's ID and for Annotation the internal ID.

Another aspect to take into consideration is the relationships. How are members and documents related to a project? How are annotations related to a document? The degree of relationships, also known as cardinality, will be represented using Chen Notation which uses the characters "1", "N" or "M" and can be of three types: one-to-one (1:1), one-to-many (1:N) or many-to-many (N:M). When users perform annotations on a document there is a "one-to-many" kind of relation, i.e. one document has multiple annotations associated to it. Furthermore, projects are task oriented and very extensive, thus a single project can support a large number of documents, thus forming a one-to-many relationship. Finally, a project is made up of a team of experts and an expert is allowed to work on multiple projects, implying a many-to-many relationship between project and member.

At this point we are able to draw a diagram representing the entity-relationship model. Figure 4.1 presents the entities as green rectangles, attributes as blue oval shapes and relationships as pink rhombus (diamonds). Key attributes are distinguished from the others by having their text underlined and a thicker outline, whereas relationships have two types: strong relationships are single rhombus and are used when an entity is existence-independent from the other, which is the case of "Participates" between Member and Project. However, this is not the case with the remaining relationships between Project and Document and Document and Annotation. Those are weak relationships represented by double rhombus, because a document does not exist (in the database context) without the existence of a project and the same happens with an annotation belonging to a document.

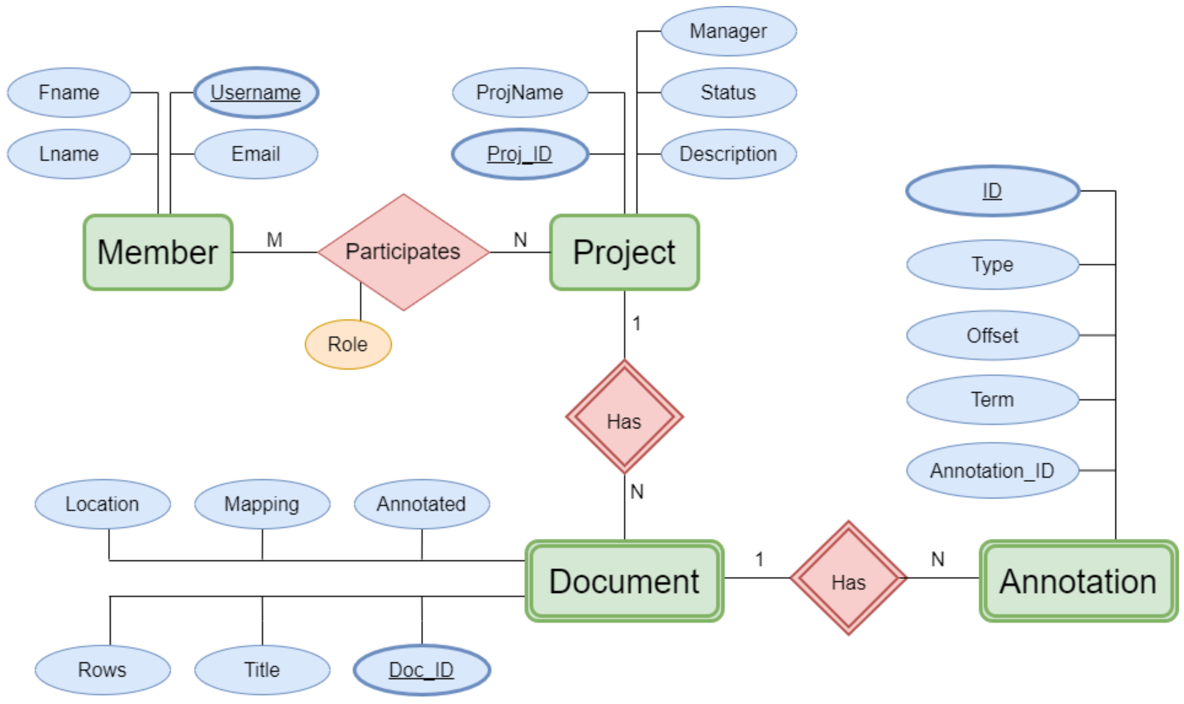


**Figure 4.1:** Entity-Relationship Model

### 4.1.3 Database Tables

The last step is to convert the entity-relationship model into tables that later will be defined in the database to store the information. Each entity represents a table and starts with all its attributes. However, we must pay attention and follow some rules in order to properly configure the database. For instance, due to the dependency of Document concerning Project, the primary key from the stronger entity becomes a foreign key of the weaker one. Consequently, "Proj_ID" will also belong to the Document table as a foreign key, thus allowing to associate documents to projects. Likewise, the Annotation table defines "Doc_ID" (Document's primary key) as its foreign key. This mechanism allows to execute SQL search queries that return a set of all annotations recorded for a given document, for example. Table 4.1 gives an overview of each entity, their attributes and a short description.

| Entity | Attribute | Type | Description |
|---|---|---|---|
| **Project** | Proj_ID | Integer | Project ID |
| | Title | String | Title of the project |
| | Description | String | Brief description of the project |
| | Manager | String | Member assigned as project manager. FK referencing Project's "username" |
| | Status | String | Indicates if an project is open or closed |
| **Member** | First Name | String | First name |
| | Last Name | String | Last name |
| | Username | String | Username used across the interface |
| | Email | String | Email |
| | Password | String | Password |
| **Document** | Doc_ID | Integer | Document identification number |
| | Title | String | Name of the document |
| | Location | String | Absolute path where the document is located |
| | Annotated | Boolean | Flag representing the document's annotation status |
| | Mapping | Boolean | Flag to differentiate mapping documents from annotation documents |
| | Rows | Integer | Number of rows, in case the document is for mapping |
| | Proj_ID | Integer | FK referencing Project's project ID |
| **Annotation** | ID | Integer | Internal ID |
| | Annotation_ID | String | Specific format returned by annotation tool |
| | Term | String | The term that was annotated |
| | Offset | Integer | Starting position of the term relative to the document |
| | Type | String | Biomedical entity type |
| | Doc_ID | Integer | FK referencing Document's document ID |

**Table 4.1:** Overview of each entity and their attributes

## 4.2 Back-end

The back-end consists of a Node.js Express server with the main purpose of establishing a connection between the front-end and the database. As such, it acts as an intermediate responsible of processing user-triggered requests, fetching the requested data from the database and returning it back. That is essentially the big picture that depicts the role of a back-end server. For a better understanding, Express.js is a Node.js framework that simplifies the development of REST APIs when compared to plain Node.js, which in turn requires to parse payloads, cookies, storing sessions and routes based on regular expressions. For that reason, Express allowed for a much easier and faster development of the server and it also interacts with PostgreSQL as we will explain shortly. Figure 4.2 represents the back-end project structure and as we can see it is divided into four main components: **Configuration**, **Controllers**, **Models** and **Routes**. The subsections from 4.2.1 to 4.2.5 further describe their roles.

```
nodejs-server/
├── App/
│   ├── Config/
│   │   └── db.config.js
│   ├── Controllers/
│   │   ├── annotation.controller.js
│   │   ├── document.controller.js
│   │   ├── member.controller.js
│   │   └── project.controller.js
│   ├── Models/
│   │   ├── annotation.model.js
│   │   ├── document.model.js
│   │   ├── member.model.js
│   │   ├── project.model.js
│   │   └── index.js
│   └── Routes/
│       ├── annotation.routes.js
│       ├── document.routes.js
│       ├── member.routes.js
│       └── project.routes.js
└── server.js
```

**Figure 4.2:** Back-end project directory tree

### 4.2.1 Server Configuration

Starting with the server configuration, this is what will allow to perform CRUD operations. The *server.js* file is where the Express web server is setup, starting by importing three modules: Express for building the API, *body-parser* to parse requests and *cors* which acts as a middleware to enable CORS with various options. In addition, it is also where the endpoints from the Routes component and the Models will be instantiated.

### 4.2.2 PostgreSQL Database & Sequelize

The file *db.config.js* is a simple configuration file for the PostgreSQL database, which defines five parameters for PostgreSQL connection and one parameter for Sequelize [1]. It is important to note that for this configuration to work, one must create beforehand a PostgreSQL server and the database. In this case, we have used pgAdmin [2] to create them, which is an administration and development platform that provides useful statistics and features. The configuration is shown in figure 4.3:

- **Host**: IP address (e.g. localhost)
- **User**: username defined at creation
- **Password**: password defined at creation
- **DB**: the database to connect to
- **Dialect**: in this case is postgres

[1]https://sequelize.org
[2]https://www.pgadmin.org

```
module.exports = {
    HOST: "localhost",
    USER: "postgres",
    PASSWORD: "neji",
    DB: "nejidb",
    dialect: "postgres",
    pool: {
        max: 5,
        min: 0,
        acquire: 30000,
        idle: 10000
    }
};
```

**Figure 4.3:** PostgreSQL database configuration

- **Pool**: Sequelize parameter
  - **max**: maximum number of connections in pool
  - **min**: minimum number of connections in pool
  - **acquire**: maximum time for trying connection before throwing an error
  - **idle**: maximum time until a connection is released

Sequelize is an open-source Node.js module commonly used for Object-Relational Mapping (ORM) operations, which consists in converting data between relational databases and objects. This way, we are able to build database tables described by objects that later are converted to relational tables by Sequelize. Thefore, the previous configuration is then used for initializing Sequelize in the *index.js* file under the Models component as well as "importing" the models we have defined. After that, the last step is to add Sequelize's `sync()` method into the *server.js* file in order to instruct Sequelize to search for those models and generate SQL queries to create them in the database, in case they do not exist yet.

### 4.2.3 Models

A model defines a table in a relational database but thanks to Sequelize ORM we are able to represent it using JavaScript notation as an object. Hence, for each entity from our entity-relationship model (see figure 4.1) we must describe all of its attributes including their inherited foreign keys. There is a collection of integrity restrictions that we can apply to attributes, apart from their type, such as *null*, *unique*, *primary key*, *foreign key* and a few extra. They are very important and should be used to achieve a robust database. The requirement analysis in subsection 4.1.1 gives a good overview of each entity's attributes and their types, which makes this stage easier by visiting table 4.1 and adding the foreign keys to the correct entities.

#### *Project*

The primary key of Project is its ID, so automatically it is unique and can not be null. Its type is integer and since it follows the project creation we define it as auto-incremented. The project's title is a string, should not be empty and we consider that it should also be unique to better distinguish projects. A description is not mandatory, therefore we allow it to be null on the user's side but actually we define a default value which is triggered in that case, so it will never be null as well. The project's status can be either "Open" or "Closed" hence it is a string. The default value is "Open" at its creation and that way will never be null. Finally, the manager is represented by the username, thus is a string. However, this is a special case given it is a foreign key. Consequently, we must tell Sequelize which attribute it references from another table. In this case, the manager references the attribute "username" from Member. Moreover, let us say that the manager is deleted from the database. This operation can be rejected because it modifies the Project table as well, so we need to define an *on delete* restriction that sets that attribute to **null**.

*Member*

A member's first name and last name are both strings and can not be null. But as mentioned before, what identifies a member is its username and for that reason is the primary key of type string. No special additional restrictions were configured for this column. The email usually is large in length, so is also a string and can not be null. In this case, we are able to verify if the input has, at least, the form of an email with the restrictions *validate* and *isEmail*.

*Document*

The document's ID is the primary key of this table and also was configured to be auto-incremented integer. Both title and location are strings that should not be null, since their values come directly from the uploaded document itself. Annotated and Mapping are two boolean flags as described before. The former does not require a value from the client-side, although we defined it to be "false" as default and it is toggled after annotation. The latter can not be null, so the client-side must include this field when uploading a document for mapping. The column "rows" is only used for mapping documents representing the number of annotations that are going to be mapped, thus is an integer and we allow it to be null for "non-mapping" documents. Finally, Document has a foreign key inherited from Project which is the project ID in order to link a document as being uploaded to a given project. Therefore, we define "proj_id" as an integer foreign key but this time we want to delete the document if the project it references is deleted. In order to do that we define the *on delete* resctriction to **cascade**.

*Annotation*

The annotation table uses the internal ID as its primary key, which is an auto-incremented integer. The annotation ID follows a specific format and is stored as a string, is not unique since the same term can be recognized multiple times in the same document generating multiple entries with equal annotation ID but with different offsets. The terms are words from the text thus defined as strings that should not be null nor unique. On the other hand, the offset is an unique integer and should not be null. We must store the biomedical type of each annotation which is represented by a string, not null nor unique. Finally, the Annotation table references Document's primary key (document ID) and, like Document's foreign key constraint, also defines *on delete* action as **cascade**.

### 4.2.4 Controllers

We need to configure a controller for each entity, similar to the Models component. The controllers are responsible for setting up the REST API that will execute all the CRUD operations that manipulate objects such as: create new objects, retrieve multiple objects that match certain conditions, retrieve single objects, update objects, delete single object and delete all objects. All these operations (and more) are supported and implemented by Sequelize and work directly with the models' instances, i.e. the controllers require the models to execute the operations since they represent the database tables.

### 4.2.5 Routes

The last step is to expose the necessary endpoints, which in turn require the controllers for associating an Uniform Resource Locator (URL) pattern with a CRUD operation. The routes define the endpoints for each Hypertext Transfer Protocol (HTTP) request (GET, POST, PUT, DELETE) and essentially instruct the server what operation must be executed from the ones defined by the controllers. The tables below from 4.2 to 4.5 summarize all the endpoints exported to interact with the database. Later they will be used by the client-side.

| **Project Routes** | | |
|---|---|---|
| **Method** | **URL** | **Action** |
| POST | /api/projects | Create a new project |
| GET | /api/projects | Get all projects from the database |
| GET | /api/projects/:proj_id | Get project by ID |
| PUT | /api/projects/:proj_id | Update project by ID |
| PUT | /api/projects?proj_name=[name] | Update project by title |
| DELETE | /api/projects/:proj_id | Delete project by ID |
| DELETE | /api/projects?proj_name=[name] | Delete project by title |
| DELETE | /api/projects | Delete all projects |

**Table 4.2:** REST API exported for Projects

| Member Routes | | |
|---|---|---|
| **Method** | **URL** | **Action** |
| POST | /api/members | Create a new member |
| GET | /api/members | Get all members from the database |
| GET | /api/members/:username | Get member by username |
| PUT | /api/members/:username | Update member by username |
| DELETE | /api/members/:username | Delete member by username |
| DELETE | /api/members | Delete all members |

**Table 4.3:** REST API exported for Members

| Document Routes | | |
|---|---|---|
| **Method** | **URL** | **Action** |
| POST | /api/documents | Create a new document |
| GET | /api/documents | Get all documents from the database |
| GET | /api/documents/:doc_id | Get document by ID |
| PUT | /api/documents/:doc_id | Update document by ID |
| PUT | /api/documents?title=[title] | Update document by title |
| DELETE | /api/documents/:doc_id | Delete document by ID |
| DELETE | /api/documents | Delete all documents |

**Table 4.4:** REST API exported for Documents

| Annotation Routes | | |
|---|---|---|
| **Method** | **URL** | **Action** |
| POST | /api/annotations | Create a new annotation |
| GET | /api/annotations?doc_id=[doc_id] | Get all annotations from a document |
| GET | /api/annotations?term=[term]&type=[type] &offset=[offset] | Get annotation by term, type and offset |
| GET | /api/annotations | Get all annotations from the database |
| GET | /api/annotations/:annotation_id | Get annotation by ID |
| PUT | /api/annotations/:annotation_id | Update annotation by ID |
| DELETE | /api/annotations?annotation_id=[annotation_id] &type=[type]&offset=[offset] | Delete annotation by ID, type and offset |
| DELETE | /api/annotations?doc_id=[doc_id] | Delete all annotations from a document |

**Table 4.5:** REST API exported for Annotations

### 4.2.6 Annotation Services

Up to this point we described the back-end implementation regarding database planning and preparing its interaction with the front-end. One of the main goals is to be able to annotate biomedical text documents, thus next we will explain how the annotation was implemented. There are two ways the user can perform annotation: either automatically through a high performance biomedical framework or manually using a node package for interactively highlighting parts of text.

*Automatic Annotation*

In order to implement automatic annotation in our tool, we resorted to a NLP tool briefly described in chapter 2 as one of the selected tools that fit our requirements of annotating biomedical entities semi or fully automatically. That being said, we have chosen Neji which is an open-source framework highly specialized for biomedical concept recognition. Besides, Neji provides a web server that allow the users to manage annotation services by adding or removing dictionaries and ML models, meaning that its performance is dependent of the configuration, and an interactive user interface for annotation. We started by forking Neji's GitHub repository [3] and followed the documentation to create our local web server. Because it is a Maven [4] project, is strictly mandatory that we meet all the dependencies before we build the project, otherwise it will not work. In addition, it is important to note that the web server only works with a specific Java [5] version, in this case **8u77** as indicated in Neji's documentation. Once everything is configured correctly and the project has built without errors, using a script we may start a web server with default configurations, running an Hypertext Transfer Protocol Secure (HTTPS) server at port 8010, which in turn provides two interaction points:

- **Service**: annotation service accessible through an interactive interface or through a REST API;
- **Administration**: an administration interface for managing services.

We have indeed used the administration interface to upload dictionaries and ML models so Neji could recognize biomedical entities. Although, we are not interested in using the interactive interface provided by the service since we want to build our own. Therefore, we took advantage of the service's REST API that exposes two endpoints:

- **Annotation**: https://localhost:8010/annotate/<service_name>/annotate
- **Exportation**: https://localhost:8010/annotate/<service_name>/export

At this point, we tested those endpoints through Postman [6], an API platform for building and using APIs, to check if they were working correctly and returning information, which they were. Although, we ran into a problem: when sending HTTP requests from the front-end directly to the endpoints, the browser did not allow that origin to fetch content from it due to security reasons, returning an *XMLHttpRequest* error which is related to *Access-Control-Allow-Origin*, i.e. a Cross-Origin Resource Sharing (CORS) header. In short, when site A tries to fetch content from site B, if they are in the same domain the browser will allow it. However, in a cross-domain situation the request is only successful if B indicates that A is allowed to do so. Consequently, our solution was to use the back-end Node.js server as a proxy by sending a request from the front-end to the back-end followed by a request from the back-end to the web server. The information is then returned following the same logic but backwards. Figure 4.4 illustrates an overview of the automatic annotation process.

[3]https://github.com/BMDSoftware/neji

[4]https://maven.apache.org

[5]www.java.com

[6]https://www.postman.com

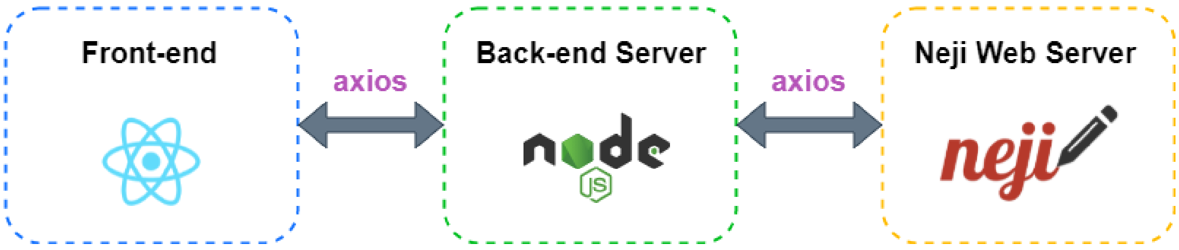


**Figure 4.4:** Automatic annotation process overview

Figure 4.5 shows how the back-end Node.js server configured CORS, starting by defining itself as the CORS origin since it will be the entity requesting content from the web server. Next, we allow the front-end (`http://localhost:3000`) to send HTTP requests of type GET, POST, PUT, DELETE and OPTIONS.

```
const express = require("express");
const cors = require("cors");
const axios = require("axios");
const app = express();
var corsOptions = {
    origin: "http://localhost:8081" // the back-end server itself
};

// https://enable-cors.org/server_expressjs.html
app.use(function(req, res, next) {
    res.header("Access-Control-Allow-Origin", "http://localhost:3000");
    res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With,
     Content-Type, Accept");
    res.header("Access-Control-Allow-Methods", "GET, POST, PUT, DELETE,
    OPTIONS");
    next();
});
```

**Figure 4.5:** CORS configuration

Then, in order to implement the connection for the annotation endpoint we use **axios** [7] which is a promise-based HTTP client that uses XMLHttpRequests on the client-side and native Node.js `http` module on the server-side. The endpoint takes two arguments, one with the text to be annotated and one with the groups representing the biomedical entities to recognize, displayed in figure 4.6.

After this we came across another problem, this time concerning Secure Sockets Layer (SSL) certificates. Neji's web server runs with HTTPS protocol and has a self-signed certificate, which is a security certificate that is not signed by a Certificate Authority (CA). What this means is that this certificate was signed using a private key instead of requesting it from a CA. Self-signed certificates provide less security properties than the ones signed from a CA and are prone to attacks. Although is not the appropriate solution to this problem, we disabled the Node.js verification for unauthorized certificates by defining an environment variable before starting the server: "`export NODE_TLS_REJECT_UNAUTHORIZED=0`".

[7]https://www.npmjs.com/package/axios

```javascript
// Neji's Web Server Annotation Endpoint
app.post(`/annotate`, function(req, res) {
    var data;
    try {
        axios.post("https://localhost:8010/annotate/default/annotate", {
                "text": req.body.text,
                "groups": req.body.groups
            }, {
                headers: {
                    'Content-Type': 'application/json'
                }
            })
            .then(response => {
                //console.log(response.data);
                data = response.data;
                return res.status(200).send(data);
            })
            .catch(error => {
                console.error(error);
            });
    } catch(err) {
        console.error(err);
        return res.status(500).send("[SERVER] Could not annotate that
            document.");
    }
});
```

**Figure 4.6:** Implementation of Neji's Web Server annotation endpoint

As such, the front-end after this point was able to fetch content from Neji's web server. It returns a JSON object with three keys, as shown in figure 4.7:

- **entities**: array of strings divided by vertical bars. The first section is the term recognized, the second an annotation ID following a specific format and the third is the index representing where the term is located in the text.
- **ids**: array of strings containing the annotation IDs of the recognized terms
- **text**: the text requested to be annotated

```json
{
    "entities": [
        DMD|UMLS:C0013264:T047:DISO|32,
        blood vessel|UMLS:C0005847:T023:ANAT|679,
        ...,
    ],
    "ids": [
        UMLS:C0013264:T047:DISO,
        UMLS:C0005847:T023:ANAT,
        ...
    ],
    "text": "In Duchenne muscular dystrophy (DMD), ..."
}
```

**Figure 4.7:** Example of Neji's annotation endpoint response

*Manual Annotation*

Another type of annotation that we decided to implement is manual annotation. It is not relevant if a given document has been pre-annotated automatically with Neji or not, as manual annotation can be performed anytime. We implemented this functionality with a React component provided by Node Package Manager (NPM), named "react-text-annotate" [8], that allows for interactively highlighting parts of text. This component provides two modes of operation which differ in the way the text is entered, either split into tokens or a full string, the latter being what we chose to work with. It allows to drag or double click to select text which in turn is highlighted depending on the selected tag, i.e. biomedical entity. Therefore, we define an object associating each biomedical entity to a color coded in hexadecimal (see figure 4.8). Without this component we would need to implement mouse events as well as manually manipulate the HTML Document Object Model (DOM) objects in order to highlight them. However, the manual annotation when compared to the automatic process has a few disadvantages. The most noticeable is that it is a very time-consuming and labor-expensive task, but in particular it will not provide annotation IDs like Neji does.

```
const TAG_COLORS = {
    'PATH'      : '#D9EDF7',
    'DISO'      : '#BBD3E6',
    'FUNC'      : '#F2DEDE',
    'PRGE'      : '#FEE4BD',
    'ENZY'      : '#E8CE9A',
    'COMP'      : '#EEFFEC',
    'PROC_FUNC' : '#DFF0D8',
    'PROC'      : '#BADDBC',
    'CHED'      : '#E3D7FF',
    'MRNA'      : '#CED4B4',
    'CELL'      : '#E7E7E7',
    'SPEC'      : '#74d3ed',
    'ANAT'      : '#8EC3ED',
}
```

**Figure 4.8:** Entity colors in Hex codes

### 4.2.7 Mapping Services

For the task of mapping terms to standard vocabulary concepts we used OHDSI's open-source tool Usagi (see subsection 2.1.5). Unlike Neji, Usagi does not provide a REST API. For this reason, we devised an alternative strategy that allowed us to exploit its functionalities in our tool: we included Usagi's code as a module inside Neji. However, this is not enough because there is no way to access it through the front-end, given that they are two completely separate projects. For instance, for annotating texts the front-end requests it through the back-end server, which in turn executes the request to Neji's web server. Thus, our goal is to follow the same logic of the annotation process.

[8] https://www.npmjs.com/package/react-text-annotate

*Usagi Search Engine*

Before breaking down the mapping endpoint implementation, we will explain how we used Usagi's search engine in order to search and map terms to standard vocabulary concepts. The index must be built beforehand because it is where all the information related to the vocabularies is stored. Later, Usagi will use the index to process the searched terms, by using the method `search(String searchTerm, boolean useMtl, Collection<Integer> filterConceptIds, ...)` from its search engine which takes eight arguments:

- **searchTerm**: term to map
- **useMlt**: boolean flag that sets filter heuristics
- **filterConceptIds**: collection of concept IDs
- **filterDomains**: vector of domains
- **filterConceptClasses**: vector of concept classes
- **filterVocabularies**: vector of vocabularies
- **filterStandard**: boolean flag to filter standard concepts
- **includeSourceConcepts**: boolean flag to include source concepts

Concerning the index building process, a contributor who developed Usagi stated [9] that they used Apache Lucene [10] to perform this operation, due to its powerful indexing and search feature, as well as spellcheking, allowing for fast comparison between millions of terms. Moreover, they used BerkeleyDB [11] to store concept-related information such as IDs, ancestors, relationships and the concepts themselves. With access to this key feature, we created a simple class that starts by checking if the index is built, as shown in figure 4.9. If not, we build it after indicating where the vocabulary folder is located. After that, we implemented a method named `searchTerms(List<String> termsList, boolean useMlt, ...)` with the same arguments as the original method, except that the first one is a list of strings instead of a single term. Inside it we loop through that list where we invoke the original method in order to map multiple annotations output by Neji or imported terms (see figure 4.10).

```java
    private void configureIndex() {
        if(!Global.usagiSearchEngine.mainIndexExists()) {
            IndexBuildCoordinator buildIndex = new IndexBuildCoordinator();
            String vocabFolder = Global.folder + "/vocabulary";
            String loincFolder = null;
            buildIndex.buildIndexes(vocabFolder, loincFolder);
        } else {
            Global.usagiSearchEngine.openIndexForSearching(false);
            Global.dbEngine.openForReading();
        }
    }
```

**Figure 4.9:** Method responsible for building the index

[9] https://forums.ohdsi.org/t/mapping-concepts-in-usagi-snomed-vocabulary/1439/6
[10] http://lucene.apache.org
[11] https://www.oracle.com/pt/database/technologies/related/berkeleydb.html

```java
    public HashMap<String, Object> searchTerms(List<String> termsList,
        boolean useMlt, Collection<Integer> filterConceptIds, ...) {

        List<List<ScoredConcept>> usagiOutput = new ArrayList<>();
        HashMap<String, Object> results = new HashMap<String, Object>();
        for (String term : termsList) {
            List<ScoredConcept> concepts =
                Global.usagiSearchEngine.search(term, useMlt,
                filterConceptIds, filterDomains, filterConceptClasses,
                filterVocabularies, filterStandard, includeSourceConcepts);

            usagiOutput.add(concepts);
            results.put(term, concepts);
        }
        return results;
    }
```

**Figure 4.10:** Method for searching terms using Usagi's search engine

*Creating a Mapping Endpoint*

The easier and most sensible solution to expose a mapping endpoint was to implement it inside Neji's web server, since it already provides the infrastructure and allows us to use the Usagi module. Therefore, the first step was to understand how the web services worked in Neji and how to implement a new one. They are implemented with a Servlet and Java Server Pages (JSP). Servlets are Java-based server-side programs that implement the Servlet interface and process all client requests directed to that server. In turn, the Servlet interface is an intermediate layer that sits between client requests and server-side programs. JSP on the other hand, is a web application development technology that is related to Servlets and is used to develop web pages using HTML code inside Java classes.

When the web server starts, it creates a Servlet container that will host each Servlet. Therefore, the first step is to configure a Servlet dedicated to the mapping task in an XML file, indicating various attributes such as name, class, packages, a resource and a URL pattern. In this case, we named it "Usagi Mapping" belonging to the class ServletContainer which will look for a resource inside the package `pt.ua.tm.neji.web.mapping` when the URL pattern matches **"/mapping/*"**. The configuration is shown in figure 4.11.

For each endpoint, Neji defines a resource that is responsible to execute its code (e.g. annotation or managing services). Hence, we created a mapping resource that allows the web server to use Usagi's module code. Figure 4.12 shows a snippet of the relevant code that implements it. This class will be invoked on a POST action matching the path **"/mapping/search"**, consuming parameters encoded in *x-www-form-urlencoded* which correspond to the same arguments as our method `searchTerms(...)`. However, this time the terms are in a comma-separated string which is then split into a list of strings and the "useMlt" boolean is always set to true. The results are then converted to JSON and returned in that format.

```
<servlet>
    <servlet-name>Usagi Mapping</servlet-name>
    <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
    <init-param>
        <param-name>com.sun.jersey.config.property.packages</param-name>
        <param-value>pt.ua.tm.neji.web.mapping</param-value>
    </init-param>
    <init-param>
        <param-name>com.sun.jersey.api.json.POJOMappingFeature</param-name>
        <param-value>true</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>Usagi Mapping</servlet-name>
    <url-pattern>/mapping/*</url-pattern>
</servlet-mapping>
```

**Figure 4.11:** Mapping Servlet configuration

```
@Path("/")
public class MappingResource {
    /**
     * Invoked on POST call to map Neji output annotations or imported terms
     * @param annotations comma separated annotations
     */
    @POST
    @Path("/search")
    @Consumes("application/x-www-form-urlencoded")
    @Produces(MediaType.APPLICATION_JSON)
    public Response search(@FormParam("annotations") final String annotations, @FormParam("filterConceptIds") final boolean filterConceptIds, @FormParam("filterStandardConcepts") final boolean filterStandardConcepts, ...) {

        Object responseToReturn;
        ...
        String[] splitAnnotations = annotations.split(",");
        List<String> annotationsList = Arrays.asList(splitAnnotations);
        if(annotationsList.size() == 0) {
            responseToReturn = "There are no annotations to map.";
        } else {
            ...
            HashMap<String, Object> results = usagi.searchTerms(annotationsList, useMlt, null, domains, conceptClasses, vocabularies, filterStandard, includeSourceConcepts);
            return Response.ok(new Gson().toJson(results)).build();
        }
        return Response.ok(responseToReturn).build();
    }
}
```

**Figure 4.12:** Implementation of Mapping Resource

*Back-end server endpoint*

With the endpoint created at Neji's web server, we must complete the "pipeline" and implement the logic inside the back-end server so it acts as a proxy between Neji and the front-end, similar to what we did for the annotation. The only difference is how the parameters are added to the request, due to the *x-www-form-urlencoded* format, as shown in figure 4.13.

```
    app.post("/mapping", function(req, res) {
        var data;

        // https://axios-http.com/docs/urlencoded
        const params = new URLSearchParams();
        params.append('annotations', req.body.annotations);
        params.append('filterConceptIds', req.body.filterConceptIds);
        params.append('filterSelectedConcepts', req.body.
            filterSelectedConcepts);
        params.append('filterStandardConcepts', req.body.
            filterStandardConcepts);
        params.append('filterIncludeSourceTerms', req.body.
            filterIncludeSourceTerms);
        params.append('filterConceptClass', req.body.filterConceptClass);
        params.append('filterVocabulary', req.body.filterVocabulary);
        params.append('filterDomain', req.body.filterDomain);

        try {
            axios.post("https://localhost:8010/mapping/search", params)
                .then(response => {
                    data = response.data;
                    return res.status(200).send(data);
                })
                .catch(error => {
                    console.error(error);
                });
        } catch (err) {
            console.error(err);
            return res.status(500).send("[SERVER] Could not map the requested
                annotations.");
        }
    })
```

**Figure 4.13:** Implementation of Neji's Web Server mapping endpoint

### 4.2.8 Handling File Uploads

Finally, we still need to describe how the back-end Node.js server handles file uploads. The idea is to allow users to upload text files for annotation and/or CSV files for mapping, originating from local storage. To that end, we used **Multer** [12] which is an NPM package primarily used for uploading files. In terms of storage, Multer provides two options: disk or memory. The memory storage engine stores the files in memory as buffer objects, but we want to store them locally organized by projects, so we used disk storage. Thus, we create a directory for uploads that is divided in sub-directories for each project, which in turn store the uploaded files. Figure 4.14 shows how Multer is configured: first we resolve the user's absolute path

[12]https://www.npmjs.com/package/multer

as the root upload path, ending in `"/uploads/projects"`. Afterwards, we use that absolute path and merge a section indicating the project ID, resulting in the final destination of the file(s) uploaded. At this stage, we use the Node.js fs module [13] to access and interact with the file system. In this case, the method `mkdirSync(path[, options])` recursively creates the directory when `var upload` is used, as we will explain next.

```
var uploadsRelPath = "uploads/projects";
var uploadsAbsPath = resolve(uploadsRelPath);

var storage = multer.diskStorage({
    destination: function (req, file, cb) {
        var projId = req.url.split("/")[2];
        var path = `${uploadsRelPath}/${projId}`;
        fs.mkdirSync(path, { recursive: true });
        cb(null, path)
    },
    filename: function (req, file, cb) {
        cb(null, file.originalname )
    }
});

// allow multiple files to be uploaded
var upload = multer({ storage: storage }).array('file');
```

**Figure 4.14:** Multer implementation

In order to actually use the Multer instance, we must use the `var upload` which from a user's perspective is used when an upload is requested. As such, the front-end sends an HTTP request through axios to the back-end server. The endpoint defined for this purpose is shown in figure 4.15.

```
// API endpoint to upload files and store them in local storage
app.post('/upload/:projId', function(req, res) {
    upload(req, res, function(err) {
        if (err instanceof multer.MulterError) {
            return res.status(500).json(err);
        } else if (err) {
            return res.status(500).json(err);
        }
        return res.status(200).send(uploadsAbsPath);
    })
});
```

**Figure 4.15:** Back-end endpoint for uploading documents

In addition to the upload endpoint, we defined three more endpoints: one when a project is deleted, another for when a single file is deleted from a project and one for reading files' content. Each project owns its directory in the uploads folder that is created without files, at first. When a given project is deleted, we must delete that directory and its contents. For this reason, we use Node.js fs method `rmdirSync(path[, options)]` with the recursive option after checking if it exists with `existsSync(path)`, as shown in figure 4.16.

[13]https://nodejs.org/api/fs.html

```
// API endpoint to delete a project directory and its contents from the
    local storage
app.delete('/deleteProject', function(req, res) {
    var projDirectory = `${uploadsAbsPath}/${req.body.projId}`;
    try {
        if(fs.existsSync(projDirectory)) {
            fs.rmdirSync(projDirectory, { recursive: true });
        }
    } catch(err) {
        console.error(err);
        return res.status(500).send(`[SERVER] Could not delete project
            directory ${projDirectory}.`);
    }
    return res.status(200).send(`[SERVER] Project directory deleted: ${
        projDirectory}`);
});
```

**Figure 4.16:** Back-end endpoint to delete a project's directory and its contents

Next, to handle with single-file deletion the Node.js fs module provides a method defined as `unlinkSync(path)` to delete a file located at the target path. Its implementation is as shown in figure 4.17.

```
// API endpoint to delete a single file from the local storage
app.delete('/deleteFile', function(req, res) {
    var path = req.body.path;
    try {
        fs.unlinkSync(path);
    } catch(err) {
        console.error(err);
        return res.status(500).send(`[SERVER] Could not delete document $
            {path}.`);
    }
    return res.status(200).send(`[SERVER] Document deleted: ${path}`);
});
```

**Figure 4.17:** Back-end endpoint to delete a project's document

Finally, to end the back-end implementation section, we defined an endpoint dedicated to feed the front-end with a given project files' contents that are loaded when the user navigates to the annotation or mapping interface. It is as simple and straightforward as the previous described, in this case we took advantage of the method `readFileSync(path[, options])` that returns the contents of a file located in the target path in the format of a string because we specify the encoding to be "utf8". This endpoint is shown in figure 4.18.

```
// Read a file and send its contents
app.get('/readFile', function(req, res) {
    var path = req.query.path;
    try {
        var content = fs.readFileSync(path, 'utf8');
    } catch(err) {
        console.error(err);
        return res.status(500).send(`[SERVER] Could not read the document
             at ${path}.`);
    }

    return res.status(200).send(content);
});
```

**Figure 4.18:** Back-end endpoint to read a file's content

## 4.3 Front-end

The front-end or client-side of the tool is developed with React.js as mentioned in chapter 3. Its structure is divided into four parts: **Components**, **Layouts**, **Services** and **Views**. Before delving into them, we will start by explaining how the pages or views are loaded and switched to provide navigation. The entry-point of the application is defined in the *index.js* file by instructing React that it should render the *App.js* layout when starting. The layout defines the main structure of the application (e.g. header, main, footer) and all the styles that will be applied to the components, thus it can be compared to a main class in programming. For enabling navigation between pages we used **React Router** [14], that allows to define multiple routes and when the URL path matches any of them it redirects the user to a specific view rendered by that route. In turn, the routes are defined in *routes.js* where, for each view that we want to render, it is defined its path, name and layout.

[14]https://reactrouter.com

```
frontend/
├── public/
└── src/
    ├── Components/
    │   ├── Copyright.js
    │   ├── Header.js
    │   ├── Nav.js
    │   └── Navigator.js
    ├── Layouts/
    │   └── App.js
    ├── Services/
    │   ├── annotation.service.js
    │   ├── document.service.js
    │   ├── mapping.service.js
    │   └── project.service.js
    ├── Views/
    │   ├── AnnotateProject.js
    │   ├── Mapping.js
    │   ├── Project.js
    │   └── ProjectsPage.js
    ├── http-common.js
    ├── index.js
    └── routes.js
```

**Figure 4.19:** Front-end project directory tree

### 4.3.1 Components

The components are independent and simple modules with very specific purposes. Usually they are pieces of the front-end that are necessary on multiple occasions or present at all times. For instance, most websites that we visit daily have a header showing perhaps a logo, a search bar or menus for navigation, a footer with contacts and social media links, etc. Those are static elements in some way because do not require updates throughout the user experience and one may think that they should be instantiated in every page. However, we only instantiate them once in the layout.

### 4.3.2 Services

The data services are the bridge between the front-end and the back-end, sending HTTP requests for any endpoint exported at the back-end, described in subsection 4.2.5. We accomplish that by using axios which is promise-based as aforementioned. A promise represents an asynchronous operation and its resulting value and can be in one of three states: pending, fulfilled or rejected. Every action that requires database interaction starts by creating a new promise, such as creating new projects, uploading documents to projects, deleting something or retrieving information. The file *http-common.js* is responsible for that task by using axios' `create()` method which takes two arguments: the baseURL that should match the URL configured at the back-end server and headers that allow to define the content type of the data exchanged, in this case JSON (see figure 4.20).

```
import axios from "axios";

export default axios.create({
    baseURL: "http://localhost:8080/api",
    headers: {
        "Content-type": "application/json"
    }
});
```

**Figure 4.20:** Axios configuration in http-common.js

Alike the back-end structure, it is necessary to create a data service for each entity that will be interacting with the database, for instance projects, documents and annotations. We did not implement it for members, because they were not used. Nevertheless, it is very straightforward to do it in the future, as the back-end and the database are indeed ready for it. They are quite simple as it is only necessary to implement functions for each endpoint. The figure 4.21 demonstrates two examples: the first retrieves an annotation by its ID represented by a *param*, because it is explicitly part of the URL after a forward slash; the second deletes all the annotations from a document by the document's ID passed through a *query*, which is represented by a match after a question mark.

```
import http from "../http-common";

class AnnotationDataService {
    ...
    // Get annotation by ID
    getByAnnotationID(annotation_id) {
        return http.get(`/annotations/${annotation_id}`);
    }

    // Delete all annotations from a document
    deleteAll(doc_id) {
        return http.delete(`/annotations?doc_id=${doc_id}`);
    }
    ...
}
```

**Figure 4.21:** Annotation Data Service Example

### 4.3.3 Views

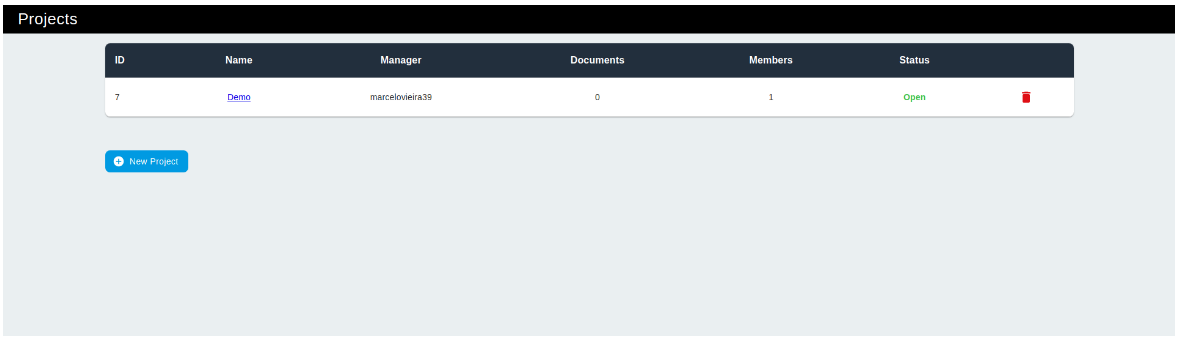
The user interfaces that users engage with are called views or pages. They carry all the information that users see and interact with, such as tables, buttons, containers and many more React components. In order to make things easier, every component used for the application is imported from Material UI [15], MUI for short, which is a simple and flexible library for creating quicker, customizable and accessible React applications by providing ready-to-use components. They allow customization through props with pre-defined values

[15]https://v4.mui.com/pt/

available, but it is also possible to override specific attributes using in-line styles which is very useful. Our annotation and mapping tool has four views: a general projects' page, an individual project's page, an annotation page and finally a mapping page.

### *Projects Page*

The projects page is the starting point of the user experience. Since our solution is based on the idea of projects, this page provides information on a table about every project stored in the database, including information related to it such as title, manager, number of documents, number of members and status. In this page the user is able to create new projects and delete existing ones.

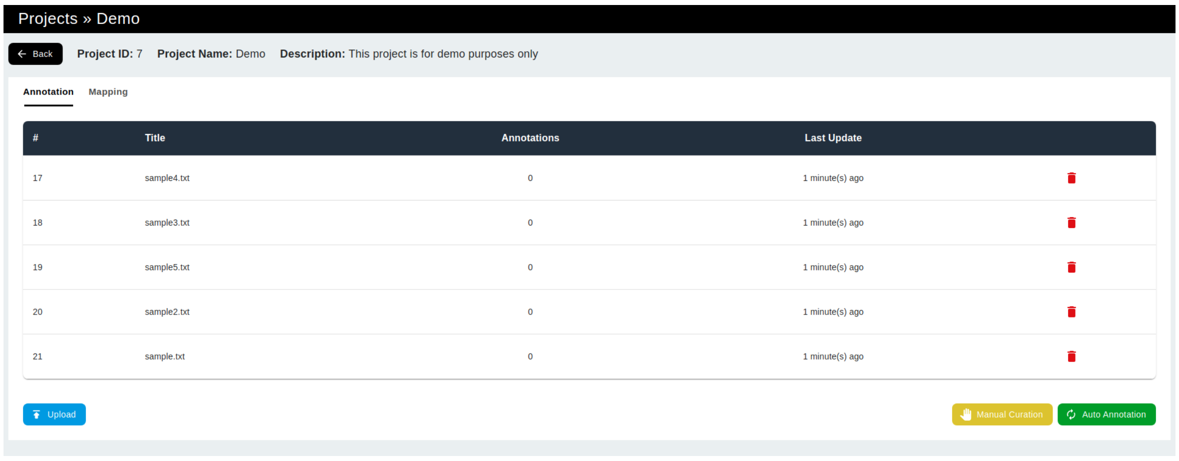

Projects

| ID | Name | Manager | Documents | Members | Status | |
|---|---|---|---|---|---|---|
| 7 | Demo | marcelovieira39 | 0 | 1 | Open | 🗑 |

**Figure 4.22:** Projects page

### *Individual Project Page*

Each project can be accessed by selecting its title from the table of projects on the projects page. This page is divided into two separate tabs, one for annotation and another for mapping. These are not the actual interfaces for that tasks, but rather the place for documents' management where the users are allowed to upload text documents for annotation and CSV documents with terms to be mapped into standard vocabulary concepts. Therefore, in this view we present simple tables with document-related information. For annotation documents, the table is populated with the documents' titles, number of annotations and the time elapsed since the last update. On the other hand, the mapping documents' table shows a list with their titles, number of rows (i.e. equivalent to the number of terms to map) and also the last update. Concerning the upload of documents, both types are restricted to 2MB and text and CSV format accordingly. Figure 4.23 shows this view on the annotation tab with five documents uploaded.


Projects » Demo

Back | **Project ID:** 7 | **Project Name:** Demo | **Description:** This project is for demo purposes only

Annotation | Mapping

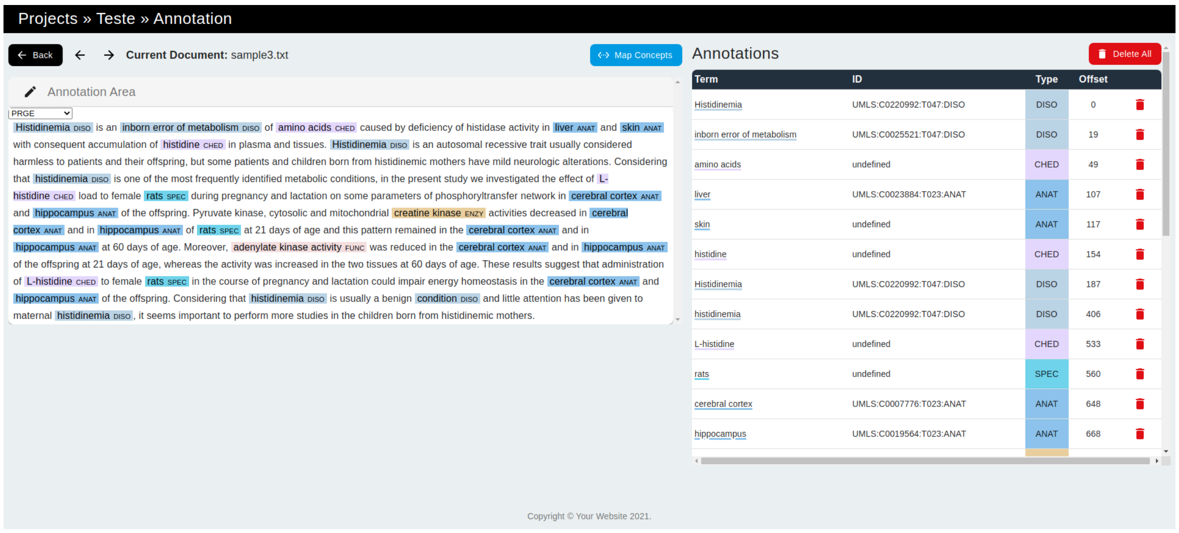
| # | Title | Annotations | Last Update |
|---|---|---|---|
| 17 | sample4.txt | 0 | 1 minute(s) ago |
| 18 | sample3.txt | 0 | 1 minute(s) ago |
| 19 | sample5.txt | 0 | 1 minute(s) ago |
| 20 | sample2.txt | 0 | 1 minute(s) ago |
| 21 | sample.txt | 0 | 1 minute(s) ago |

Upload | Manual Curation | Auto Annotation

**Figure 4.23:** Individual project's page

*Annotation Interface*

The annotation interface is where the process of annotation takes place. It is intended to work for an entire project, which means that it must provide easy navigation between the project's documents. Moreover, the user interface design is very important in the sense that an annotator's performance will be directly influenced by how the tool works, i.e. it should be fast, simple to use and intuitive to a point that there is no need to teach users how to use it. Therefore, we designed this page divided into two vertical sections: the left column holds an annotation area where a document's text is loaded and highlighted with annotations. Plus, is where the experts will work with manual annotations by adding new ones or deleting existing ones. The right column is reserved to a table populated with the existing annotations belonging to that document, allowing the annotators to easily monitor their work. Each annotation in the table is displayed with the term underlined with the color of its type, the annotation ID, its type with background color, its offset in the text and a delete button icon.


Projects » Teste » Annotation

Back | **Current Document:** sample3.txt | Map Concepts

Annotation Area

PRGE

Histidinemia DISO is an inborn error of metabolism DISO of amino acids CHED caused by deficiency of histidase activity in liver ANAT and skin ANAT with consequent accumulation of histidine CHED in plasma and tissues. Histidinemia DISO is an autosomal recessive trait usually considered harmless to patients and their offspring, but some patients and children born from histidinemic mothers have mild neurologic alterations. Considering that histidinemia DISO is one of the most frequently identified metabolic conditions, in the present study we investigated the effect of L-histidine CHED load to female rats SPEC during pregnancy and lactation on some parameters of phosphoryltransfer network in cerebral cortex ANAT and hippocampus ANAT of the offspring. Pyruvate kinase, cytosolic and mitochondrial creatine kinase ENZY activities decreased in cerebral cortex ANAT and in hippocampus ANAT of rats SPEC at 21 days of age and this pattern remained in the cerebral cortex ANAT and in hippocampus ANAT at 60 days of age. Moreover, adenylate kinase activity FUNC was reduced in the cerebral cortex ANAT and in hippocampus ANAT of the offspring at 21 days of age, whereas the activity was increased in the two tissues at 60 days of age. These results suggest that administration of L-histidine CHED to female rats SPEC in the course of pregnancy and lactation could impair energy homeostasis in the cerebral cortex ANAT and hippocampus ANAT of the offspring. Considering that histidinemia DISO is usually a benign condition DISO and little attention has been given to maternal histidinemia DISO, it seems important to perform more studies in the children born from histidinemic mothers.

Annotations | Delete All

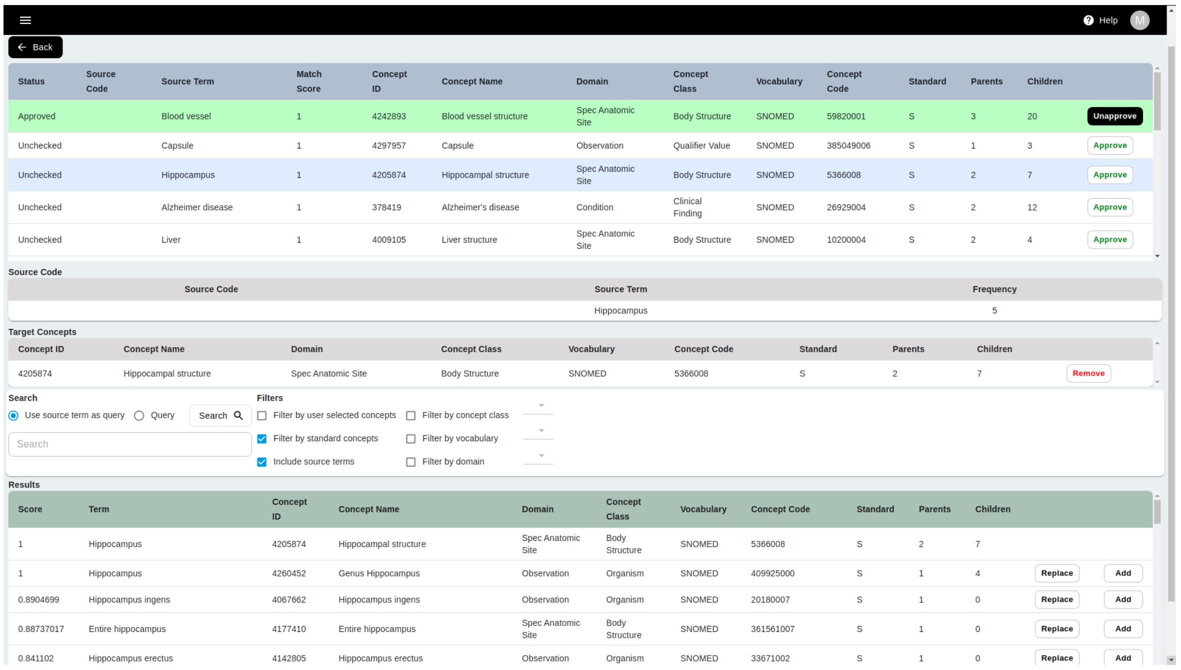
| Term | ID | Type | Offset |
|---|---|---|---|
| Histidinemia | UMLS:C0220992:T047:DISO | DISO | 0 |
| inborn error of metabolism | UMLS:C0025521:T047:DISO | DISO | 19 |
| amino acids | undefined | CHED | 49 |
| liver | UMLS:C0023884:T023:ANAT | ANAT | 107 |
| skin | undefined | ANAT | 117 |
| histidine | undefined | CHED | 154 |
| Histidinemia | UMLS:C0220992:T047:DISO | DISO | 187 |
| histidinemia | UMLS:C0220992:T047:DISO | DISO | 406 |
| L-histidine | undefined | CHED | 533 |
| rats | undefined | SPEC | 560 |
| cerebral cortex | UMLS:C0007776:T023:ANAT | ANAT | 648 |
| hippocampus | UMLS:C0019564:T023:ANAT | ANAT | 668 |

Copyright © Your Website 2021.

**Figure 4.24:** Annotation interface

*Mapping Interface*

The mapping interface aims to aid the manual process of mapping terms to standard vocabulary concepts, by automatically searching and comparing them with millions of concepts and associate the results to a match score representing its degree of confidence. As stated before, we used OHDSI's open-source tool Usagi, which in turn uses Apache Lucene, for the task of searching and indexing. The mapping interface is more complex than the annotation's, due to the vast amount of information output by Usagi. Thus, it is vertically divided into 3 components that span the entire width of the screen: at the top is an overview of the current mapped concepts which are the suggested cope mappings based on search filters set at import stage and term similarity, or in other words, achieved by comparing the terms to concept names and synonyms. Each automatically generated mapping has a match score associated to it, ranging from 0 to 1 with 1 being a confident match. When a match score is low, then an intervention is needed to replace that mapping for another concept with a higher score. At the bottom section, the user can search for a more suitable concept and choose to replace the target concept present at the middle section or add another one, as well as define a set of filters. The middle section displays a larger view of the selected mapping. In case a mapping is correct, the user can simply hit the approve button and continue his work [69].



| Status | Source Code | Source Term | Match Score | Concept ID | Concept Name | Domain | Concept Class | Vocabulary | Concept Code | Standard | Parents | Children | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Approved | | Blood vessel | 1 | 4242893 | Blood vessel structure | Spec Anatomic Site | Body Structure | SNOMED | 59820001 | S | 3 | 20 | Unapprove |
| Unchecked | | Capsule | 1 | 4297957 | Capsule | Observation | Qualifier Value | SNOMED | 385049006 | S | 1 | 3 | Approve |
| Unchecked | | Hippocampus | 1 | 4205874 | Hippocampal structure | Spec Anatomic Site | Body Structure | SNOMED | 5366008 | S | 2 | 7 | Approve |
| Unchecked | | Alzheimer disease | 1 | 378419 | Alzheimer's disease | Condition | Clinical Finding | SNOMED | 26929004 | S | 2 | 12 | Approve |
| Unchecked | | Liver | 1 | 4009105 | Liver structure | Spec Anatomic Site | Body Structure | SNOMED | 10200004 | S | 2 | 4 | Approve |

Source Code

| Source Code | Source Term | Frequency |
|---|---|---|
| | Hippocampus | 5 |

Target Concepts

| Concept ID | Concept Name | Domain | Concept Class | Vocabulary | Concept Code | Standard | Parents | Children | |
|---|---|---|---|---|---|---|---|---|---|
| 4205874 | Hippocampal structure | Spec Anatomic Site | Body Structure | SNOMED | 5366008 | S | 2 | 7 | Remove |

Search

[✓ Use source term as query] Query Search

Search

Filters

☐ Filter by user selected concepts
☐ Filter by concept class
☑ Filter by standard concepts
☐ Filter by vocabulary
☑ Include source terms
☐ Filter by domain

Results

| Score | Term | Concept ID | Concept Name | Domain | Concept Class | Vocabulary | Concept Code | Standard | Parents | Children | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Hippocampus | 4205874 | Hippocampal structure | Spec Anatomic Site | Body Structure | SNOMED | 5366008 | S | 2 | 7 | | |
| 1 | Hippocampus | 4260452 | Genus Hippocampus | Observation | Organism | SNOMED | 409925000 | S | 1 | 4 | Replace | Add |
| 0.8904699 | Hippocampus ingens | 4067662 | Hippocampus ingens | Observation | Organism | SNOMED | 20180007 | S | 1 | 0 | Replace | Add |
| 0.88737017 | Entire hippocampus | 4177410 | Entire hippocampus | Spec Anatomic Site | Body Structure | SNOMED | 361561007 | S | 1 | 0 | Replace | Add |
| 0.841102 | Hippocampus erectus | 4142805 | Hippocampus erectus | Observation | Organism | SNOMED | 33671002 | S | 1 | 0 | Replace | Add |

**Figure 4.25:** Mapping interface

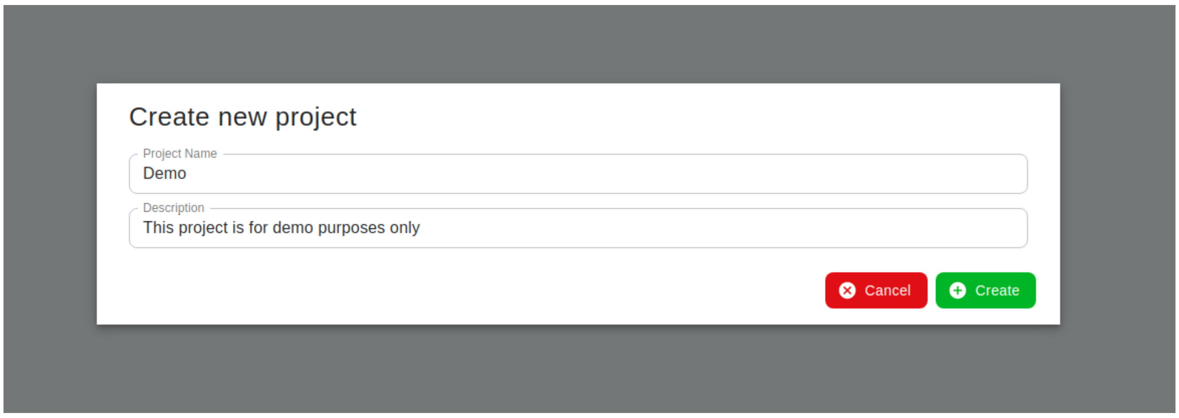
CHAPTER 5

# Walkthrough

In this chapter, we describe a walkthrough of our tool explaining step-by-step how the users can exploit its functionalities in order to perform domain-oriented tasks, either related to automatically recognized biomedical entities through NLP techniques or mapping non-standard concepts to standard vocabulary concepts or both tasks. We start at section 5.1 which is the “home page” of the application where the creation of projects take place, followed by uploading documents at section 5.2 which are then ready to be annotated or mapped, as described in sections 5.3 and 5.4, respectively.

## 5.1 Manage Projects

The first page that the user interacts with is the projects’ management page, as aforementioned in subsection 4.3.3, providing information about every project stored in the database. It serves two purposes, the main one being the management of projects, i.e. creating and deleting, and allow accessing the next page. If there are no records of projects this page will be empty, so in order to create one the user must click the blue button "New Project" which will prompt a creation dialog as figure 5.1 shows, where at least a title must be input and an optional description. After pressing "Create" the dialog closes and the project is stored in the database simultaneously. Moreover, a snackbar (also known as a toast) is rendered at the bottom center of the screen to inform the user that the operation was successful and disappears after six seconds without any user’s mouse clicks (see figure 5.2). After this, the projects page has information to display on a table, thus will look like figure 5.4, showing the project’s ID which is meaningless but we decided to include it as a way of better identifying the project if the list is extensive, the title (name), manager, number of documents and members and its status. In order to delete it, the user may press the red bin icon, which alike the creation, will prompt a dialog to confirm this action (see figure 5.5). In case of confirmation, a snackbar is rendered informing that the operation was executed (see figure 5.3), deleting the project from the database. To continue navigating the user must click on the title which is a link that routes the application to the next page.

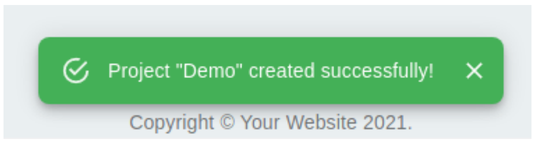


Create new project

Project Name: Demo

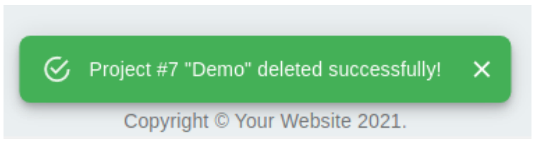
Description: This project is for demo purposes only

**Figure 5.1:** Project creation dialog



Project "Demo" created successfully!

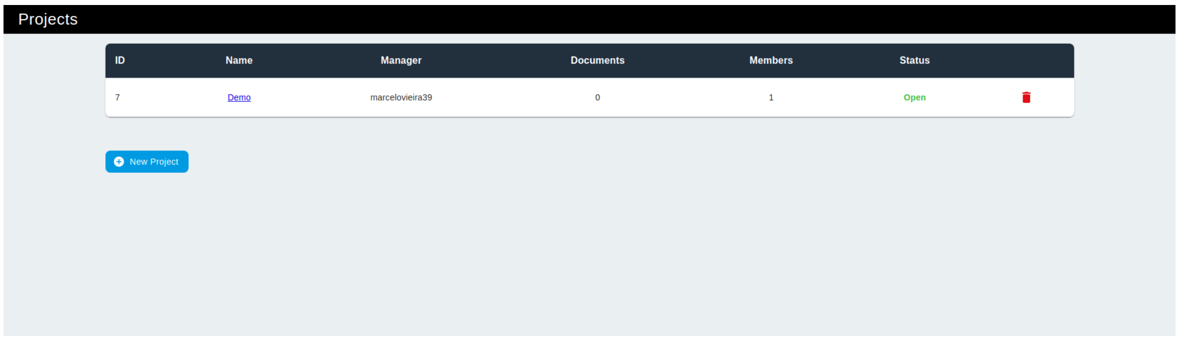
Copyright © Your Website 2021.



Project #7 "Demo" deleted successfully!

Copyright © Your Website 2021.

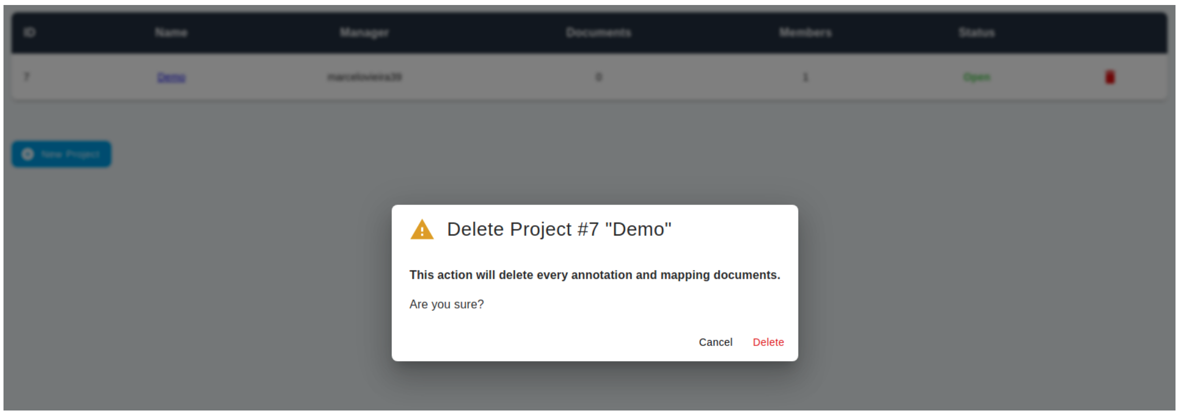
**Figure 5.2:** Project creation snackbar **Figure 5.3:** Project deletion snackbar



Projects

| ID | Name | Manager | Documents | Members | Status | |
|---|---|---|---|---|---|---|
| 7 | Demo | marcelovieira39 | 0 | 1 | Open | |

**Figure 5.4:** Projects page



Delete Project #7 "Demo"

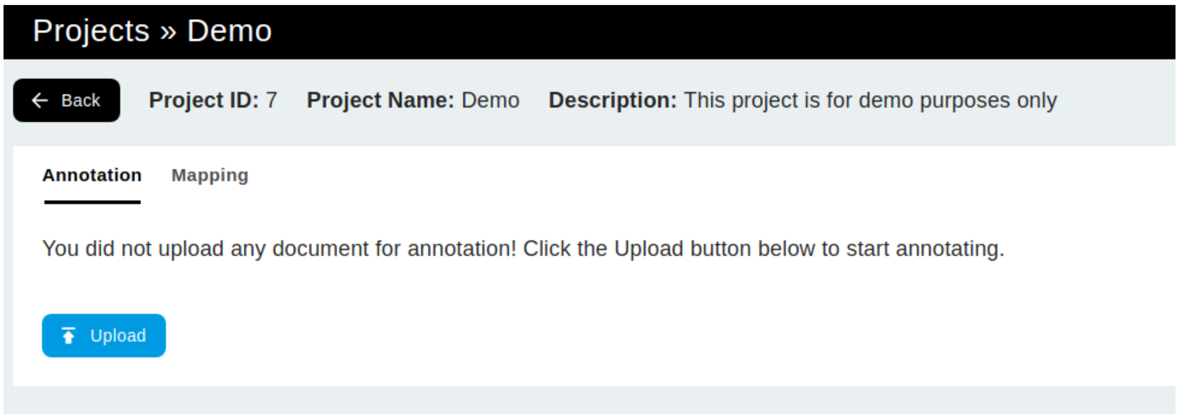
This action will delete every annotation and mapping documents.

Are you sure?

**Figure 5.5:** Project deletion dialog

## 5.2 Manage Documents

Now the user is looking at a dedicated page belonging to the project that he/she clicked earlier. This page is shared between annotation and mapping separated into tabs, allowing for managing their related documents. If there are none, each tab will look like figure 5.6, informing the user to upload documents to start annotating/mapping.

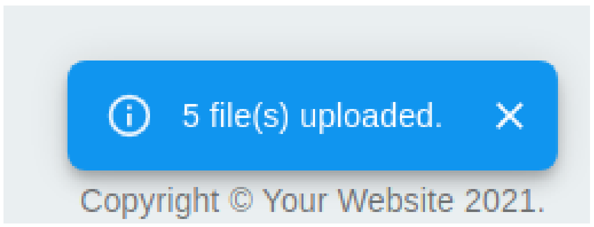

Projects » Demo

Project ID: 7 Project Name: Demo Description: This project is for demo purposes only

You did not upload any document for annotation! Click the Upload button below to start annotating.

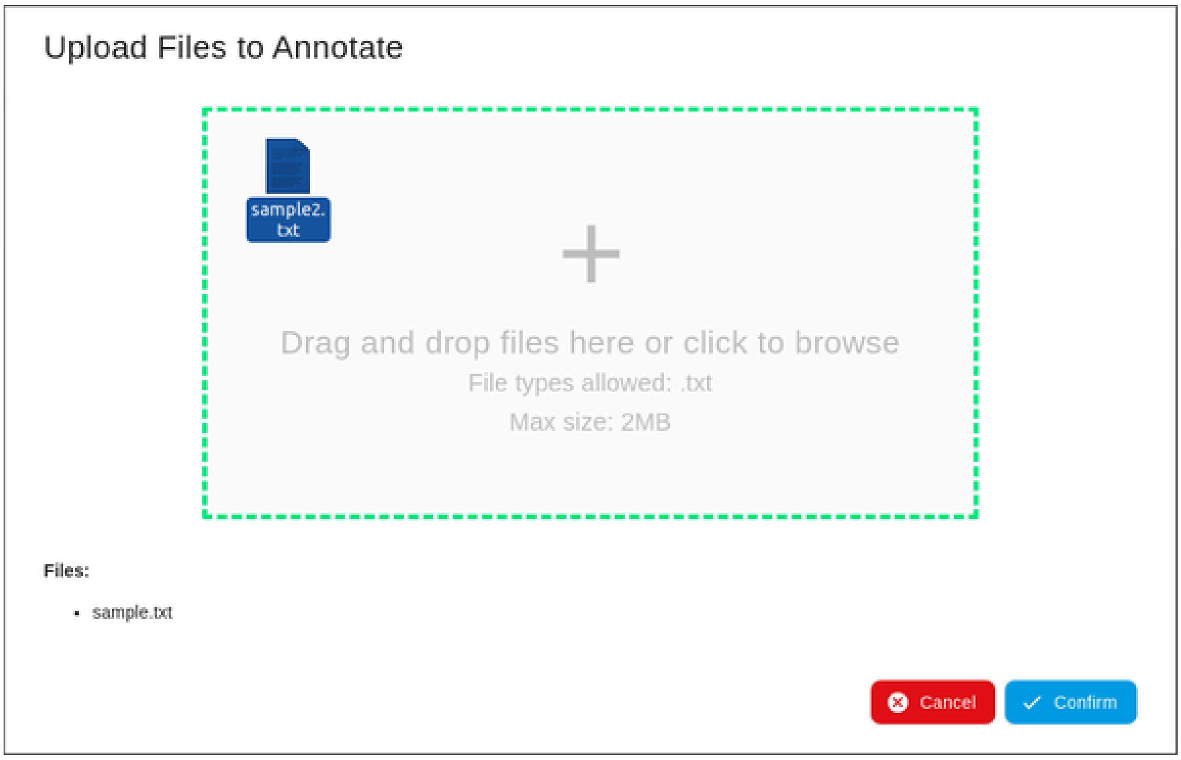
**Figure 5.6:** Annotation tab without uploaded documents

That said, when clicking the "Upload" button a dialog is prompt (see figure 5.8) with a dropzone that allows to select one or multiple files from local storage or simply drag them into the dropzone. They must comply with the restrictions imposed, which are relative to their format and size: for annotation is only allowed text files (.TXT) and for mapping CSV files, weighing both a maximum of 2MB. Dragged files that do not comply will trigger a red dotted line leading to rejection, while those who comply are accepted with a green dotted line, which in turn form the list of accepted files below. Concerning the annotation tab, the upload process ends when the user confirms it while the dialog closes and a table is rendered with annotation documents' information, such as ID, title, number of annotations, time elapsed since the last update, which goes back to zero when a document's annotations are modified, and a delete icon button. At the same time, during six seconds a snackbar informs how many files were uploaded, as shown in figure 5.7. Figure 5.9 shows what the interface looks like after this process.


5 file(s) uploaded.

Copyright © Your Website 2021.

**Figure 5.7:** Informative snackbar on uploaded files


Upload Files to Annotate

Drag and drop files here or click to browse

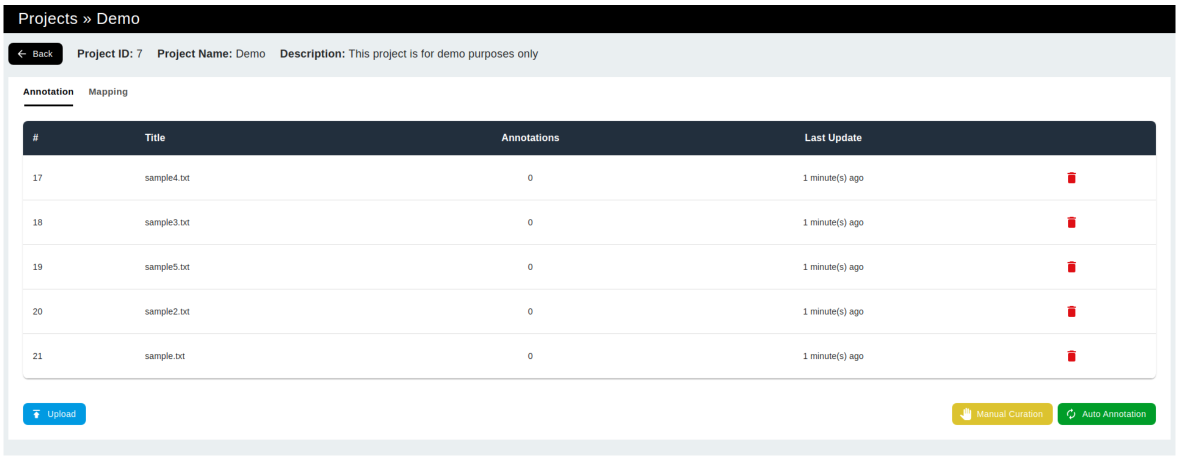
File types allowed: .txt

Max size: 2MB

Files:

- sample.txt

**Figure 5.8:** Dropzone for uploading annotation files

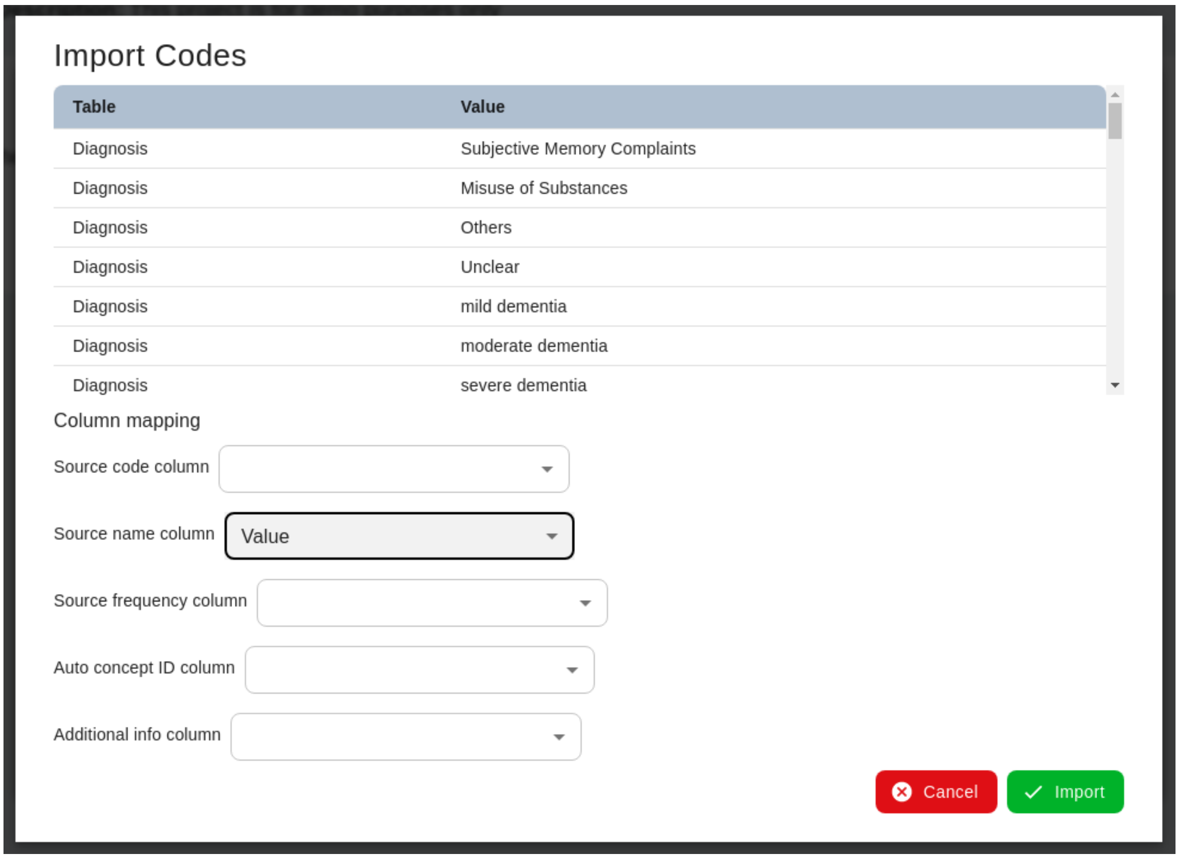

Projects » Demo

**Project ID:** 7 **Project Name:** Demo **Description:** This project is for demo purposes only

| # | Title | Annotations | Last Update |
|---|---|---|---|
| 17 | sample4.txt | 0 | 1 minute(s) ago |
| 18 | sample3.txt | 0 | 1 minute(s) ago |
| 19 | sample5.txt | 0 | 1 minute(s) ago |
| 20 | sample2.txt | 0 | 1 minute(s) ago |
| 21 | sample.txt | 0 | 1 minute(s) ago |

**Figure 5.9:** Individual project's annotation documents

As for the mapping tab, the process has an extra step. The interface when empty looks the same as figure 5.6 and the same dropzone dialog shown in figure 5.8 is prompt when pressing "Upload", but only allows CSV files to be selected or dragged and instead of a "Confirm" there is a "Next" button. This time, when hitting "Next" an additional dialog is prompt to map the CSV columns, because we need to indicate which column represents what, in order to map those imported terms. In that dialog displayed in figure 5.10, the user may configure five parameters by associating each one to a column, including source code, source name, source frequency, auto concept ID and additional info. It is not mandatory nor required to map every single one, even because not every CSV file will hold all the columns, as the figure itself shows. Nevertheless, at least one column is strictly mandatory to be able to finish the upload/import process, which is the source name column because it holds the list of terms to map to standard vocabulary concepts. After clicking "Import" the same behaviour is expected as the annotation tab by rendering the same informative snackbar in figure 5.7 and a table showing the documents' ID, title, number of rows and a delete button icon (see figure 5.11). The delete button presents and triggers the same components and behaviour as the others already described, i.e. prompt a dialog to confirm followed by a snackbar if confirmed. In order to map these imported concepts, the users just need to press the yellow rectangle button "Map Concepts", which will redirect them to the mapping page.

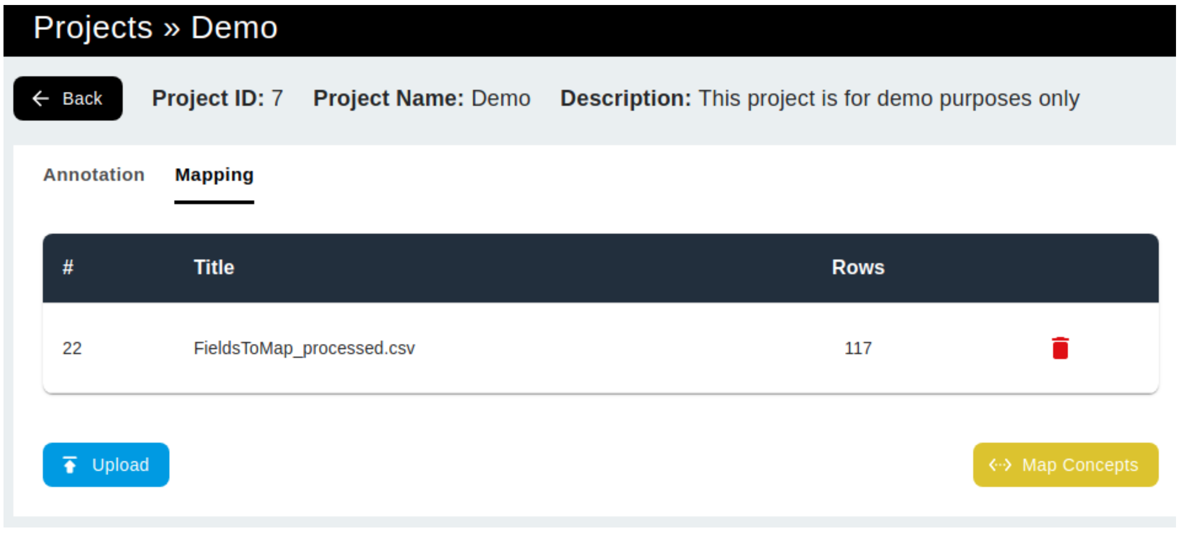

Import Codes

| Table | Value |
|---|---|
| Diagnosis | Subjective Memory Complaints |
| Diagnosis | Misuse of Substances |
| Diagnosis | Others |
| Diagnosis | Unclear |
| Diagnosis | mild dementia |
| Diagnosis | moderate dementia |
| Diagnosis | severe dementia |

Column mapping

| | |
|---|---|
| Source code column | |
| Source name column | Value |
| Source frequency column | |
| Auto concept ID column | |
| Additional info column | |

**Figure 5.10:** Import dialog to map columns


Projects » Demo

Back **Project ID:** 7 **Project Name:** Demo **Description:** This project is for demo purposes only

Annotation **Mapping**

| # | Title | Rows |  |
|---|---|---|---|
| 22 | FieldsToMap_processed.csv | 117 |  |

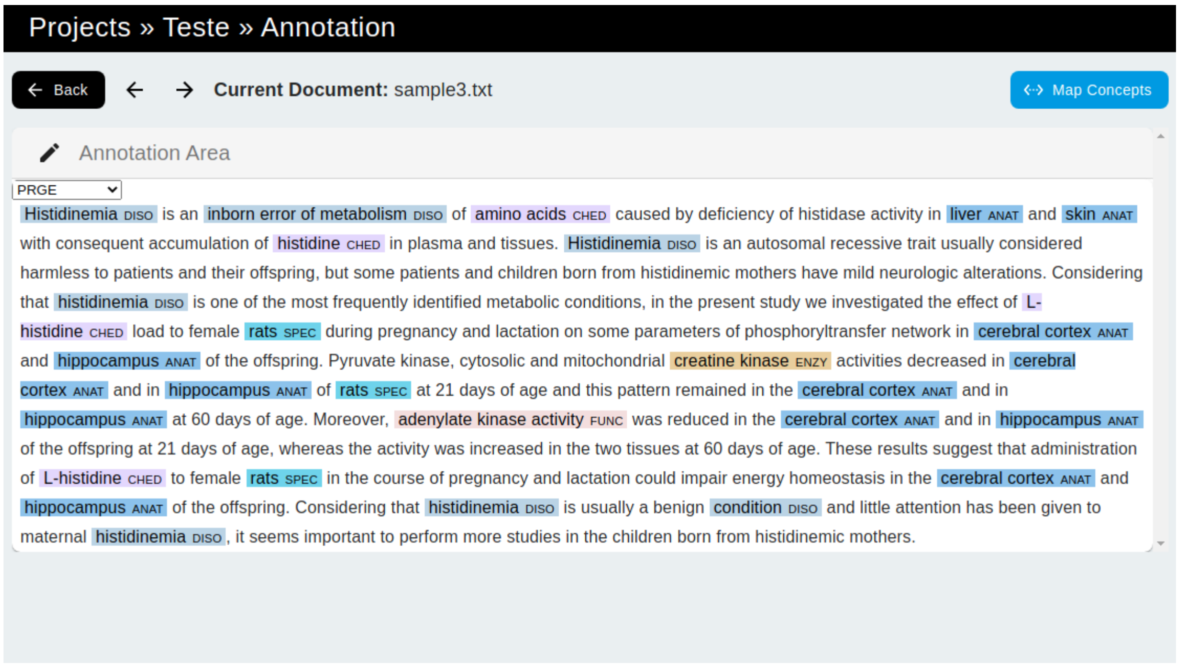
Upload Map Concepts

**Figure 5.11:** Uploaded .CSV documents for mapping

## 5.3 Annotation

In order to start annotating, the user can choose two modes of operation: either manually by clicking the yellow “Manual Curation” button with a hand icon or automatically by hitting the green “Auto Annotation” button, which will lead to the annotation interface. It is important to note that the automatic annotation is only executed once on documents that have not been annotated yet, or in other words, if they have zero annotations recorded in database. The annotation interface is divided into two vertical sections as mentioned in subsection 4.3.3. Figure 5.12 presents an interactive area with highlighted terms as well as their type (e.g. disorder [DISO], species [SPEC], anatomy [ANAT], and so on), where the users add new ones or delete existing ones. To annotate a term the user must choose its type from the dropdown menu, followed by double-clicking a word or selecting a portion of text. Above the annotation area, the user is able to change between documents with navigation arrows which will also change the typography at their right stating which one is selected. In addition, we placed a blue button on the far right of this column in order to allow the users to execute both annotation and mapping as a “pipeline”. It allows for mapping the whole project, i.e. gathers all the annotations from all the project’s documents, filters duplicate entries and maps the resultant unique terms.

On the right column, we render a table with the selected document’s annotations as seen in figure 5.13. When a term is manually annotated it is stored in the database which will trigger the table on the right column to update its content. Notice that some terms have an “undefined” ID, unlike others with actual value, meaning that they were manually annotated thus do not have access to an annotation ID, because that information is output by the automatic procedure. If the users wish to delete an annotation they may do it by clicking on the red bin icon, without prompting a confirmation dialog to make it less intrusive. However, a snackbar is rendered telling which annotation has been deleted and provides an “Undo” option (see figure 5.14, which will last for six seconds. After that period, that annotation can

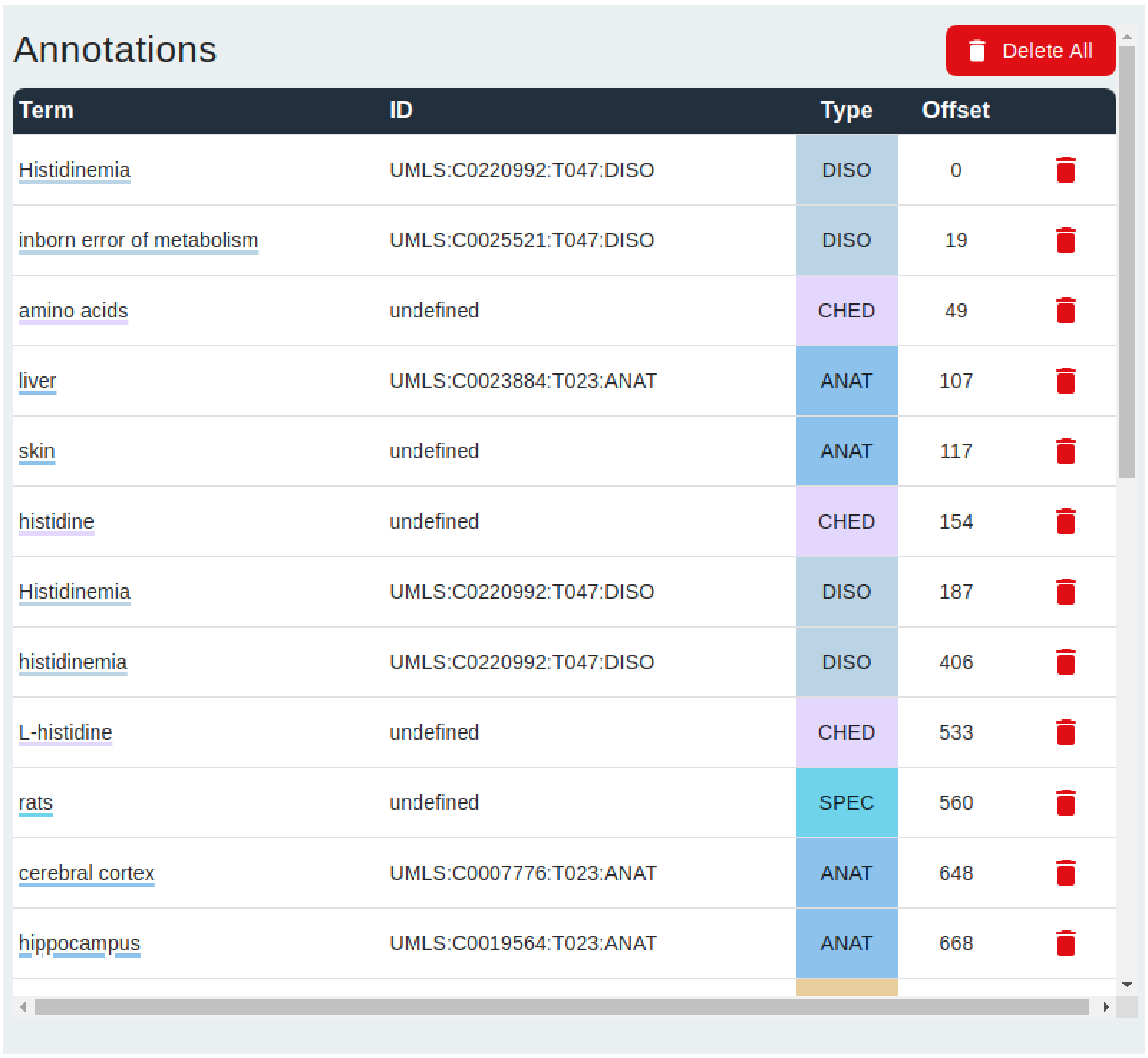


Histidinemia DISO is an inborn error of metabolism DISO of amino acids CHED caused by deficiency of histidase activity in liver ANAT and skin ANAT with consequent accumulation of histidine CHED in plasma and tissues. Histidinemia DISO is an autosomal recessive trait usually considered harmless to patients and their offspring, but some patients and children born from histidinemic mothers have mild neurologic alterations. Considering that histidinemia DISO is one of the most frequently identified metabolic conditions, in the present study we investigated the effect of L-histidine CHED load to female rats SPEC during pregnancy and lactation on some parameters of phosphoryltransfer network in cerebral cortex ANAT and hippocampus ANAT of the offspring. Pyruvate kinase, cytosolic and mitochondrial creatine kinase ENZY activities decreased in cerebral cortex ANAT and in hippocampus ANAT of rats SPEC at 21 days of age and this pattern remained in the cerebral cortex ANAT and in hippocampus ANAT at 60 days of age. Moreover, adenylate kinase activity FUNC was reduced in the cerebral cortex ANAT and in hippocampus ANAT of the offspring at 21 days of age, whereas the activity was increased in the two tissues at 60 days of age. These results suggest that administration of L-histidine CHED to female rats SPEC in the course of pregnancy and lactation could impair energy homeostasis in the cerebral cortex ANAT and hippocampus ANAT of the offspring. Considering that histidinemia DISO is usually a benign condition DISO and little attention has been given to maternal histidinemia DISO, it seems important to perform more studies in the children born from histidinemic mothers.

**Figure 5.12:** Annotation area with highlighted terms

not be retrieved and must be manually added again, if needed.

There is also an option to delete a document's annotations all at once by clicking the red rectangular "Delete All" button on the top right corner, which will prompt a dialog to confirm this action (see figure 5.15). After confirmation, the given document becomes "un-annotated" and automatic annotation is available again.

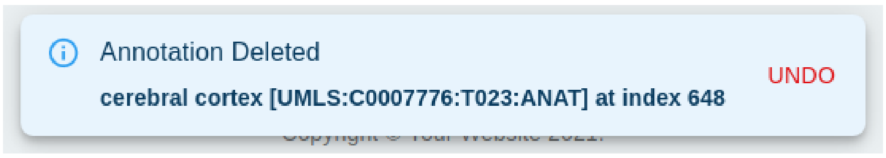


Annotations

Delete All

| Term | ID | Type | Offset | |
|---|---|---|---|---|
| Histidinemia | UMLS:C0220992:T047:DISO | DISO | 0 | |
| inborn error of metabolism | UMLS:C0025521:T047:DISO | DISO | 19 | |
| amino acids | undefined | CHED | 49 | |
| liver | UMLS:C0023884:T023:ANAT | ANAT | 107 | |
| skin | undefined | ANAT | 117 | |
| histidine | undefined | CHED | 154 | |
| Histidinemia | UMLS:C0220992:T047:DISO | DISO | 187 | |
| histidinemia | UMLS:C0220992:T047:DISO | DISO | 406 | |
| L-histidine | undefined | CHED | 533 | |
| rats | undefined | SPEC | 560 | |
| cerebral cortex | UMLS:C0007776:T023:ANAT | ANAT | 648 | |
| hippocampus | UMLS:C0019564:T023:ANAT | ANAT | 668 | |

**Figure 5.13:** Annotations table

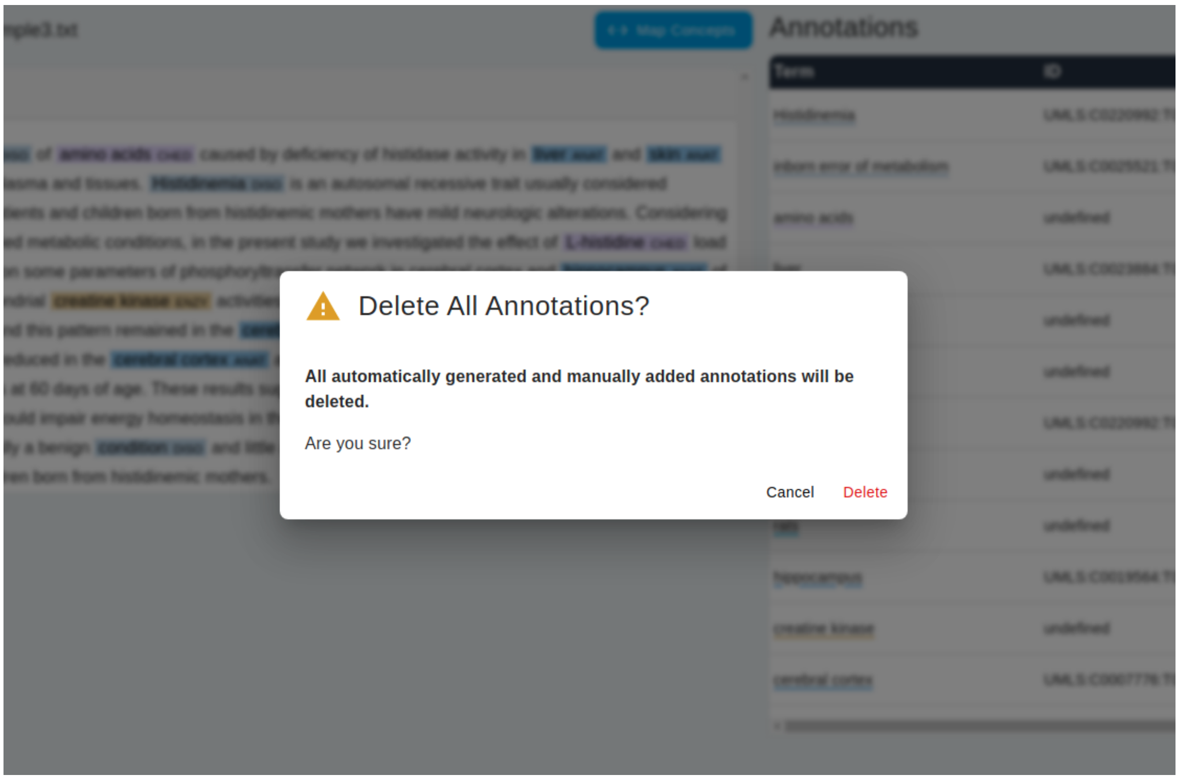


Annotation Deleted

cerebral cortex [UMLS:C0007776:T023:ANAT] at index 648

UNDO

**Figure 5.14:** Annotation deleted snackbar


Delete All Annotations?

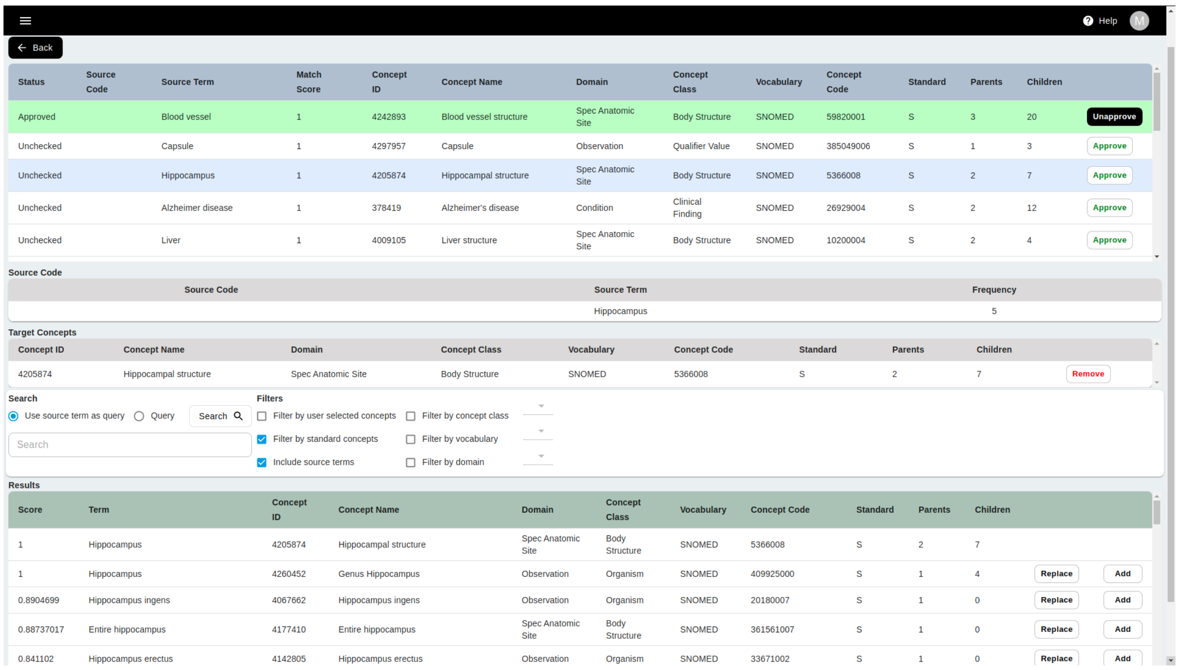
**All automatically generated and manually added annotations will be deleted.**

Are you sure?

**Figure 5.15:** Dialog to delete every annotation from a document

## 5.4 Mapping

Regarding the mapping page, we have seen two distinct ways of navigating towards it. One is by following a "pipeline" that includes both annotation and mapping, which is intended to map annotated concepts from the annotation page. The other one consists of mapping imported concepts without going through the annotation process. As such, the difference is only the input method so we will explain how this interface works for both use cases. Mapping non-standard concepts to standard vocabulary concepts is a computation-intensive task because it is necessary to search and compare concepts with millions of names and synonyms and the majority are irrelevant. Therefore, given that the first step is to search each individual concept, the user must wait a few seconds to see results while everything is processed in the background. As soon as it finishes, the page is rendered and populated with the results separated into three sections (see figure 5.16), as they are described individually below.



| Status | Source Code | Source Term | Match Score | Concept ID | Concept Name | Domain | Concept Class | Vocabulary | Concept Code | Standard | Parents | Children | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Approved | | Blood vessel | 1 | 4242893 | Blood vessel structure | Spec Anatomic Site | Body Structure | SNOMED | 59820001 | S | 3 | 20 | Unapprove |
| Unchecked | | Capsule | 1 | 4297957 | Capsule | Observation | Qualifier Value | SNOMED | 385049006 | S | 1 | 3 | Approve |
| Unchecked | | Hippocampus | 1 | 4205874 | Hippocampal structure | Spec Anatomic Site | Body Structure | SNOMED | 5366008 | S | 2 | 7 | Approve |
| Unchecked | | Alzheimer disease | 1 | 378419 | Alzheimer's disease | Condition | Clinical Finding | SNOMED | 26929004 | S | 2 | 12 | Approve |
| Unchecked | | Liver | 1 | 4009105 | Liver structure | Spec Anatomic Site | Body Structure | SNOMED | 10200004 | S | 2 | 4 | Approve |

**Source Code**

| Source Code | Source Term | Frequency |
|---|---|---|
| | Hippocampus | 5 |

**Target Concepts**

| Concept ID | Concept Name | Domain | Concept Class | Vocabulary | Concept Code | Standard | Parents | Children | |
|---|---|---|---|---|---|---|---|---|---|
| 4205874 | Hippocampal structure | Spec Anatomic Site | Body Structure | SNOMED | 5366008 | S | 2 | 7 | Remove |

**Search**

[✓ Use source term as query] Query | Search

Search

**Filters**

☐ Filter by user selected concepts
☐ Filter by concept class
☑ Filter by standard concepts
☐ Filter by vocabulary
☑ Include source terms
☐ Filter by domain

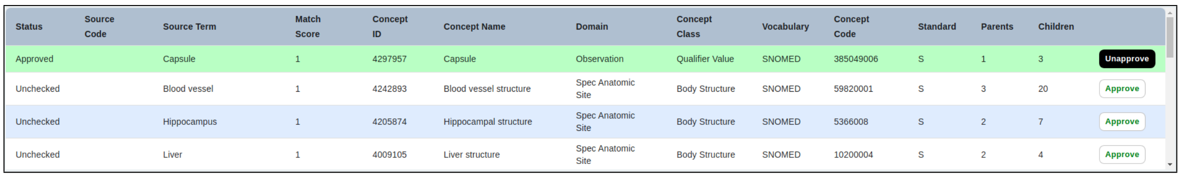
**Results**

| Score | Term | Concept ID | Concept Name | Domain | Concept Class | Vocabulary | Concept Code | Standard | Parents | Children | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Hippocampus | 4205874 | Hippocampal structure | Spec Anatomic Site | Body Structure | SNOMED | 5366008 | S | 2 | 7 | | |
| 1 | Hippocampus | 4260452 | Genus Hippocampus | Observation | Organism | SNOMED | 409925000 | S | 1 | 4 | Replace | Add |
| 0.8904699 | Hippocampus ingens | 4067662 | Hippocampus ingens | Observation | Organism | SNOMED | 20180007 | S | 1 | 0 | Replace | Add |
| 0.88737017 | Entire hippocampus | 4177410 | Entire hippocampus | Spec Anatomic Site | Body Structure | SNOMED | 361561007 | S | 1 | 0 | Replace | Add |
| 0.841102 | Hippocampus erectus | 4142805 | Hippocampus erectus | Observation | Organism | SNOMED | 33671002 | S | 1 | 0 | Replace | Add |

**Figure 5.16:** Mapping interface

Starting at the top, there is a table that displays a scrollable list of the suggested code mappings for each concept. For each concept the tool retrieves dozens or hundreds of results ordered by a match score which represents the confidence, ranging from 0 to 1 with 1 being a confident match. Thus, in this table we display the best results with various informations including status, source code, source term, concept ID, domain, concept class, vocabulary, concept code, standard, number of parents and children, which are all output by the search engine. The user may approve concepts by clicking the "Approve" button, which turns that concept approved and its row changes color to green. This table is very important for other functionalities that influence the rest of the page, for instance it supports mouse click events represented by a blue colored row. This event will change the other sections' information.

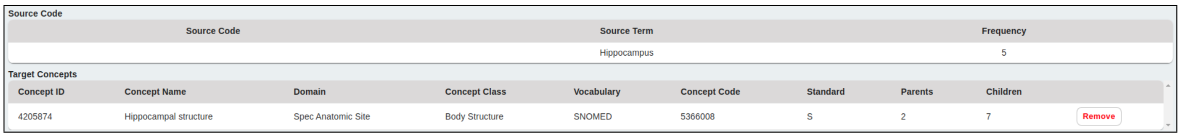
Figure 5.17 shows a closer look at the suggested code mappings.



| Status | Source Code | Source Term | Match Score | Concept ID | Concept Name | Domain | Concept Class | Vocabulary | Concept Code | Standard | Parents | Children | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Approved | | Capsule | 1 | 4297957 | Capsule | Observation | Qualifier Value | SNOMED | 385049006 | S | 1 | 3 | Unapprove |
| Unchecked | | Blood vessel | 1 | 4242893 | Blood vessel structure | Spec Anatomic Site | Body Structure | SNOMED | 59820001 | S | 3 | 20 | Approve |
| Unchecked | | Hippocampus | 1 | 4205874 | Hippocampal structure | Spec Anatomic Site | Body Structure | SNOMED | 5366008 | S | 2 | 7 | Approve |
| Unchecked | | Liver | 1 | 4009105 | Liver structure | Spec Anatomic Site | Body Structure | SNOMED | 10200004 | S | 2 | 4 | Approve |

**Figure 5.17:** Code mappings suggested by the search engine

Directly below that table, in the middle section there are two small tables presenting an overview of the selected concept. The first one contains three columns regarding the source code, source term and frequency of the selected concept from the first table at the top. Notice that only imported CSV files may contain source codes, otherwise that column will be empty at all times and the same applies to the first table. The frequency represents the number of occurrences of the selected concept, either in the annotated documents or in the imported file. The second table usually shows one target concept, which represents the mapping for that concept. However, the user may remove it hitting the red "Remove" button and that concept will become unmapped, or replace it by another or even add a target concept, which will be explained next. Figure 5.18 shows an example of these two tables when "Hippocampus" is selected.



Source Code

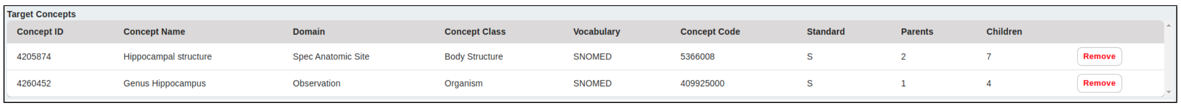
| Source Code | Source Term | Frequency |
|---|---|---|
| | Hippocampus | 5 |

Target Concepts

| Concept ID | Concept Name | Domain | Concept Class | Vocabulary | Concept Code | Standard | Parents | Children | |
|---|---|---|---|---|---|---|---|---|---|
| 4205874 | Hippocampal structure | Spec Anatomic Site | Body Structure | SNOMED | 5366008 | S | 2 | 7 | Remove |

**Figure 5.18:** Source code and target concepts tables for "Hippocampus"

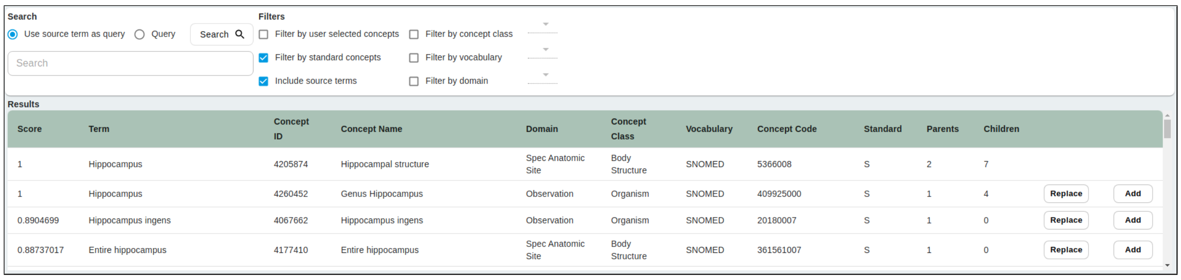
The last section of the mapping interface is a search facility that is subdivided into two portions: one for defining a set of filters and input search queries and a table with results (see figure 5.20). As default, the table shows results relative to the selected concept and contains the same columns as the suggested mappings table at the top. It is also possible to search for specific terms rather than searching the selected concept, by selecting the "Query" option and input the desired term. After hitting the "Search" button, the results are updated. Moreover, the user may filter them by selected concepts, standard concepts, including source terms, concept class, vocabulary or domain, in order to narrow them down. Finally, the user may replace the target concept for a concept from the results or add new ones. Figure 5.19 shows a situation where "Hippocampus" has two target concepts, both with match score of 1.



Target Concepts

| Concept ID | Concept Name | Domain | Concept Class | Vocabulary | Concept Code | Standard | Parents | Children | |
|---|---|---|---|---|---|---|---|---|---|
| 4205874 | Hippocampal structure | Spec Anatomic Site | Body Structure | SNOMED | 5366008 | S | 2 | 7 | Remove |
| 4260452 | Genus Hippocampus | Observation | Organism | SNOMED | 409925000 | S | 1 | 4 | Remove |

**Figure 5.19:** Concept "Hippocampus" with two target concepts



☐ Filter by user selected concepts
☑ Filter by standard concepts
☑ Include source terms
☐ Filter by concept class
☐ Filter by vocabulary
☐ Filter by domain

Results

| Score | Term | Concept ID | Concept Name | Domain | Concept Class | Vocabulary | Concept Code | Standard | Parents | Children | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Hippocampus | 4205874 | Hippocampal structure | Spec Anatomic Site | Body Structure | SNOMED | 5366008 | S | 2 | 7 | | |
| 1 | Hippocampus | 4260452 | Genus Hippocampus | Observation | Organism | SNOMED | 409925000 | S | 1 | 4 | Replace | Add |
| 0.8904699 | Hippocampus ingens | 4067662 | Hippocampus ingens | Observation | Organism | SNOMED | 20180007 | S | 1 | 0 | Replace | Add |
| 0.88737017 | Entire hippocampus | 4177410 | Entire hippocampus | Spec Anatomic Site | Body Structure | SNOMED | 361561007 | S | 1 | 0 | Replace | Add |

**Figure 5.20:** Search results for the selected concept "Hippocampus"

CHAPTER 6

# Conclusions

The focus of this dissertation was to build a state-of-the-art tool to aid the labor-intensive and time-consuming manual tasks of annotation and curation of biomedical literature, as well as provide the ability to convert clinical concepts into a standard format. Several state-of-the-art NLP solutions and techniques were studied to acquire knowledge concerning biomedical TM and how it played a very important role to build GSC. On the other side of the spectrum, it was described the importance of harmonizing data into a common data standard and a successful approach by the open-science collaborative OHDSI, which remains active with different-skilled contributors working collaboratively allowing to improve research and promoting better healthcare.

Taking that into consideration, this work contributes to the community by combining annotation and mapping in a single tool, which allows experts to still perform each operation individually but also to form a pipeline and use the annotation stage output as input for the mapping stage. An overview of the architecture and its detailed implementation were described, including the database planning and configuration using PostgreSQL and Sequelize ORM, the back-end development through Node.js and Express.js that allowed to create a server which handles the operations between the client-side and the database, as well as external services for annotation and mapping, while establishing communications throughout the different components with Axios. The relevant information is displayed in a simple and easy-to-use user interface developed with well-known technologies provided by React.js.

As a result, the tool allows the users to upload text documents and annotate biomedical entities present in them, either manually by selecting portions of text or double clicking words, or automatically with Neji's web services and manage those generated annotations. For "independent" mapping, the users can upload CSV documents containing terms to be mapped to standard vocabulary concepts, using Usagi's open-source code developed by OHDSI which allows to build an index from a vocabulary and search terms on it. Moreover, the users can review and validate suggested mappings based on match score. However, not every requirement was implemented, for instance user registration would be essential to implement the necessary components in order to provide collaboration among experts. Moreover, some operations are not as efficient as expected, due to less orthodox methodologies, namely the mapping search feature because the processing time is directly proportional to the number of terms to search, adding the overhead caused by the React user interface itself which also processes and stores variables in the background. Nevertheless, the solution developed meets most of the objectives that were planned.

## 6.1 Future work

Regarding future work there are several points that would add value to this dissertation, which are enumerated below:

- Currently the users are able to upload text files for annotation, but it would be very useful to implement PDF and even PMC articles directly to promote versatility;
- Give more control over the tool's configuration to the users by creating new interfaces for managing annotation resources, for instance dictionaries and ML models, instead of using Neji's and then return to the tool;
- Export code mappings after the users are finished reviewing and validated all suggested mappings, as well as save their work and continue later on another time;
- Improve the overall efficiency by implementing better algorithms, methodologies and technologies, for example use an in-memory database as cache which would decrease the loading times and responsiveness;
- Implement user registration which in turn would allow to develop collaborative features, such as real-time collaboration between annotators, assuming that the system is deployed so users do not execute it locally.

# References

[1] OHDSI, *The Book of OHDSI: Observational Health Data Sciences and Informatics*. OHDSI, 2021, ISBN: 9781088855195. [Online]. Available: `https://ohdsi.github.io/TheBookOfOhdsi/`.

[2] Wikipedia contributors, *Cohort (statistics) — Wikipedia, the free encyclopedia*, [Online; accessed 26-March-2021], 2020. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=Cohort_(statistics)&oldid=959136074`.

[3] D. P. Lubeck, "Use of observational databases (registries) in research," in *Clinical Research Methods for Surgeons*, Humana Press, pp. 95–104. DOI: `10.1007/978-1-59745-230-4_6`. [Online]. Available: `https://doi.org/10.1007/978-1-59745-230-4_6`.

[4] P. Vassiliadis and A. Simitsis, "Near real time etl," *Annals of Information Systems New Trends in Data Warehousing and Data Analysis*, pp. 1–31, 2008. DOI: `10.1007/978-0-387-87431-9_2`.

[5] *Data warehouse*, Sep. 2021. [Online]. Available: `https://en.wikipedia.org/wiki/Data_warehouse`.

[6] S. Chaudhuri and U. Dayal, "An overview of data warehousing and olap technology," *ACM SIGMOD Record*, vol. 26, no. 1, pp. 65–74, 1997. DOI: `10.1145/248603.248616`.

[7] T. Suzumura, T. Yasue, and T. Onodera, "Scalable performance of system s for extract-transform-load processing," *Proceedings of the 3rd Annual Haifa Experimental Systems Conference on - SYSTOR 10*, 2010. DOI: `10.1145/1815695.1815704`.

[8] A. D. U. o. Illinois, A. Doan, U. o. Illinois, U. o. I. Profile, R. R. U. o. Wisconsin, R. Ramakrishnan, U. o. Wisconsin, U. o. W. Profile, S. V. I. R. at Almaden, S. Vaithyanathan, and et al., *Managing information extraction: State of the art and research directions*, Jun. 2006. [Online]. Available: `https://dl.acm.org/doi/abs/10.1145/1142473.1142595`.

[9] M. Hearst, "What is text mining," *SIMS, UC Berkeley*, vol. 5, 2003.

[10] R. J. Gaizauskas and A. M. Robertson, "Coupling information retrieval and information extraction: A new text technology for gathering information from the web.," Citeseer.

[11] A. Mansouri, L. S. Affendey, and A. Mamat, "Named entity recognition approaches," *International Journal of Computer Science and Network Security*, vol. 8, no. 2, pp. 339–344, 2008.

[12] H. Shelar, G. Kaur, N. Heda, and P. Agrawal, "Named entity recognition approaches and their comparison for custom ner model," *Science & Technology Libraries*, vol. 39, no. 3, pp. 324–337, 2020. DOI: `10.1080/0194262X.2020.1759479`. eprint: `https://doi.org/10.1080/0194262X.2020.1759479`. [Online]. Available: `https://doi.org/10.1080/0194262X.2020.1759479`.

[13] G. Zhou, J. Zhang, J. Su, D. Shen, and C. Tan, "Recognizing names in biomedical texts: a machine learning approach," *Bioinformatics*, vol. 20, no. 7, pp. 1178–1190, Feb. 2004, ISSN: 1367-4803. DOI: `10.1093/bioinformatics/bth060`. eprint: `https://academic.oup.com/bioinformatics/article-pdf/20/7/1178/679155/bth060.pdf`. [Online]. Available: `https://doi.org/10.1093/bioinformatics/bth060`.

[14] D. Campos, S. Matos, and J. Luis, "Biomedical named entity recognition: A survey of machine-learning tools," *Theory and Applications for Advanced Text Mining*, 2012. DOI: `10.5772/51066`.

[15] L. Wissler, M. Almashraee, D. Monett, and A. Paschke, "The gold standard in corpus annotation," Jun. 2014. DOI: `10.13140/2.1.4316.3523`.

[16] S. Sun, C. Luo, and J. Chen, "A review of natural language processing techniques for opinion mining systems," *Information Fusion*, vol. 36, pp. 10–25, 2017, ISSN: 1566-2535. DOI: https://doi.org/10.1016/j.inffus.2016.10.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1566253516301117.

[17] *Romance languages*, Oct. 2021. [Online]. Available: https://en.wikipedia.org/wiki/Romance_languages.

[18] C. Silva and B. Ribeiro, "The importance of stop word removal on recall values in text categorization," in *Proceedings of the International Joint Conference on Neural Networks, 2003.*, vol. 3, Jul. 2003, 1661–1666 vol.3. DOI: 10.1109/IJCNN.2003.1223656.

[19] V. Balakrishnan and L.-Y. Ethel, "Stemming and lemmatization: A comparison of retrieval performances," *Lecture Notes on Software Engineering*, vol. 2, no. 3, pp. 262–267, 2014. DOI: 10.7763/lnse.2014.v2.134.

[20] A. Voutilainen, "Part-of-speech tagging," *The Oxford handbook of computational linguistics*, pp. 219–232, 2003.

[21] B. Rink, C. A. Bejan, and S. Harabagiu, "Learning textual graph patterns to detect causal event relations," in *Twenty-Third International FLAIRS Conference*, 2010.

[22] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

[23] C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky, "The stanford corenlp natural language processing toolkit," in *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, 2014, pp. 55–60.

[24] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 conference on empirical methods in natural language processing*, 2013, pp. 1631–1642.

[25] R. Řehůřek and P. Sojka, "Software framework for topic modelling with large corpora. english," in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pp. 45–50.

[26] X. Qiu, Q. Zhang, and X.-J. Huang, "Fudannlp: A toolkit for chinese natural language processing," in *Proceedings of the 51st annual meeting of the association for computational linguistics: system demonstrations*, 2013, pp. 49–54.

[27] W. Che, Z. Li, and T. Liu, "Ltp: A chinese language technology platform," in *Coling 2010: Demonstrations*, 2010, pp. 13–16.

[28] J. Zhu, M. Zhu, Q. Wang, and T. Xiao, "Niuparser: A chinese syntactic and semantic parsing toolkit," in *Proceedings of ACL-IJCNLP 2015 System Demonstrations*, 2015, pp. 145–150.

[29] M. Neves and U. Leser, "A survey on annotation tools for the biomedical literature," *Briefings in Bioinformatics*, vol. 15, no. 2, pp. 327–340, Dec. 2012, ISSN: 1467-5463. DOI: 10.1093/bib/bbs084. eprint: https://academic.oup.com/bib/article-pdf/15/2/327/555100/bbs084.pdf. [Online]. Available: https://doi.org/10.1093/bib/bbs084.

[30] Mariananeves, *Mariananeves/annotation-tools.* [Online]. Available: https://github.com/mariananeves/annotation-tools.

[31] M. Neves and J. Ševa, "An extensive review of tools for manual annotation of documents," *Briefings in Bioinformatics*, vol. 22, no. 1, pp. 146–163, Dec. 2019, ISSN: 1477-4054. DOI: 10.1093/bib/bbz130. eprint: https://academic.oup.com/bib/article-pdf/22/1/146/35934686/bbz130.pdf. [Online]. Available: https://doi.org/10.1093/bib/bbz130.

[32] M. Neves and J. Seva, *Annotationsaurus: A searchable directory of annotation tools*, 2020. arXiv: 2010.06251 [cs.CL].

[33] E. F. Sang and F. De Meulder, "Introduction to the conll-2003 shared task: Language-independent named entity recognition," *arXiv preprint cs/0306050*, 2003.

[34] A. Lourenço, R. Carreira, S. Carneiro, P. Maia, D. Glez-Peña, F. Fdez-Riverola, E. C. Ferreira, I. Rocha, and M. Rocha, "@note: A workbench for biomedical text mining," *Journal of Biomedical Informatics*, vol. 42, no. 4, pp. 710–720, 2009, ISSN: 1532-0464. DOI: https://doi.org/10.1016/j.jbi.2009.04.002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1532046409000537.

[35] R. Rak, A. Rowley, W. Black, and S. Ananiadou, "Argo: an integrative, interactive, text mining-based workbench supporting curation," *Database*, vol. 2012, Feb. 2012, bas010, ISSN: 1758-0463. DOI: 10.1093/database/bas010. eprint: https://academic.oup.com/database/article-pdf/doi/10.1093/database/bas010/1192080/bas010.pdf. [Online]. Available: https://doi.org/10.1093/database/bas010.

[36] C. Cano, T. Monaghan, A. Blanco, D. Wall, and L. Peshkin, "Collaborative text-annotation resource for disease-centered relation extraction from biomedical text," *Journal of Biomedical Informatics*, vol. 42, no. 5, pp. 967–977, 2009, Biomedical Natural Language Processing, ISSN: 1532-0464. DOI: https://doi.org/10.1016/j.jbi.2009.02.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1532046409000215.

[37] D. Kwon, S. Kim, S.-Y. Shin, A. Chatr-aryamontri, and W. J. Wilbur, "Assisting manual literature curation for protein–protein interactions using BioQRator," *Database*, vol. 2014, Jul. 2014, bau067, ISSN: 1758-0463. DOI: 10.1093/database/bau067. eprint: https://academic.oup.com/database/article-pdf/doi/10.1093/database/bau067/8246754/bau067.pdf. [Online]. Available: https://doi.org/10.1093/database/bau067.

[38] P. Stenetorp, S. Pyysalo, G. Topić, T. Ohta, S. Ananiadou, and J. Tsujii, "Brat: A web-based tool for NLP-assisted text annotation," in *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*, Avignon, France: Association for Computational Linguistics, Apr. 2012, pp. 102–107. [Online]. Available: https://aclanthology.org/E12-2021.

[39] E. Apostolova, S. Neilan, G. An, N. Tomuro, and S. Lytinen, "Djangology: A light-weight web-based tool for distributed collaborative text annotation," in *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta: European Language Resources Association (ELRA), May 2010. [Online]. Available: http://www.lrec-conf.org/proceedings/lrec2010/pdf/543_Paper.pdf.

[40] P. Ciccarese, M. Ocana, and T. Clark, "Open semantic annotation of scientific publications using domeo," *Journal of Biomedical Semantics*, vol. 3, no. S1, 2012. DOI: 10.1186/2041-1480-3-s1-s1.

[41] D. Campos, J. Lourenço, S. Matos, and J. L. Oliveira, "Egas: a collaborative and interactive document curation platform," *Database*, vol. 2014, Jun. 2014, bau048, ISSN: 1758-0463. DOI: 10.1093/database/bau048. eprint: https://academic.oup.com/database/article-pdf/doi/10.1093/database/bau048/8245970/bau048.pdf. [Online]. Available: https://doi.org/10.1093/database/bau048.

[42] D. Kwon, S. Kim, C.-H. Wei, R. Leaman, and Z. Lu, "ezTag: tagging biomedical concepts via interactive learning," *Nucleic Acids Research*, vol. 46, no. W1, W523–W529, May 2018, ISSN: 0305-1048. DOI: 10.1093/nar/gky428. eprint: https://academic.oup.com/nar/article-pdf/46/W1/W523/25110520/gky428.pdf. [Online]. Available: https://doi.org/10.1093/nar/gky428.

[43] R. Leaman and Z. Lu, "TaggerOne: joint named entity recognition and normalization with semi-Markov Models," *Bioinformatics*, vol. 32, no. 18, pp. 2839–2846, Jun. 2016, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btw343. eprint: https://academic.oup.com/bioinformatics/article-pdf/32/18/2839/24406872/btw343.pdf. [Online]. Available: https://doi.org/10.1093/bioinformatics/btw343.

[44] C.-H. Wei, H.-Y. Kao, and Z. Lu, "Gnormplus: An integrative approach for tagging genes, gene families, and protein domains," *BioMed research international*, vol. 2015, 2015.

[45] C.-H. Wei, B. R. Harris, H.-Y. Kao, and Z. Lu, "tmVar: a text mining approach for extracting sequence variants in biomedical literature," *Bioinformatics*, vol. 29, no. 11, pp. 1433–1439, Apr. 2013, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btt156. eprint: https://academic.oup.com/bioinformatics/article-pdf/29/11/1433/587543/btt156.pdf. [Online]. Available: https://doi.org/10.1093/bioinformatics/btt156.

[46] K. Bontcheva, H. Cunningham, I. Roberts, A. Roberts, V. Tablan, N. Aswani, and G. Gorrell, "Gate teamware: A web-based, collaborative text annotation framework," *Language Resources and Evaluation*, vol. 47, no. 4, pp. 1007–1029, 2013. DOI: 10.1007/s10579-013-9215-6.

[47] H. Cunningham, D. Maynard, and K. Bontcheva, *Text Processing with GATE.* Gateway Press CA, 2011, ISBN: 0956599311.

[48] D. Salgado, M. Krallinger, M. Depaule, E. Drula, A. V. Tendulkar, F. Leitner, A. Valencia, and C. Marcelle, "MyMiner: a web application for computer-assisted biocuration and text annotation," *Bioinformatics*, vol. 28, no. 17, pp. 2285–2287, Jul. 2012, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bts435. eprint: https://academic.oup.com/bioinformatics/article-pdf/28/17/2285/16905276/bts435.pdf. [Online]. Available: https://doi.org/10.1093/bioinformatics/bts435.

[49] *Uniprot*, Oct. 2021. [Online]. Available: https://en.wikipedia.org/wiki/UniProt.

[50] F. Rinaldi, S. Clematide, H. Marques, T. Ellendorff, M. Romacker, and R. Rodriguez-Esteban, "Ontogene web services for biomedical text mining," *BMC Bioinformatics*, vol. 15, no. S14, 2014. DOI: 10.1186/1471-2105-15-s14-s6.

[51] C.-H. Wei, H.-Y. Kao, and Z. Lu, "PubTator: a web-based text mining tool for assisting biocuration," *Nucleic Acids Research*, vol. 41, no. W1, W518–W522, May 2013, ISSN: 0305-1048. DOI: 10.1093/nar/gkt441. eprint: https://academic.oup.com/nar/article-pdf/41/W1/W518/3859973/gkt441.pdf. [Online]. Available: https://doi.org/10.1093/nar/gkt441.

[52] M. Huang, J. Liu, and X. Zhu, "Genetukit: A software for document-level gene normalization," *Bioinformatics*, vol. 27, no. 7, pp. 1032–1033, 2011.

[53] C.-H. Wei and H.-Y. Kao, "Cross-species gene normalization by species inference," *BMC bioinformatics*, vol. 12, no. 8, pp. 1–11, 2011.

[54] C.-H. Wei, H.-Y. Kao, and Z. Lu, "Sr4gn: A species recognition software tool for gene normalization," *PloS one*, vol. 7, no. 6, e38460, 2012.

[55] R. Leaman, R. Islamaj Doğan, and Z. Lu, "DNorm: disease name normalization with pairwise learning to rank," *Bioinformatics*, vol. 29, no. 22, pp. 2909–2917, Aug. 2013, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btt474. eprint: https://academic.oup.com/bioinformatics/article-pdf/29/22/2909/888873/btt474.pdf. [Online]. Available: https://doi.org/10.1093/bioinformatics/btt474.

[56] T. C. Wiegers, A. P. Davis, and C. J. Mattingly, "Collaborative biocuration—text-mining development task for document prioritization for curation," *Database*, vol. 2012, Nov. 2012, bas037, ISSN: 1758-0463. DOI: 10.1093/database/bas037. eprint: https://academic.oup.com/database/article-pdf/doi/10.1093/database/bas037/1202655/bas037.pdf. [Online]. Available: https://doi.org/10.1093/database/bas037.

[57] R. Islamaj, D. Kwon, S. Kim, and Z. Lu, "TeamTat: a collaborative text annotation tool," *Nucleic Acids Research*, vol. 48, no. W1, W5–W11, May 2020, ISSN: 0305-1048. DOI: 10.1093/nar/gkaa333. eprint: https://academic.oup.com/nar/article-pdf/48/W1/W5/33433452/gkaa333.pdf. [Online]. Available: https://doi.org/10.1093/nar/gkaa333.

[58] D. Campos, S. Matos, and J. L. Oliveira, "A modular framework for biomedical concept recognition," *BMC Bioinformatics*, vol. 14, no. 1, 2013. DOI: 10.1186/1471-2105-14-281.

[59] F. Landragin, T. Poibeau, and B. Victorri, "ANALEC: a New Tool for the Dynamic Annotation of Textual Data," in *International Conference on Language Resources and Evaluation (LREC 2012)*, E. L. R. A. (ELRA), Ed., Istanbul, Turkey, May 2012, pp. 357–362. [Online]. Available: https://halshs.archives-ouvertes.fr/halshs-00698971.

[60] A. Przepiórkowski and G. Murzynowski, "Manual annotation of the National Corpus of Polish with Anotatornia," in *The proceedings of Practical Applications in Language and Computers PALC 2009*, S. Goźdź-Roszkowski, Ed., Forthcoming, Frankfurt am Main: Peter Lang, 2009.

[61] A. Widlöcher and Y. Mathet, "The glozz platform: A corpus annotation and mining tool," in *Proceedings of the 2012 ACM Symposium on Document Engineering*, ser. DocEng '12, Paris, France: Association for Computing Machinery, 2012, pp. 171–180, ISBN: 9781450311168. DOI: 10.1145/2361354.2361394. [Online]. Available: https://doi.org/10.1145/2361354.2361394.

[62] M. Tesconi, F. Ronzano, S. Minutoli, C. Aliprandi, and A. Marchetti, "Kafnotator: A multilingual semantic text annotation tool," in *The Second International Conference on Global Interoperability for Language Resources*, vol. 1, 2010.

[63] A. Burchardt, K. Erk, A. Frank, A. Kowalski, and S. Pado, “SALTO - a versatile multi-level annotation tool,” in *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC’06)*, Genoa, Italy: European Language Resources Association (ELRA), May 2006. [Online]. Available: `http://www.lrec-conf.org/proceedings/lrec2006/pdf/341_pdf.pdf`.

[64] T. Daudert, “A web-based collaborative annotation and consolidation tool,” English, in *Proceedings of the 12th Language Resources and Evaluation Conference*, Marseille, France: European Language Resources Association, May 2020, pp. 7053–7059, ISBN: 979-10-95546-34-4. [Online]. Available: `https://aclanthology.org/2020.lrec-1.872`.

[65] É. V. de La Clergerie, “A collaborative infrastructure for handling syntactic annotations,” in *proc. of The First Workshop on Automated Syntactic Annotations for Interoperable Language Resources*, 2008.

[66] X. Artola, A. D. D. Ilarraza, N. Ezeiza, K. Gojenola, A. Sologaistoa, and A. Soroa, *Eulia: A graphical web interface for creating, browsing and editing linguistically annotated corpora*, 2004.

[67] J. Sonntag and M. Stede, “GraPAT: A tool for graph annotations,” in *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC’14)*, Reykjavik, Iceland: European Language Resources Association (ELRA), May 2014, pp. 4147–4151. [Online]. Available: `http://www.lrec-conf.org/proceedings/lrec2014/pdf/824_Paper.pdf`.

[68] H. Shindo, Y. Munesada, and Y. Matsumoto, “Pdfanno: A web-based linguistic annotation tool for pdf documents,” in *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[69] *Using the application functions.* [Online]. Available: `http://ohdsi.github.io/Usagi/usage.html`.