

ALMA MATER STUDIORUM - UNIVERSITA' DI BOLOGNA

Second Cycle Degree Programme in

Digital Humanities and Digital Knowledge

DISSERTATION TITLE

Flying Digital Editions:

A Tool for Enhancing Digital Text Exploration through Wikidata

Final Dissertation in

Information Modeling and Web Technologies

Supervisor: Prof. Fabio Vitali

Co-supervisor: PhD s. Andrea Schimmenti

Presented by: Sara Vellone (Id. 1056935)

Session III

Academic Year

2022-2023

TABLE OF CONTENTS

ABSTRACT	4
INTRODUCTION	5
1. State of the Art and Related Works	9
1.1 Entity Linking and Linked Data	9
1.2 Knowledge Bases	10
1.2.1 DBpedia	10
1.2.2 YAGO	11
1.2.3 Wikidata	11
1.3 Information Panels	13
1.4 Existing Tools	15
1.4.1 Wikidata API SandBox	16
1.4.2 Stanza	19
1.4.3 Wikifier	21
1.4.4 OpenTapioca	23
1.4.5 Falcon 2.0	25
1.4.6 TagMe	28
1.4.7 DBPedia Spotlight	30
1.4.8 entity-fishing	32
1.4.9 ReFinEd	34
1.5 Preliminary observations	37
2. Development	40
2.1 Setup	40
2.2 Code organisation	46
2.2.1 Server-side	46
2.2.2 Client-side	54
2.3 A new reading experience	62
2.3.1 FDE mocked demo	74

3. Results	77
4. Conclusions	85
5. Future Works.....	89
TABLE OF FIGURES	93
REFERENCES.....	95

ABSTRACT

Flying Digital Editions (FDE) integrates technologies, standards, and tools to enhance the publication steps of digital editions. It leverages the Semantic Web to facilitate interaction with texts, connecting them to external knowledge databases like Wikidata and Wikipedia. This integration allows for enriched text exploration through knowledge panels and links, utilizing RDF and XML-TEI for data integration. FDE features AI (Artificial Intelligence) models for Named Entity Recognition and Entity Linking, alongside APIs for information retrieval, offering a platform for error correction and data curation by users. The result is a tool which enables the visualisation and exploration of trustful information about entities inside a given text. Through this innovative reuse of open data and AI, FDE invites users into a dynamic space where a text can uncover its internal knowledge by itself with user curatorship.

INTRODUCTION

In a world totally geared towards digital, people are always looking for new ways to delve into texts, to give them new expressions and create new ‘editions’, namely the forms in which a text is published (Merriam-Webster 2024).

In *[What is a Scholarly Digital Edition?](#)* (Driscoll and Pierazzo 2016), Patrick Sahle points out the incredible opportunity offered by this expression, capable of overcoming the limitation of the print technology. In explaining what a ‘scholarly digital edition’ is, Patrick Sahle tries first to provide a punctual definition of ‘digital edition’, stating that ‘A digitalised edition is not a digital edition’ and that ‘A digital edition cannot be given in print without significant loss of content and functionalities.’ (Driscoll and Pierazzo 2016, 27). Hence, transforming a text into a digitalized document is not enough to consider it ‘digital’, because it should provide meaningful information that would be lost if the document were printed.

Moreover, a digital edition is a particular form of edition because it requires transversal skills, from computer science competences to philological ones. By exploring The Catalogue of Digital Editions¹, it has been noted that the life cycle of a digital edition is based on the following features:

- Text encoding and markup;
- Interoperability and shareability;
- Critical apparatus;
- Hyperlinking and cross-referencing;
- Search and analysis tools.

Depending on specific context they can also provide user interaction and collaboration, accessibility features, versioning and variants representation (Driscoll and Pierazzo, 2016).

In the past years, several types of tools have been developed to both allow the creation of digitalized and digital editions, by transforming printed texts into electronic documents and by enabling the possibility of manually annotating parts of a text. An example of this tools is given by Oxygen XML Editor², which is used for creating, editing, and managing XML documents.

¹ <https://dig-ed-cat.acdh.oeaw.ac.at/>

² <https://www.oxygenxml.com/>

In most cases, these tools are not thought to be used for everyday needs, because they usually are designed to be implemented inside other systems and to perform technical or specific tasks. For instance, the EVT (Edition Visualization Technology)³ software provides the visualisation and the reading of digital editions in a digital environment, allowing the users to compare the different editions of a text, such as the diplomatic and the interpretative one. Other systems, such as ResearchSpace⁴, provides instead semantic tools to create, analyse and visualise data, but they require a considerable effort and competence.

For what concern the tools that performs text annotation such as entity detection and linking, they often lack interaction with the user. Indeed, when the result is returned, they do not provide information about the found entities, forcing the users to exit the page, and be redirected to – usually – the corresponding Wikipedia page.

Many people, such as teachers, often are not aware of these kinds of tools because their usage is conditioned by computer skills that sometimes they do not have.

By starting from previously mentioned features, this experimental project can be divided into four main phases that respond to three questions: can a Digital Edition be produced from a text automatically? At what expense and in which contexts is it possible to do so? Is the output of this process useful?

Firstly, this research tried to group all the resource already available and to compare them, for choosing the ones that are more suitable for the purposes of the project.

Hence, the first step was searching for the existing knowledge bases. The attention fell on three main ones - DBpedia, YAGO and Wikidata - which have been compared among them for understanding their characteristics and their usefulness if employed in this project.

Then, the problem was to understand if with these knowledge bases, we can perform automatic entity detection and linking processes. The research investigated what tools are available to accomplish this task: as mentioned above, several systems have been developed, therefore a comparative study has been conducted for selecting the most convenient. They are mainly based on Wikipedia, others on DBpedia and few on Wikidata. Particularly, the criteria used to judge these tools were their usability by non-expert users and how these tools are made available for using them on a daily basis. For this

³ <https://www.labcd.unipi.it/progetti/evt-edition-visualization-technology/>

⁴ <https://researchspace.org/semantic-tools/>

reason, to test the usefulness in uncommonly familiar contexts, the documents to be annotated were about astronomy, and they were of different times. This choice has been made for testing the efficiency with texts that might be considered as ‘difficult’ because of what they concern and to understand how this tools deal with these kinds of fields.

The third phase was about founding a way to display all the information that are retrieved from the knowledge base to the user. It was the major task, because it was important to realise what a user wishes to see and know about a particular entity belonging to a particular class. The *Google Knowledge Panel* was almost an easy choice to be inspired by, due to their characteristic and the familiarity that the users usually have with it. It contains all the basic information that one could be interested in, displayed in an understandable way and to be easy to interpret. The attempt was also to add other kind of information from a digital humanities point of view, including – for example – the possibility of downloading turtle files of the collected data, and the reference to the VIAF ID, if any.

Finally, the fourth phase consisted in the creation of a tool that, by grouping all the tasks that one is often forced to do separately, provides a more interactive experience while maintaining an ease of use and a minimal interface. It was created using recent technologies such as the Angular framework and Python APIs and taking certain steps to ensure a complete exploration of the text, such as the creation of a ‘facets panel’, a particular way to search within a text.

Thus, *Flying Digital Editions* (FDE) is a project that has the goal to generate a lightweight digital edition on the fly, starting from a simple text, adding the ‘significant information’ that will enrich the comprehension and the reading of the text itself, up to its visualisation. By reusing existing tools, FDE attempts to simplify and reduce the long process of creating and managing a new digital edition, challenging the possible difficulties of such a combination. The possibility to do all of this ‘on-the-fly’ means that users do not have to move from the page they are on, by having the results they are looking for just a few clicks away, all included in a single flow.

The user can upload a text or a document that will be analysed by automatically recognising the entities using a Natural Language Processing library, called ReFinEd. FDE provides, despite other entity linking tools, to edit the entities retrieved from this process, so that the users are in control over what has to be visualised. They can decide if keeping all the found entities, delete, or modify them, but also to add new ones, by simply providing the Wikidata ID and selecting the class to which the

entity belongs. In fact, by using SPARQL queries templates, the data is retrieved from Wikidata, which is a knowledge base that guarantees up to and reliable information, thanks both to the validity of the community agreements and to the presence of the data's sources for most of the items (Färber et al., 2015). Finally, it gives the possibility to navigate the entities using panels that summarise the most important facts about the specific entity, without requiring any additional action.

For personal use, teaching or development does not matter, because it has been thought to be employed both by expert and inexperienced users, by giving a new experience in reading and exploring any kind of text. Even if sacrificing some accuracy and precision in defining a new digital edition, opening this type of knowledge the public could facilitate their studies and interests, because it would allow to delve into all kinds of documents.

In addition, the extreme flexibility of the tool itself allows to customize every feature that it provides, because each part can be enhanced, modified, or completely changed by applying little adjustments to the other parts of the code, while maintaining the core stable and working. Even the model for the text analysis can be easily substituted with any other by respecting a few simple constants that must be maintained for proper entity management (see [Chapter 2](#)).

Summarising, the Chapter 1 will provide an overview of the current state of the art and the related works, as well as some initial considerations about what found in this first searching phase. Following, Chapter 2 will describe more deeply how FDE has been thought and its functionalities, by providing a punctual documentation of the code of the project, with all the commands for running the tool on premises. Chapter 3 focuses on the results achieved from the development of the tool, its advantages and the difficulties encountered. Chapter 4 summarises the core of the project, its aims and uses, and Chapter 5 will make a detailed overview of the possible future implementations of the tool.

1. State of the Art and Related Works

In this chapter, the reader can find an overview of the current state of the art for each of the main features of the tool, which are: the automatic detection of the entities through the process of entity-linking, the knowledge base from which data can be extracted, the way to visualise the retrieved information, and finally the tools that performs text annotation presently available online.

1.1 Entity Linking and Linked Data

‘Flying Digital Edition’ is a project which puts down its roots in the idea that plain text can be processed into a light-readable digital edition. To achieve this purpose, the text must be deeply analysed, and the implicit knowledge contained in the text must be drawn out and made explicit for the user. In the digital world, this extraction turns into what is called ‘entity linking’ - or entity resolution –, which is the process of mapping a discourse entity to some real-world individual (Martin and Jurafsky 2024, 482–83).

These words of interest are called ‘named entities’ or ‘mentions’ and can be defined as anything that can be referred to a proper name, such as a person, a location, or an organization. Nevertheless, the term is also extended to include things such as dates, times and other types of expressions that are not entities per se. (Martin and Jurafsky 2024, 167).

Entity-linking process usually involves three sub-tasks (Devopedia 2021):

- Information extraction from unstructured data;
- *Named Entity Recognition*, i.e. finding spans inside the text that constitute proper names and tagging the entity’s type (Martin and Jurafsky 2024, 167);
- *Named Entity Linking*, i.e. alignment between a Named Entity and an entry of the same entity from a knowledge base (both as an authority control and to fetch more information).

Part of this process is the ‘Disambiguation’ phase (or *mention disambiguation*), which is also one of the key issues of entity linking and consists in resolving the ambiguity that could arise from named entities in the text when they can have multiple meanings (Coppola 2023).

For easily linking structured data on the Web, there are some ‘best practices’ that can be followed. These practices are known as *Linked Data* and rely on documents that contain data in an RDF format (Berners-Lee, Bizer, and Heath 2009). RDF stands for ‘Resource Description Format,’ a standard

model used for data interchange on the Web. By using URIs to identify the relationship between objects and the ‘two ends of the link’ - the so-called ‘triple’-, RDF expands the Web’s linking structure, sharing structured and semi-structured data across different applications (‘RDF - Semantic Web Standard’ 2014). Linked Data principles and the entity-linking process are strictly connected because the first gives to the second the infrastructure useful for representing interconnecting data, while the EL process connect the entities in the document to the defined entities in the infrastructure itself. Thus, entity-linking exploits the Linked Data technologies for improving the usability of information.

A subset of these procedures is called Linked *Open Data*⁵. This slightly different terminology means that the Linked Data principles are extended by using open licenses⁶ that grant permission to make use of a work at no cost, allowing modifications with minimal – or no – restrictions.

1.2 Knowledge Bases

The term ‘knowledge base’ (KB) was coined to distinguish this type of knowledge store from the more common ‘database’. It is a technology used to deposit complex structured data (‘Wikidata’ 2024) and represent a dynamic machine-readable resource. In the next paragraphs, three KBs freely accessible and usable will be presented, briefly discussed, and compared. In particular, the attention is focused on the quantity of items contained in each knowledge base, the validity of the information provided (e.g., the presence of the source), the temporal aspects (meaning the time span in which the fact is considered valid) and the license with the information is made available to the users.

1.2.1 DBpedia

DBpedia⁷ is a project with the aim of building a large-scale, multilingual, and general knowledge base by extracting data from Wikipedia (Lehmann et al. 2015, 2). It is served as Linked Data on the Web and since it covers a wide range of topics, many Linked Data publishers have decided to set RDF links pointing to DBpedia from their datasets. It contains more than 228 million entities to date. The ontology on which the extraction of the information is based is curated by the community, which guarantees an internationalization of DBpedia. It allows an automatic extraction of structured information contained in Wikipedia and it has the role of being ‘the Hub of the Linked Open Data’

⁵ <https://pro.europeana.eu/page/linked-open-data>

⁶ <https://www.yearofopen.org/what-are-open-licenses/>

⁷ <https://www.dbpedia.org/>

(Färber et al. 2015, 7), which means that it contains links to other datasets. The main language of DBpedia is English, but linked localized versions are available in 125 languages (Färber et al. 2015, 9). The information is made accessible under the terms of the Creative Commons Attribution-ShareAlike 3.0 License⁸ and the GNU Free Documentation License⁹. For what concerns the provenance of the facts and their quality, they depend on the Wikipedia content and the template mappings (Färber et al. 2015, 13). Temporal aspects are not considered, and the source of fact is not relevant because the data is entirely from Wikipedia.

1.2.2 YAGO

Yago¹⁰ (*Yet Another Great Ontology*) is a knowledge base with general knowledge about people, cities, countries, movies, and organizations. It has been in existence since 2008, and the latest version is the 4.5 which was designed to combine the data about instances from Wikidata with the taxonomy and the properties from Schema.org (Suchanek et al. 2023, 2). It contains 49 million entities and 109 million facts. The recent developments add a rich layer of informative classes to the YAGO original core, while keeping it consistent, and enlarges its use because the Wikidata entities have an abstract identifier which makes them language-independent and persistent in time (Suchanek et al. 2023, 2). This work is licensed under the Creative Commons Attribution 4.0 License¹¹, which allows sharing and adapting but with appropriate credit to the author. With the most recent release, it claims to be a ‘simplified, cleaned and ‘reasonable’ version of Wikidata’ (‘Getting Started’, 2024). Yago is one of the knowledge bases which considers both the temporal aspects and the sources of the items, hence improving its ability to represent and understand data, and to attribute reliability to the information provided.

1.2.3 Wikidata

Wikidata¹² is one of the most known knowledge bases, and the one that has been chosen for the development of the *Flying Digital Editions* project. Wikidata was born in 2012 as a ‘sister project’ of Wikipedia, with the aim to collect and structure the basic data for Wikimedia projects (Martinelli 2016, 75).

⁸ <https://creativecommons.org/licenses/by-sa/3.0/>

⁹ <https://www.gnu.org/licenses/fdl-1.3.html>

¹⁰ <https://yago-knowledge.org/>

¹¹ <https://creativecommons.org/licenses/by/4.0/>

¹² https://www.wikidata.org/wiki/Wikidata:Main_Page

Specifically, Wikidata is defined as a ‘secondary, free, collaborative and multilingual knowledge base’ because it contains facts’ sources (secondary); the data are published with the Creative Commons Zero licence so that they can be easily reused (free); the navigation is guarantee in all the 285 official languages of Wikimedia Foundation (multilingual) and the data management is supervised by the Wikidata community (collaborative) (Martinelli 2016, 76). Wikidata supports – unlike others knowledge bases previously described – multiple languages; in addition, data are made available with the licence CC0¹³, differently from the others that mostly use the Creative Common Attribution¹⁴ (Färber et al. 2015, 9). All these aspects make Wikidata as one of the main knowledge bases currently available: one of its major advantages lies in the continuous evolution that characterises the project, in addition with the community effort to maintain data always up to date (Färber et al. 2015, 21). The official page of Wikidata currently counts 108,493,775 data items¹⁵, making it one of the largest knowledge bases, which is why it was also chosen within the FDE project.

Galileo Galilei (Q307)

Italian polymath

Galileo | G. Galilei

► In more languages

Statements

instance of	human
	► 2 references

Figure 1: Example of Wikipedia data model, property-value pairs.

Each item has a unique identifier, while data is described through property-value pairs; so, for example, the item for ‘Galileo Galilei’ has the property ‘instance of’ with the value ‘human’ (see Figure 1).

If a value of a property is unknown or the value does not exist at all, Wikidata allows special types of statement making it explicit and guaranteeing complete information about the item. Moreover, every claim can include a list of references that support the claim, and some sources may be represented by

¹³ <https://creativecommons.org/public-domain/cc0/>

¹⁴ <https://creativecommons.org/licenses/by/4.0/deed.en>

¹⁵ https://www.wikidata.org/wiki/Wikidata:Main_Page

Wikidata items themselves (Vrandečić and Krötzsch 2014, 82–83). Each entity contained in Wikidata is identified by a unique URI and it allows, using the already-mentioned Linked Data principles, to obtain the item information - also in different formats.

The adoption of Linked Data practices makes Wikidata part of the Semantic Web, namely ‘a Web of data that can be processed directly or indirectly by machines’ (Berners-Lee, Bizer, and Heath 2009, 18).

1.3 Information Panels

Once data is retrieved, one of the most important tasks is to find a way to display the information. Because the information for the entities can be extremely different depending on the data that has to be displayed, the research was focused on finding a way to visualise the main facts using something that is easy to understand for any kind of users. Increasingly common is the use of small information windows that give users an overview of a topic, and an example is provided by the *Google Knowledge Panel*. The Knowledge panel is the summarised information box that appears when a user is looking for an entity on Google (see Figure 2).



Figure 2: An example of a Google Knowledge Panel for the entity "Galileo Galilei".

It is characterised by information on biography, history, photos and other data (Mancini 2023) and it aims to provide quick and relevant facts about the searched entity without visiting external websites. On Google, not all the searches will actually return a knowledge panel, because they are reserved only for entities or identifiable things ('What Is Google Knowledge Panel?' 2024).

Google Knowledge Panels are automatically generated, and the information comes from different sources across the web, sometimes combined together. They are updated automatically as information changes on the web, not only directly from the entities but also from the general user feedback ('About Knowledge Panel' 2024). The speed of access to data, coupled with the possibility to have all key

information without the need to search elsewhere, represents an improvement in the user experience, increasing the chances of engagement (Sanjay 2023).

Flying Digital Editions exploits the idea of using information windows to summarise and extract the most important data, following some conventional practices for describing entities - e.g. the date of birth or death if the entity is a human - but also adding interesting useful facts. Moreover, the use of Wikidata will provide a similar behaviour to Google Knowledge Panels, because it will allow for up-to-date data.

1.4 Existing Tools

In this section, several entity linking tools will be briefly examined and discussed. To test the performance of some of these, it will be used a paragraph from the chapter *The system of the world* from ‘The Mathematical Principles of Natural Philosophy’¹⁶ written by Isaac Newton (1846). The choice of this text is based on the idea of leveraging these tools for exploring texts of any type, including scientific ones. Furthermore, a text from the 1800s was chosen to test the ability of these NER tools with non-contemporary English and to find out their effectiveness and validity under these conditions. For some of these tools, the F-score (or F-measure) is highlighted to provide a complete analysis and comparison as much as possible.

The F-score is a statistical analysis of predicted performance. It is calculated from the combination of *precision* and *recall*. The precision measures the percentage of the items detected by the system as positive that are in fact positive, and it is defined as the ratio of true positives to the sum of true and false positive. The recall, instead, measures the percentage of the items correctly identified by the system and it is defined as the ratio of true positive to true positives and false negatives (Martin and Jurafsky 2024, 71). When the importance of recall and precision is balanced one talks about the F1, which is the harmonic mean of precision and recall. This metric is used when aiming for a balance between model’s ability to make accurate predictions and its ability to identify all the positive instances. If the F1 score is high, it means better performance of the model.

¹⁶ [https://en.wikisource.org/wiki/Page%3ANewton's_Principia_\(1846\).djvu/517](https://en.wikisource.org/wiki/Page%3ANewton's_Principia_(1846).djvu/517)

1.4.1 Wikidata API SandBox

API Sandbox is an online tool provided by Wikidata to experiment with the MediaWiki Web Service API. For this tool, it was not possible to use the sample text because it does not annotate a whole

The screenshot shows the Wikidata API Sandbox interface. On the left is a sidebar with links: 'main', 'action=wbgetentities' (highlighted), 'format=json', and 'Results'. The main area is titled 'action=wbgetentities' and 'Gets the data for multiple Wikibase entities.' Below this is an 'Examples' button. The form contains several fields: 'ids' (empty), 'sites' (set to 'enwiki'), 'titles' (set to 'John Bauer'), 'redirects' (set to 'yes'), 'props' (a list of property types including 'info', 'sitelinks', 'aliases', 'labels', 'descriptions', 'claims', 'datatype'), and 'languages' (set to 'en'). Each field has a brief description and a maximum value limit. Checkmarks are visible next to the 'sites', 'titles', and 'languages' fields.

Figure 3: Example of API request for the title ‘John Bauer’ with action *wbgetentities*.

document, but it works with short strings, such as names. In fact, by choosing the action, the format and other parameters, the user can make a request and copy the request URL for reusing it again (‘API Sandbox’ 2023). For the FDE’s purposes, two types of action have been considered and, later, excluded: *wbgetentities*¹⁷ and *query*¹⁸.

The *wbgetentities* action (see Figure 3 and Figure 4) can be used for retrieving data for multiple Wikibase entities by customising the request with information about the language, the site on which the item resides, the title or the id of the entity itself. The issue with this type of approach lies in the impossibility of having a disambiguation starting from the entities’ name, without the identifier of the Wikidata item. If we use the title of the page, but this title is used in more than one page, the API

¹⁷ <https://www.wikidata.org/w/api.php?action=help&modules=wbgetentities>

¹⁸ <https://www.wikidata.org/w/api.php?action=help&modules=query>

directly returns the *Wikipedia Disambiguation Page*, and not all the list of the possible entities. For the FDE project purposes, this behaviour is not convenient because it means to check one by one each entity contained in the disambiguation page.

The screenshot displays the Wikidata API interface. On the left, a sidebar contains links: 'main', 'action=wbgetentities', 'format=json', and 'Results' (highlighted in blue). The main area shows 'Show request data as:' set to 'URL query string' and 'Request URL:' as 'https://www.wikidata.org/w/api.php?action=wbgetentities&for'. Below this, a JSON response is shown, detailing the entity 'John Bauer' with fields like 'pageid', 'title', 'lastrevid', 'modified', 'type', 'id', and 'labels'.

```
{
  "entities": {
    "Q3809082": {
      "pageid": 3632697,
      "ns": 0,
      "title": "Q3809082",
      "lastrevid": 2026025950,
      "modified": "2023-12-07T23:25:52Z",
      "type": "item",
      "id": "Q3809082",
      "labels": {
        "en": {
          "language": "en",
          "value": "John Bauer"
        }
      }
    }
  }
}
```

Figure 4: Example of response after API request for the title 'John Bauer' with action *wbgetentities*.

On the contrary, the *query* action fetches data from and about MediaWiki, which is a collaboration and documentation platform that helps to collect and organise knowledge and make it available to people ('MediaWiki' 2023). If one configures the 'list' parameter to 'search' and insert in 'srsearch' the entity name one is looking for, the action will perform a text search that can be refined by adding some specific settings, e.g. which metadata or properties to return (see Figure 5 and Figure 6).

main	list=search Perform a full text search.
action=query	<div> <div>?</div> <div>Help links</div> <div>{}</div> <div>Examples</div> </div>
list=search	<div> <div>srsearch</div> <div>John Bauer</div> <div>*</div> </div>
format=json	<div> <div>Search for page titles or content matching this value. You can use the search string to invoke special search features, depending on what the wiki's search backend implements.</div> </div>
Results	<div> <div>srnamespace</div> <div>(Main)</div> <div></div> </div>
	<div> <div>Search only within these namespaces.</div> </div>
	<div> <div>srlimit</div> <div>10</div> <div></div> </div>
	<div> <div>How many total pages to return. The value must be between 1 and 500. Enter max to use the maximum limit.</div> </div>
	<div> <div> <div>sroffset</div> <div>-</div> <div>0</div> <div>+</div> <div></div> </div> </div>
	<div> <div>When more results are available, use this to continue. More detailed information on how to continue queries can be found on mediawiki.org. The value must be no less than 0.</div> </div>
	<div> <div>srqprofile</div> <div>engine_autoselect</div> <div></div> </div>
	<div> <div>Query independent profile to use (affects ranking algorithm).</div> <div>[Expand]</div> </div>

Figure 5: Example of API request for the title 'John Bauer' with action query.

This action returns a list of items characterised with the Wikidata ID, a snippet that corresponds to a description of the items itself, but it does not return any kind of information about the class or the type of the retrieved entities.

main	<div> <div>Show request data as:</div> <div>URL query string</div> <div></div> </div>
action=query	<div> <div>Request URL:</div> <div>https://www.wikidata.org/w/api.php?action=query&format=json&prop=&list=search&formatv</div> <div>Copy</div> </div>
list=search	<div> <div>format=json</div> </div>
Results	<div> <pre>{ "batchcomplete": true, "continue": { "sroffset": 10, "continue": "- " }, "query": { "searchinfo": { "totalhits": 3235 }, "search": [{ "ns": 0, "title": "Q214043", "pageid": 209448, "snippet": "Swedish painter and illustrator (1882-1918)", "categorysnippet": "" }, { "ns": 0, "title": "Q6221125", "pageid": 6021305, "snippet": "American football guard and tackle who played for the New York Giants (1932-2010)", "categorysnippet": "" }] } }</pre> </div>

Figure 6: Example of response after API request for the title 'John Bauer' with action query

The result of this action has been also considered during the realisation of the *Flying Digital Editions* project. The main problem is that the entities must be already detected from another process for

allowing this API to work fine. In addition to that, the items retrieved are lacking any kind of categorisation. This fact makes the retrieval of the data more difficult because each entity must be firstly associated with a class (namely, the item type) to be able to perform the query suitable for collecting the specific information wanted for that type of item. Better information about this kind of approach can be found in [2.2.1](#).

1.4.2 Stanza

*Stanza*¹⁹ is an open-source Python toolkit for natural language processing (Qi et al. 2020, 1). It is a neural pipeline that performs tasks of text analysis, such as tokenization and named entity recognition. Compared to other NLP toolkits, it takes raw text as input, and produces annotations including – other the ones already mentioned – also dependency parsing, multi-word token expansion, and lemmatization. Since its architecture is designed to be language-agnostic and data-driven, the models have been trained to support 66 human languages. Indeed, it has been tested on 112 datasets with competitive results which demonstrates that its neural pipeline adapts well on different genres texts. In fact, on all the dataset on which Stanza has been tested, it achieved higher results of F1 compared to the famous spaCy²⁰, which is an open-source library for Natural Language Processing. For example, on the CoNLL03 corpus with the English language, Stanza score was of 92.1 against the 81.0 of spaCy.

The Stanza interface expands the Standfort CoreNLP software capabilities to incorporate coreference resolution and relation extraction (Qi et al. 2020). A demo online has been created to help visualise documents and their annotations, running the pipeline interactively (see Figure 7).

¹⁹ <http://stanza.run/>

²⁰ <https://spacy.io/>

— Text to annotate —

It was the ancient opinion of not a few, in the earliest ages of philosophy, that the fixed stars stood immoveable in the highest parts of the world; that under the fixed stars the planets were carried about the sun; that the earth, as one of the planets, described an annual course about the sun, while by a diurnal motion it was in the mean time revolved about its own axis; and that the sun, as the common fire which served to warm the whole, was fixed in the centre of the universe. This was the philosophy taught of old by Philolaus, Aristarchus of Samos, Plato in his riper years, and the whole sect of the Pythagoreans; and this was the

— Annotations —

named entities x

— Language —

English

Submit

Named Entity Recognition:

1 It was the ancient opinion of not a few , in the earliest ages of philosophy , that the fixed stars stood immoveable in the highest parts of the world ; that under the fixed stars the planets were carried about the sun ; that the earth , as one of the planets , described an annual course about the sun , while by a diurnal motion it was in the mean time revolved about its own axis ; and that the sun , as the common fire which served to warm the whole , was fixed in the centre of the universe .

2 This was the philosophy taught of old by Philolaus , Aristarchus of Samos , Plato in his riper years , and the whole sect of the Pythagoreans ; and this was the judgment of Anaximander , more ancient than any of them ; and of that wise king of the Romans , Numa Pompilius , who , as a symbol of the figure of the world with the sun in the centre , erected a temple in honour of Vesta , of a round form , and ordained perpetual fire to be kept in the middle of it .

3 The Egyptians were early observers of the heavens ; and from them , probably , this philosophy was spread abroad among other nations ; for from them it was , and the nations about them , that the Greeks , a people of themselves more addicted to the study of philology than of nature , derived their first , as well as soundest , notions of philosophy ; and in the vestal ceremonies we may yet trace the ancient spirit of the Egyptians ; for it was their way to deliver their mysteries , that is , their philosophy of things above the vulgar way of thinking , under the veil of religious rites and hieroglyphic symbols .

4 It is not to be denied but that Anaxagoras , Democritus , and others , did now and then start up , who would have it that the earth possessed the centre of the world , and that the stars of all sorts were revolved towards the west about the earth quiescent in the centre , some at a swifter , others at a slower rate .

5 However , it was agreed on both sides that the motions of the celestial bodies were performed in spaces altogether free and void of resistance .

6 The whim of solid orbs was of a later date , introduced by Eudoxus , Calippus , and Aristotle ; when the ancient philosophy began to decline , and to give place to the new prevailing fictions of the Greeks .

Figure 7: Example of text annotation with Stanza

The tool, which has a strong resemblance with the CoreNLP Client Interface²¹, provides a text-area in which the user can insert the text to annotate, to choose the annotation's types and to select the language. After the submit, the text is displayed divided into smaller paragraphs with the annotated recognized entities. The explanation of the annotation is not always explicit and clear for the user. For example, to understand what 'norp' means the user must go to the documentation's page of Stanza²² and then search for it. 'Norp' stands for 'Nationalities/religious/political group' and easily it can be noticed that the word 'Calippus' was misinterpreted.

Stanza has been considered during the development of the Flying Digital Editions project for the extraction of the entities from a text. In combination with the previous discussed Wikidata API, it would recreate a named entity recognition and entity-linking operation. The main problem was the delay this approach would be subject to, because for each entity detected by Stanza, a request would

²¹ <https://corenlp.run/>

²² https://stanfordnlp.github.io/stanza/ner_models.html

be sent to the endpoint of the Wikidata API. The risk of a slow and overly complicated process resulted in the decision to exclude this combo and adopt a single tool to perform both functions.

1.4.3 Wikifier

*Wikifier*²³ is a tool that operates a specific semantic annotation, known as *wikification*, involving Wikipedia as a source (Brank and Grobelnik 2017). This semantic annotation includes several subtasks, such as the identification of phrases or words in the input document referring to a Wikipedia concept and of which concept a phrase refers to. Wikifier implements a ‘global disambiguation approach’, constructing a *mention-concept graph* for the input document that can be thought as a bipartite graph which is used as the basis of calculating a vector of pagerank scores (Brank and Grobelnik 2017, 2). This approach to Wikification requires no external data except the Wikipedia itself. In fact, the Wikifier currently supports a total of 134 languages, which are the languages of the pages for which Wikipedia is available. Annotations are returned in a JSON format and can optionally include detailed information about which mentions support each annotation and alternative candidate.

For what concern the evaluation of this tool, the authors report that their Wikifier does not by default distinguish between named entities and other Wikipedia concepts, so they had to explicitly exclude non-entity concepts. By comparing the available wikifier systems and the gold standard, it results that its performance is slightly worse than AIDA²⁴ system (respectively 0.593 and 0.723) but better than other wikifiers as DBpedia Spotlight, which has a score of 0.279. The experiment also shows the agreement between each pair of wikifiers, resulting that the Wikifier-AIDA has an agreement of 0.625 while the level is 0.363 with Spotlight (Brank, Leban, and Grobelnik 2017, 4). The authors make clear that one of the main weaknesses of this tool concern the treatment of minority languages.

An online implementation as a web service is running for testing the system and its functionalities. It is clearly focused to semantically annotate documents and provides minimal information about the entities themselves. The page is divided into four columns: ‘Text’, ‘Annotations’, ‘Support’ and ‘Link Targets’ (see Figure 8). The ‘Annotations’ columns in three sub-columns:

- The Page Rank, by which the entities are ordered in the list;

²³ <https://wikifier.org/>

²⁴ <https://github.com/codepie/aida>

- The annotation, which presents two letters: ‘W’ and ‘D’. The first shows a tooltip with all the Wikidata items related to the entity, while the second shows the DBpedia properties;
- The annotation in english language, which is linked to the Wikipedia page.

By clicking on the ‘>>’ symbol next to the annotation, the ‘Support’ column is valued with the phrases in the text that supports the annotation. Finally, ‘Link Targets’ display where links with a particular anchor text point to. For visualize the results, the user has to select a phrase on the text and click on it.



Figure 8: Example of Wikifier’s analysis result on a text.

Immediately below, the tool provides the part-of-speech tagging process results by using different colours for the different parts, and, by clicking on each word, it provides the possibility to see a list of corresponding synsets in Wordnet (see Figure 9).

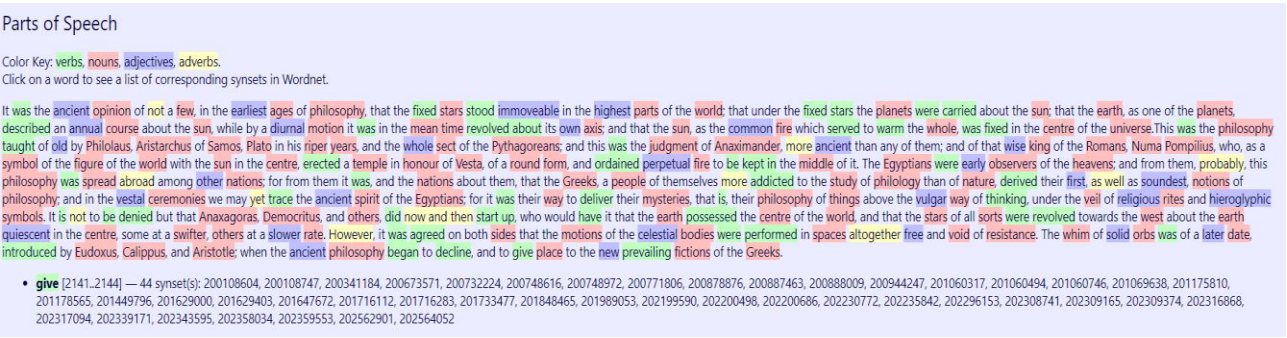


Figure 9: Parts of Speech annotation on a text with Wikifier tool.

Wikifier is a tool for experiments and for semantic annotation and not an instrument for exploring and reading the text better. The knowledge implicit in the document is not actually considered and the tool do not provide any additional information or feature but Natural Language Processing procedures. It seems to be suitable for use by expert users only, who are familiar with the subject matter and terminology.

1.4.4 OpenTapioca

*OpenTapioca*²⁵ is a Named Entity Linking system that can be trained from Wikidata only and can be efficiently kept up to date in real time as Wikidata evolves, according to the advantage that this knowledge base provides. In addition, Wikidata also contains items which do not have any corresponding Wikipedia articles, on which others knowledge base are based instead, meaning it contains a lot of more entities that can be identified and linked.

²⁵ <https://opentapioca.org/#>

[[OpenTapioca^β]]

OpenTapioca annotates text with locations, organizations and people from [Wikidata](#).
It is synchronous with Wikidata.
This is a prototype. See the [GitHub project](#) for more info.

en ▼

described an annual course about the sun, while by a diurnal motion it was in the mean time revolved about its own axis; and that the sun, as the common fire which served to warm the whole, was fixed in the centre of the universe. This was the philosophy taught of old by Philolaus, Aristarchus of Samos, Plato in his riper years, and the whole sect of the Pythagoreans; and this was the judgment of Anaximander, more ancient than any of them; and of that wise king of the Romans, Numa Pompilius, who, as a symbol of the figure of the world with the sun in the centre, erected a temple in honour of Vesta, of a round form, and ordained perpetual fire to be kept in the middle of it.

Annotate

It was the **ancient** **opinion** of not a **few**, in the earliest **ages** of **philoso** stood immoveable in the highest par **r** the fixed **stars** the **planets** **n**; that the **earth**, as **one** of th **annual** **course** **about** the **sun**, whil **s** in the **mean** **time** revolved **a** that the **sun**, as the **common** **f** the whole, was fixed in the **centre** of **philosophy** taught of **old** by **Philola** **Plato** in **his** riper years, and the who **s**; and this was the judgment of **Anaximander**, **more** **ancient** **than** **any** of **them**; and of that **wise** **king** of the **Romans**, **Numa** **Pompilius**, **who**, as a **symbol** of the **figure** of the **world** with the **sun** in the **centre**, erected a **temple** in **honour** of **Vesta**, of a **round** **form**, and ordained **perpetual** **fire** to be **kept** in the **middle** of it.

ancient	Ancient (Q489320) Norwegian black metal band Rank: 0.17, phrase: 10.71 Statements: 31, sitelinks: 19 Score: -2.0115251024331666
opinion	Ancient (Q2845962) Japanese video game company Rank: -0.58, phrase: 10.71 Statements: 12, sitelinks: 9 Score: -2.0918018893467094
few	Ancient (Q109503186) Wellcome Collection story Rank: -1.57, phrase: 10.71 Statements: 9, sitelinks: 0 Score: -2.200822980042066

Figure 10: Example of text annotation with OpenTapioca.

This system can recognize only items whose type is a subclass of, - i.e. it has property (P279) with the value of - human (Q5), organisation (Q43229) or geographical object (Q618123) (Delpeuch 2020, 8).

For testing the performance of this tool, the authors had to convert an entity linking dataset from DBpedia to Wikidata, since most entity-linking datasets are annotated against DBpedia itself or YAGO. In addition, they also annotated a new dataset. The results on the AIDA-CoNLL shows an F1 lower for the OpenTapioca than the AIDA. Better performances are instead noticed for the ISTEEX-1000 corpus, showing 0.870 and 0.858 than the AIDA's 0.531 and 0.494 (Delpuch 2020, 8). Currently, the demo of the OpenTapioca project is working on a smaller server²⁶ due to maintenance costs, and the project has made available two NIF (Natural Language Interchange Format)²⁷ endpoints; the first exposes the matches considered good enough and the second exposes all the other matches.

The interface of this tool allows users to annotate texts in different languages, just by selecting the wanted one through a dropdown. The list includes a lot of languages, such like Bangla (Bangladesh), Hindi (India), and Korean. By testing it with the sample text, the annotation progress recognizes and links a lot of entities and relationships (see Figure 10), but the disambiguation on some detected words, such as 'few' and 'ages', appears not accurate. For example, for the word 'ancient', the tool links three Wikidata items that are not relevant to the content of the text.

The demo is actually easy and intuitive to use, but the user experience is not one of the best. The information box displays little basic data, such as the title of the Wikidata page, its identifier, and a brief description, while the other information concerns the 'score' of the word for being preferred according to the context. The presence of a lot of rectangular brackets makes the reading of the annotated text a bit confusing and unpleasant. Point in favour of this tool is the speed it has in annotating the text: in fact, it takes only a few seconds to complete the process, even if the length of input is quite long.

1.4.5 Falcon 2.0

*Falcon 2.0*²⁸ is a tool that derives from a previous work, *Falcon*, which is a rule-based approach effective for entity and relation linking on a short text over DBpedia (Sakor et al. 2020, 3142). Falcon 2.0 joins entity and relation linking over Wikidata and relies on the principles of English morphology, by linking entity and relation surface forms to its Wikidata mentions. Its baseline is the

²⁶ <https://opentapioca.wordlift.io/#>

²⁷ <https://github.com/dice-group/gerbil/wiki/NIF>

²⁸ <https://labs.tib.eu/falcon/falcon2/>

previously discussed OpenTapioca, which, however, does not provide Wikidata identifiers of relations in a sentence. Falcon 2.0 is accessible via a CURL request or using the web interface (see Figure 11).

The tool receives short input texts and outputs a set of entities and relations extracted from the text, which are associated with a unique identifier in Wikidata. The background knowledge combines Wikidata labels with their aliases and the alignments between nouns and entities are stored in a text search engine, while the English morphology rules are stored into a catalogue (Sakor et al. 2020, 3143). The whole process is essentially divided into two steps: the recognition step and the linking one. The first phase, which also includes the extraction of the entities, consists of three modules: the Part-of-speech tagging, the Tokenization & Compounding, and the N-Gram Tiling. Following, the linking phase involves: the candidate list generation, the matching & ranking process, the relevant rule selection and the n-gram splitting.

The demo available was designed to be a simple interface showing the functionality of the Falcon 2.0 API. It consists of a text-area in which the user can insert the text, a search button can be clicked to show the results, and few templates' phrases are provided below the box to test the tool. The users can choose between two settings: 'Relations and Entities in a sentence' and 'Keyword Search'.

For what concerns the performance, it appears that if the text is too long the server is not able to process it and returns a 500 error. Hence, the sample text has been reduced to about seven sentences to be allowed to be processed from the system, that will return both the original Falcon and the Falcon 2.0 results. The JSON response is directly showed inside a box, displaying both the entities and the 'relations' recognized. The Wikidata items returned if checked manually seems to be not accurate: for the surface form detected 'sun; that the earth', the linked item is the one referring to the Earth, totally excluding the 'sun' word (see Figure 11).

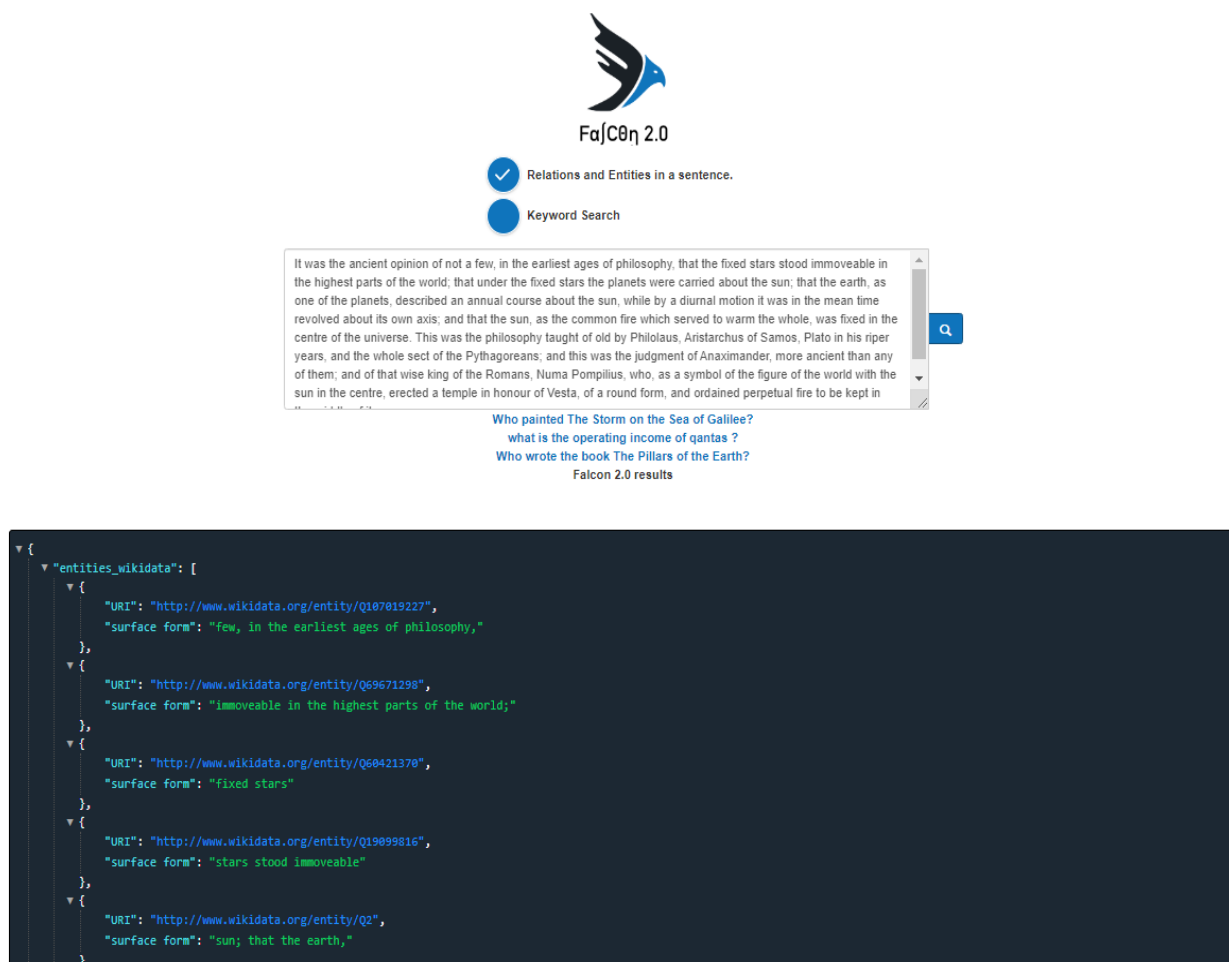


Figure 11: Example of entity linking with Falcon 2.0.

Same happens for ‘immutable in the highest parts of the world’ which is linked with a scientific article²⁹, missing completely the text context.

Falcon 2.0 is intuitive and easy to use, but the data retrieved by the tool remains implicit and the user must click on the URI provided to visualize additional information. Furthermore, the response is difficult to read due to the format in which it is displayed. One must make a continuous comparison with the text above to understand where one is, which is unpleasant and inefficient.

The evaluation of this tool has been compared to the OpenTapioca performance from the authors of the system themselves. On the LC-QuAD 2.0 & SimpleQuestion datasets it looks like to be significantly better, outperforming OpenTapioca by a wide margin; the test in which this difference is most obvious shows Falcon 2.0 with a score of 0.60 against the OpenTapioca’s F-score of 0.01.

²⁹ <https://www.wikidata.org/wiki/Q69671298>

Falcon 2.0 was also tested on the WEBQSP dataset, outperforming all the others approach with a score of 0.82 – while OpenTapioca only achieved 0.02 (Sakor et al. 2020, 3147).

1.4.6 TagMe

*TagMe*³⁰ is a tool able to ‘augment a plain text with pertinent hyperlinks to Wikipedia pages’ (Ferragina and Scaiella 2010, 1). The strength of TagMe lies in the possibility to annotate short text, as brief descriptions, because the system tries to identify ‘spots’ - sequence of terms of the input - and to annotate them with disambiguated entities (Ferragina and Scaiella 2010, 1). The authors have adapted the tool to work also on long texts and the user can set the ‘number of links’ that the tool can return related to the text.

The tool interface is simple and intuitive. A double box allows to paste a text into the first text area, by also selecting the language in which the annotation must be performed. It supports three languages: Italian, English, and German. The user can decide ‘how many’ links the tool must annotate on the text, by setting a cursor on a bar which has two ends: ‘few links’ and ‘many links’.

The annotation result is displayed in the second area, under the ‘Tagged Text’ tab, where the whole text is returned, and the recognised entity are underlined.

The information about the entities can be previewed through a little tooltip window that appears when the mouse goes over the text; it shows the name and the start of the Wikipedia description found. To have the whole information retrieved, the user must click on the entity, and he will be redirect to the corresponding Wikipedia page. In a second tab, ‘Topics’, the tool displays all the *topics* - i.e. the recognized entities – by grouping them.

For testing this tool with the sample text, several attempts were performed to understand the change in setting a different number of links in the configuration. If configured to ‘Few links’, TagMe highlights only the main entities. On the contrary, by increasing the number of links, it looks like the disambiguation process suffers a little and some incorrectness appears (see Figure 12). If the other top of the bar is reached, the tagged text is difficult to read because is completely underlined in blue, and the misrecognition of the entities are more frequent.

By reducing the text to a few sentences, it looks like the performance is consistent with what happens with the full text.

³⁰ <https://sobigdata.d4science.org/web/tagme/demo>

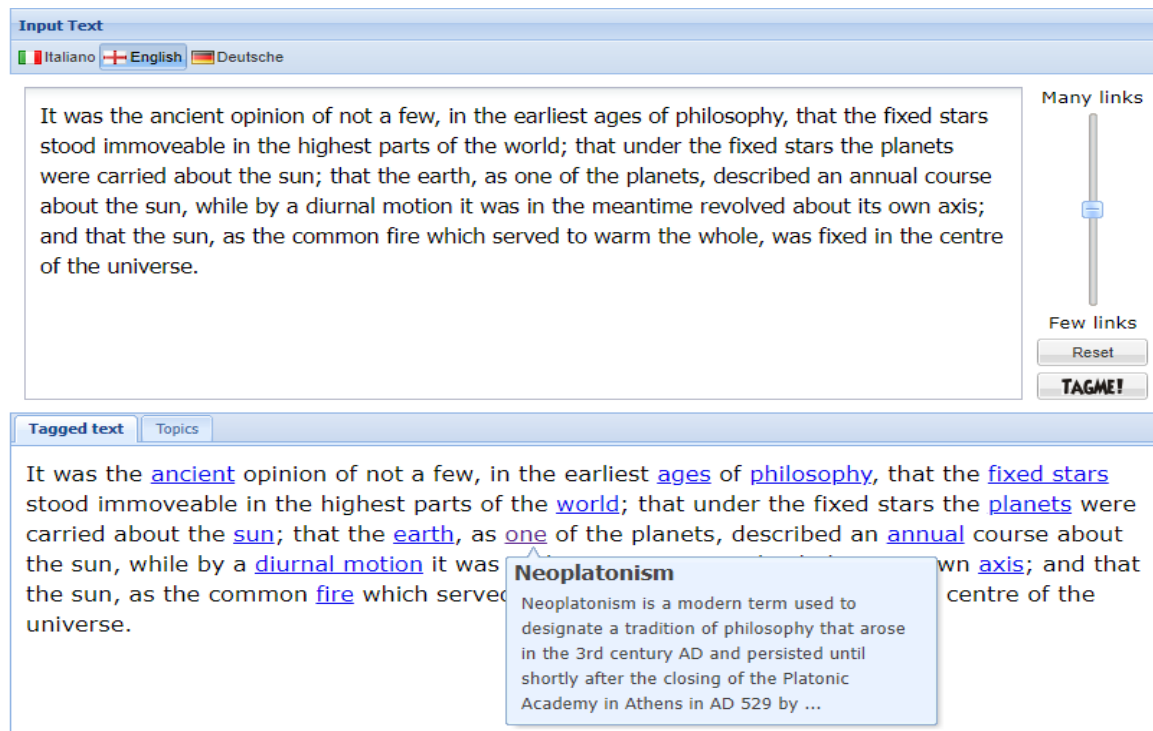


Figure 12: Example of text annotation with TagMe.

Unlike the other tools discussed, this one seems to be more user-friendly for non-experts. It remains intuitive and easy to understand, although the ‘link bar’ is not explicitly explained. The fact that the text is re-presented in full and annotated with underlining makes reading pleasant. A negative point is that the user is forced to leave the current page for viewing the entity information on the Wikipedia page.

For what concern statistic information about the performance, the tools was tested on three datasets: the first is a manually annotated dataset, the second and third ones were derived from Wikipedia. They used one for evaluating the disambiguation phase, calling it WIKI-DISAMB30, while the other one was employed for estimating the overall system performance. It is called WIKI-ANNOT30 and consists of 150K fragments. The comparison was conducted between TagMe and the system of Milne&Witten³¹, with the TagMe approach outperforming the latter’s recall to precision ratio. It achieves an F-score of the 78%, by surpassing its competitor by almost 5% (Ferragina and Scaiella 2010, 3–4).

³¹ https://www.researchgate.net/publication/33052014_Learning_to_Link_with_Wikipedia

1.4.7 DBpedia Spotlight

*DBpedia Spotlight*³² is an open-source project for a system that automatically annotates DBpedia entities in natural languages texts (Daiber et al. 2013, 1). As other demos previously analysed, DBpedia Spotlight provides interfaces for recognizing phrases to be annotated and entity disambiguation. In its last version, the performance and accuracy of the entity recognition and disambiguation component have been enhanced, in addition with the feature to work with other languages.

The implementation consists of three main phases: the first is the phrase spotting, which is the task of finding phrases inside a text; then, there is the step of disambiguation, performed with the generative probabilistic model form; and finally, the indexing process (Daiber et al. 2013, 1–3).

For evaluating the annotation process, the developers of this tool used a test corpus consisted of 35 paragraphs from New York Times documents from different categories. The ratio of annotated to not-annotated tokens was 33% and this result was compared with other publicly available annotation services, such as the already mentioned Milne&Witten’s wikifier, which does not explicitly return DBpedia URIs, but it can be inferred from the Wikipedia page link that it is retrieved.

The comparison among these different systems shows at the top the F1 score of the DBpedia Spotlight tool in its best configuration with a 56.0%. Following, again DBpedia Spotlight, this time without any configuration, achieving the 45.2%.

The demo made available online allows the user to select among different configurations. In particular, the list includes 16 languages, for the majority European. A bar can be used to set the confidence of the results, and the tool also provide a checkbox that can be checked for having the ‘n-best candidates’. Interesting feature is the possibility to select the types of entities that the user wants to retrieve. Once clicked on the button, a modal appears with the different class contained in DBpedia, Freebase, Schema.org and, also, a tab that allow the user to filter the results by creating a Custom SPARQL query.

The result of the process is an annotated text (see Figure 13) with a considerable number of recognized entities, which are navigable thanks to the direct link to Dbpedia.org. For the sample text, it can be noted that a lot of entities that represent persons have not been recognized by the system.

³² <https://demo.dbpedia-spotlight.org/>



Confidence: 0.5 Language:

☐ n-best candidates

It was the ancient opinion of not a few, in the earliest ages of [philosophy](#), that the [fixed stars](#) stood immoveable in the highest parts of the world; that under the [fixed stars](#) the planets were carried about the sun; that the earth, as one of the planets, described an annual course about the sun, while by a [diurnal motion](#) it was in the mean time revolved about its own axis; and that the sun, as the common fire which served to warm the whole, was fixed in the centre of the universe. This was the [philosophy](#) taught of old by [Philolaus](#), Aristarchus of Samos, Plato in his riper years, and the whole sect of the [Pythagoreans](#); and this was the judgment of Anaximander, more ancient than any of them; and of that wise king of the Romans, Numa Pompilius, who, as a symbol of the figure of the world with the sun in the centre, erected a [temple](#) in honour of [Vesta](#), of a round form, and [ordained](#) perpetual fire to be kept in the middle of it. The [Egyptians](#) were early observers of the heavens; and from them, probably, this [philosophy](#) was spread abroad among other nations; for from them it was, and the nations about them, that the [Greeks](#), a people of themselves more addicted to the study of [philology](#) than of nature, derived their first, as well as soundest, notions of [philosophy](#); and in the vestal ceremonies we may yet trace the ancient spirit of the [Egyptians](#); for it was their way to deliver their mysteries, that is, their [philosophy](#) of things above the vulgar way of thinking, under the [veil](#) of religious rites and [hieroglyphic](#) symbols. It is not to be denied but that Anaxagoras, Democritus, and others, did now and then start up, who would have it that the earth possessed the centre of the world, and that the stars of all sorts were revolved towards the west about the earth quiescent in the centre, some at a swifter, others at a slower rate. However, it was agreed on both sides that the motions of the celestial bodies were performed in spaces altogether free and void of resistance. The whim of solid orbs was of a later date, introduced by Eudoxus, [Calippus](#), and Aristotle; when the [ancient philosophy](#) began to decline, and to give place to the new prevailing fictions of the [Greeks](#).

Only showing the types: DBpedia:owl:Thing, DBpedia:unknown, Freebase:http://www.w3.org/2002/07/owl#Thing, Freebase:unknown, Schema:owl#Thing, Schema:unknown

Figure 13: Example of text annotation using DBpedia Spotlight.

The interface consists of a textbox in which the user can insert the text, that will be returned after the processing in the same area, but with the entities underlined. A button on the right enables the user to modify again the text and a box under the text-area shows the types that the user selected before sending the annotation request. This means that the analysis is limited to entities and concepts categorised under the specific classes shown there within the DBpedia knowledge graph.

The configuration settings are quite intuitive and easy to use, but it looks like a test page more than a real usable tool. No information about the entities is displayed clearly, and the user must click on the

underlined entity to be able to navigate it on the corresponding DBpedia page, interrupting every time the reading of the text.

1.4.8 entity-fishing

The services provided by *entity-fishing*³³ aims to automate the tasks concerning entities' recognition and disambiguation, avoiding as much as domains' limitations or specific usages. It disambiguates entities against Wikidata which, as already pointed out, is a multilingual knowledge base that ensures the possibility to take advantage of cross-lingual information (Lopez 2022, 'Motivation'). Through a dropdown the tool allows users to select among several services that can perform different tasks: language identification, sentence segmentation, disambiguation (on a text or on a PDF), concept look-up and term look-up.

entity-fishing is the tool that comes closest to the idea behind the Flying Digital Editions project because it provides the possibility for users to easily upload a text and immediately visualise it annotated, adding an in-page general description of the entity itself. The tool supports 15 languages and even if the documentation appoints an experimental editing mode for interactive disambiguation, this feature is not currently active online and cannot be tested for this research purposes. By using the sample text, it has been noticed that the system can identify a considerable number of entities, but sometimes the linking process performs some incorrect associations (see Figure 14).

³³ <https://cloud.science-miner.com/nerd/>

entity-fishing

About Services

Service to call

disambiguate - text

{

"text": "It was the ancient opinion of not a few, in the earliest ages of philosophy, that the fixed stars stood immovable in the highest parts of the world; that under the fixed stars the planets were carried about the sun; that the earth, as one of the planets, described an annual course about the sun, while by a diurnal motion it was in the mean time revolved about its own axis; and that the sun, as the common fire which served to

}

Submit

Clear

WW1

PubMed_1

PubMed_2

HAL_1

Italiano

News_1

News_2

French

German

Spanish

COVID-19

query_1

query_2

query_3

query_4

Annotations

Response

It was the ancient opinion of not a few, in the earliest ages of philosophy, that the **FIXED STARS** stood immovable in the highest parts of the world; that under the **FIXED STARS** the planets were carried about the sun; that the earth, as one of the planets, described an annual course about the sun, while by a **DIURNAL MOTION** it was in the mean time revolved about its own axis; and that the sun, as the common fire which served to warm the whole, was fixed in the **CENTRE OF THE UNIVERSE**. This was the philosophy taught of old by **PHILOLAUS**, **ARISTARCHUS OF SAMOS**, **PLATO** in his **RIPER** years, and the whole **SECT** of the **PYTHAGOREANS**; and this was the judgment of **ANAXIMANDER**, more ancient than any of them; and of that wise **KING OF THE ROMANS**, **NUMA POMPILIUS**, who, as a symbol of the figure of the world with the sun in the centre, **ERECTED** a temple in honour of **VESTA**, of a round form, and **ORDAINED** **PERPETUAL** fire to be kept in the middle of it. The **EGYPTIANS** were early **OBSERVERS** of the **HEAVENS**; and from them, probably, this philosophy was spread abroad among other nations; for from them it was, and the nations about them, that the **GREEKS**, a people of themselves more **ADDICTED** to the study of **PHILOLOGY** than of nature, derived their first, as well as soundest, **NOTIONS** of philosophy; and in the vestal **CEREMONIES** we may yet trace the ancient spirit of the **EGYPTIANS**; for it was their way to deliver their **MYSTERIES**; that is, their philosophy of things above the **VULGAR** way of thinking, under the **VEIL** of **RELIGIOUS RITES** and **HEROGLYPHS**. It is not to be denied but that **ANAXAGORAS**, **DEMOCRITUS**, and others, did now and then start up, who would have it that the earth **POSSESSED** the **CENTRE OF THE WORLD**, and that the stars of all sorts were revolved towards the west about the earth **QUESCIENT** in the centre, some at a swifter, others at a **SLOWER** rate. However, it was agreed on both sides that the **MOTIONS** of the **CELESTIAL BODIES** were performed in spaces altogether free and **VACUUM** of resistance. The **MYTH** of solid **ORBS** was of a later date, introduced by **EUROMUS**, **CALIPPUS**, and **ARISTOTLE**; when the **ANCIENT PHILOSOPHY** **BEGAN TO DECLINE**, and to give place to the new **PREVAILING** **FIXCTIONS** of the **GREEKS**.

PHILOLAUS

Normalized: Philolaus (crater)

Domains: Astronomy

conf: 0.7354

Philolaus is a **lunar impact crater** that is located in the northern part of the **Moon's** near side. It lies within one crater diameter to the east-southeast of the flooded crater **Anaximenes**, and to the west of the smaller **Anaxagoras**. It overlies the older and heavily worn Philolaus C to the south. This crater retains a well-defined form that has not changed significantly since it was originally created. The outer rim edge is roughly circular, but with a somewhat irregular edge that displays signs of slumping. The most notable slump is a triangular area along the eastern rim. The inner wall of the crater has a complex system of **terrace** with a sharp-edged rim in locations where slumping has occurred. On the exterior of the rim is an outer **rampart** that extends outwards for nearly half a crater diameter in all directions. The crater has a **ray system**, and is consequently mapped as part of the **Copernican System**.

Wikidata statements

References:

Figure 14: Example of text annotation using entity-fishing.

The result of the annotation can be visualized both as a text in which the entities are highlighted with colours that represent their domain, or in the JSON format.

The information panel that appears when mousing over the entity includes the normalised name of the entity, the belonging domain and a brief description that derives from its Wikipedia page. Before the references icons that connect to Wikidata and Wikipedia item’s page, there is also a hidden accordion in which are contained the ‘Wikidata statements’. In a table form, all the properties of the item and their values included in the Wikidata page are listed.

Surely, the inclusion of an information panel represents a novelty compared to the other tools, but entity-fishing remains something very technical to use and not so much intuitive, especially in the way the user must enter the text to be analysed. The aim of being a ‘generic’ tool is the reason it neither inclines to be an educational nor to be totally a testing and analytical tool.

33

For this tool does not exist a real paper presentation. Most of the information can be retrieved on the GitHub repository³⁴ of the project, which also provides some evaluation scores, and the evaluation npage on the online documentation³⁵. They correspond to the ‘overall unnormalized accuracy’ scenario in Blink – which is an entity linking python library - and are limited to Named Entities. entity-fishing performs a F-score of 0.765 for the AIDA-CoNLL dataset, while for the AQUAINT one, it surpasses Blink with a score of 0.891 against the 0.858, and the Wikifier tool (which achieved the 0.862).

1.4.9 ReFinEd

*ReFinEd*³⁶ (Representation and Fine-grained typing for Entity Disambiguation) is an end-to-end entity linking model that uses fine-grained entity types and entity description to perform linking and entity disambiguation for the mentions within a document. The combination of information from entity types and descriptions in a simple transformer-based encoder returns performance that improves the entity linking performance. ReFinEd is also a ‘zero-shot-capable entity-linking tool’, meaning it allows liking to entities that are not included in the training data (Ayoola et al. 2022, 2). It uses Wikidata as a target knowledge base, mainly because Wikidata enables to link 15 times more entities compared to Wikipedia. In addition, the model can also be trained by following the instructions provided in its code repository³⁷.

ReFinEd project lacks a demo for users to try the model. For this reason, its implementation is necessary to experience the performance of this tool; hence, the Flying Digital Editions project has decided to adopt this model and testing it out while developing. ReFinEd makes available several trained models that are ready to use, and different settings, such as the configuration of the ‘entity_set’ to resolve against Wikidata or Wikipedia entities. The model is automatically downloaded at first run of the code in the local directory form S3, which is the Amazon Storage Service³⁸.

By providing the sample text to the model, the printed result shows the list of entities with the corresponding Wikidata identifier, the item title and, occasionally, the categorisation class (Figure 15).

³⁴ <https://github.com/kermitt2/entity-fishing?tab=readme-ov-file>

³⁵ <https://nerd.readthedocs.io/en/latest/evaluation.html>

³⁶ <https://github.com/amazon-science/ReFinED/tree/main>

³⁷ <https://github.com/amazon-science/ReFinED/blob/main/TRAINING.md>

³⁸ https://en.wikipedia.org/wiki/Amazon_S3

```

spans [['Philolaus', Entity(wikidata_entity_id=Q212338, wikipedia_entity_title=Philolaus), 'PERSON'], ['Aristarchus of Samos', Entity(wikidata_entity_id=Q140188, wikipedia_entity_title=Aristarchus of Samos), 'PERSON'], ['Pythagoreans', Entity(wikidata_entity_id=Q191359, wikipedia_entity_title=Pythagoreanism), None], ['Anaximander', Entity(wikidata_entity_id=Q42458, wikipedia_entity_title=Anaximander), 'PERSON'], ['Numa Pompilius', Entity(wikidata_entity_id=Q200031, wikipedia_entity_title=Numa Pompilius), 'PERSON'], ['Vesta', Entity(wikidata_entity_id=Q178710, wikipedia_entity_title=Vesta (mythology)), None], ['Democritus', Entity(wikidata_entity_id=Q41980, wikipedia_entity_title=Democritus), 'PERSON'], ['Eudoxus', Entity(wikidata_entity_id=Q185150, wikipedia_entity_title=Eudoxus of Cnidus), 'PERSON'], ['Calippus', Entity(wikidata_entity_id=Q471218, wikipedia_entity_title=Calippus), 'PERSON'], ['Aristotle', Entity(wikidata_entity_id=Q868, wikipedia_entity_title=Aristotle), 'PERSON']]

```

Figure 15: Example of ReFinEd annotation process results.

As pointed out in the GitHub project repository, the complete response of ReFinEd annotation process also includes a lot of other information (see Figure 16).

```

(10) [{"-", "-", "-", "-", "-", "-", "-", "-", "-", "-"}] t
  0:
    candidate_entities: Array(5)
      0: Array(2)
        0: "Q212338"
        1: 0.8154
        length: 2
        [[Prototype]]: Array(0)
      1: (2) ["Q116169", 0.1385]
      2: (2) ["Q3745492", 0.0154]
      3: (2) ["Q107349808", 0.0154]
      4: (2) ["Q3381153", 0.0154]
        length: 5
        [[Prototype]]: Array(0)
      coarse_mention_type: "PERSON"
      coarse_type: "MENTION"
      date: null
      doc_id: 290963452
      entity_linking_model_confidence_score: 0.9924
      failed_class_check: null
      gold_entity: null
      ln: 9
    predicted_entity:
      human_readable_name: null
      parsed_string: null
      wikidata_entity_id: "Q212338"
      wikipedia_entity_title: "Philolaus"
      [[Prototype]]: Object
    predicted_entity_types: Array(2)
      0: (3) ["Q16727193", 'humanities scholar', 0.9887]
      1: (3) ["Q901", 'scientist', 0.535]
        length: 2
      [[Prototype]]: Array(0)
      pruned_candidates: null
      start: 528
      text: "Philolaus"
      top_k_predicted_entities: (6) [Array(2), Array(2), Array(2), Array(2), Array(2), Array(2)]

```

Figure 16: Example of complete response of ReFinEd annotation process.

An example is provided with the entity ‘Philolaus’ of the sample text, for which the model recognises a list of entities that could match with the one detected and the predicted entity type. There are also some quantitative scores, such as the confidence one.

For what concern its performance, ReFinEd is one of the best entity-linking tools currently available. To measures the entity detection performance on non-Wikipedia entities, such as the Wikidata ones, the authors evaluated their model on the ISTEEX dataset. It outperforms the OpenTapioca model with an F1 score of 92.1 vs 87.0. On the WebQSP dataset, the authors tested the entity-linking performance on questions and the results was to a F1 score of 89.1 compared to TagMe, which achieved only the 36.1 and a similar score (89.0) to the ELQ model, which is optimised for questions (Ayoola et al. 2022, 10–11).

On other datasets, such as the AIDA-CoNLL (Table 1), MSNBC or Derczynski, it is always in the top 5 positions as ‘best model’³⁹ for the entity-linking process.

Model	AIDA-CoNLL dataset (F1 scores)
OpenTapioca	0.399
entity-fishing	0.765
ReFinEd	0.849

Table 1: F1 score on AIDA-CoNLL dataset

³⁹ <https://paperswithcode.com/task/entity-linking#task-home>

1.5 Preliminary observations

After the analysis conducted on the previous paragraphs regarding the state of the art, and before continuing to explain how FDE was build, it is necessary to shortly summarise and define the results emerged from this search and what have been the choices made following the comparison of the different tools.

By analysing knowledge bases currently available, the choice of Wikidata was in fact easy and almost effortless. This knowledge base, which is completely open and accessible thanks to its license, provide the possibility to have always updated data, reliable due to the source references and it is comprehensive of millions of items, which enhance the probability of finding the entity wanted. In fact, Yago is based on Wikidata itself, hence it would make no sense for the purposes of this project to use a restricted base when you have access to the original and complete data. DbPedia, instead, is based on Wikipedia, which has a significantly fewer entities than Wikidata, which includes all the items in the Wikimedia projects.

The idea of adding a panel inspired by Google Knowledge Panels was also a perfect solution due to the familiarity that users already have with this kind of interface and the fact that this kind of visualisation is easy to read and understand, unlike a graph. Plus, the information retrieved must be handled and adapted, it cannot be just displayed as happens, for example, with the entity-fishing tool, which just previews the page of Wikipedia.

Once decided the knowledge base, the choice of ReFinEd as the model to detect and annotate the entities came as a natural consequence. This model is one of the best entity linking models currently available⁴⁰, also specifically for the Wikidata knowledge base (Ayoola et al. 2022, 11). Plus, it does not have a developed user interface, which makes it a perfect option for testing it within the Flying Digital Editions project. Differently from the other

One of the critical points found out during this comparative analysis is that non-expert users are not considered at all as possible ‘consumers’. All the ways for utilising these tools are – in the most cases – extremely technical and specialistic, which means that a person with no expertise is heavily penalised, and often not interested because they cannot understand what to do and what kind of use these tools can have. Some of the ones investigated are simply a ‘face’ for the API working in

⁴⁰ <https://paperswithcode.com/sota/entity-linking-on-webqsp-wd>

background, meaning that they do not provide a real ‘reading experience’ of the text. In fact, once the annotation process is completed, the text is often just displayed, the entities underlined, and no other feature is given to the user. Only Wikifier tries to create a list of the entities contained in the text to provide users an overview of what the process has returned, even if it is a bit confusing, while others do not even reproduce the new annotated text, leaving the user the task of understanding where each detected entity is collocated inside the document. In addition to that, excluding the entity-fishing tool which provides a little summary panel for each of the entities, all the others do not provide any further information, forcing the users to interrupt its reading for retrieving what they need. But even in this case, what is shown is not remarkably interesting because the focus is on the domain of the entity and on the confidence with which the entity itself has been recognised.

In some cases, such as the Falcon 2.0 one, these tools are not designed to analyse long texts, which reduce the possibility to leverage them for daily need – for example to analyse an article or a chapter of a book.

All the existent analysed tools do not allow to edit the response of the entity detection and linking process. For example, on DBpedia Spotlight or also in OpenTapioca, the text is immediately shown with the entities underlined (or highlighted). If some of the entities are recognised wrong, the user does not have any control over it, which makes difficult to handle the text for specific purposes. For example, if teachers must show the annotated text to their students, they must do it also including the incorrectness, unless they have enough expertise to take the response and handle differently – e.g. by including them inside other systems capable of such tasks.

Hence, to answer to the questions emerged after defining the research investigation, we can state that yes, there are tools that provides a kind of creation of digital editions, *but* they are not strictly thought to do so. In fact, they are designed to simply annotate the text, without having a specific aim to which employ the result of the process. Their work is limited to displaying the result and – eventually – to provide some statistic information about the linking process. In addition to that, sometimes the performance is not good enough and the outcome is littered with inconsistencies and inaccuracies. This is why, by comparing the F-scores of the various tool and looking at the best models currently available, FDE has chosen ReFinEd as the one for annotating the text.

Due to what just explained, these tools are not suitable for non-expert users, who would be obliged to employ additional systems to effectively exploit the results of these annotations. For this reason, the need of having all these tasks just in one place seems to be valid experiment to be conducted.

Therefore, Flying Digital Editions project tries to take all this issues and to create a comprehensive and complete tool for analysing a text and creating a light-version of a digital edition for any kind of user and for different purposes. In the following chapter, it will be explained how this tool has been developed along with the features it has been equipped with.

2. Development

2.1 Setup

Flying Digital Editions is a tool that has been developed to be extremely flexible and easy to adapt to users' needs. Attempting to ensure an application that exploits the latest technologies, the main languages used are Python, Typescript (with Angular framework), and SPARQL.

In the following paragraphs, a brief description will be provided with some specific information about the adopted modules and versions. In addition, because the tool provides the possibility to download the information panel in turtle format, section 2.1.5 will shortly discuss this feature.

Python

Python⁴¹ is a general-purpose programming language that has the advantage of code readability due to the use of significant indentation ('Python', 2024). All its functionalities are not available into its core, but it was designed to be extensible via modules. The Python's version used in the development of the FDE system is the 3.10.1, but currently the latest stable one is the 3.12.1 (8 December 2023), which can be easily updated through the installer provided by the official website.

In this project, several modules have been used to develop the system functionalities. Following, a brief explanation of each of them with the adopted versions:

- `flask` (v. 2.3.3)⁴²: it is a so-called 'microframework', since it does not require additional tools or libraries. It can be used to create online applications and APIs ('Flask', 2023);
 - `flask_cors`: it is an extension that allows cross origin resource sharing (CORS) by using a decorator.
- `Waitress` (v. 3.0.0)⁴³: it is a pure Python WSGI (Web Server Gateway Interface) server. It is used to run a flask application in a production environment.
- `refined` (v. 1.0.0): it is the module that must be imported to use the ReFinEd entity-linking tool (see [1.4.9 ReFinEd](#));

⁴¹ <https://www.python.org/>

⁴² <https://flask.palletsprojects.com/en/3.0.x/>

⁴³ <https://flask.palletsprojects.com/en/3.0.x/deploying/waitress/>

- `SPARQLWrapper`(v. 2.0.0)⁴⁴: it is a Python wrapper around a SPARQL service used to remotely execute queries. It allows to manage the result in a more suitable format.
- `rdflib` (v. 7.0.0)⁴⁵: it is a package used for handling and manipulating RDF constructs.
- `sys` (System-specific parameters and functions)⁴⁶: it is a module that provides access to variables and functions interacting with the interpreter.
- `datetime`⁴⁷: it is a module that supplies classes for manipulating dates and times.
 - `dateutil` (v. 2.8.2)⁴⁸: it is an extension to the standard Python module.

For installing all the packages specified in the file `requirements.txt` can be used the following command inside the terminal:

```
pip install -r -requirements.txt
```

TypeScript

TypeScript⁴⁹ is Microsoft's developed high-level programming language, free, open, and strongly typed that adds static typing to JavaScript. It is considered as a superset of JavaScript ('TypeScript', 2024), hence it can use type inference to give tooling without additional code. The version in use is the 5.1.6, although the latest stable release is the 5.3.2 (20 November 2023).

Angular

Related with the use of TypeScript is the implementation of the development platform Angular⁵⁰. Indeed, Angular is a single-web application framework based on TypeScript, free and open-source; it is the result of rewriting AngularJS⁵¹, which is based on JavaScript instead. Both are maintained (mainly) by Google and by a community of individuals and corporations ('Angular', 2024).

⁴⁴ <https://sparqlwrapper.readthedocs.io/en/latest/>

⁴⁵ <https://rdflib.readthedocs.io/en/stable/>

⁴⁶ <https://docs.python.org/3/library/sys.htm>

⁴⁷ <https://docs.python.org/3/library/datetime.html>

⁴⁸ <https://dateutil.readthedocs.io/en/stable/>

⁴⁹ <https://www.typescriptlang.org/>

⁵⁰ <https://angular.io/>

⁵¹ <https://en.wikipedia.org/wiki/AngularJS>

For maintaining and developing Angular applications, the users can work with a command-line interface tool, called Angular CLI, directly from a command shell. It allows you to create projects, start the application, run tests, and generate components, which are the ‘fundamental building block’ for creating applications. Angular CLI is based on Node.js⁵², the open-source runtime environment, hence having it installed is required to use it. To develop a scalable and maintainable code, Angular helps to configure the project into organised parts with clear responsibilities. For a component, it is possible to define functions and, most importantly, each component has an HTML template that represents what will be rendered to the DOM and that can be defined in the same file of the component class or in a separate one. The same happens for what concerns the style.

Angular gives the possibility to install several libraries for different purposes. An example is the *RxJS*⁵³ library, which provides utility functions for creating and working with observables, an event handling technique and asynchronous programming. About pre-build, ready-to-use components that can be easily implemented into projects, one of the main libraries is PrimeNg.

*PrimeNg*⁵⁴ is a User Interface (UI) component library developed by PrimeTek Informatics (‘PrimeNg’, 2024) that offers a wide range of components, such as modals, inputs, and more. It follows Material Design guidelines, providing a consistent look and style and simplifies the implementation of complex features because the components are ready to use and actively maintained, with continuous updates and performance improvements. With this library it is possible also to create customizable application style templates or reuse some already existing and also provides a default (but optional) icon library that can be installed later with the proper command. For those interested, PrimeNg provides a ‘Playground’ page in which the user can interactively experience the tool.

⁵² <https://nodejs.org/en>

⁵³ <https://angular.io/guide/rx-library>

⁵⁴ <https://primeng.org>

Following, the list of the installed versions used in the Flying Digital Edition project:

- Angular v. 16.2.7
- Angular CLI v. 16.2.4
- RxJS v. 7.8.0
- PrimeNg v. 16.4.1
 - PrimeNg Icons v. 6.0.1

For downloading all the dependencies can be used the following Node.js command, that will install everything the project needs:

```
npm install
```

SPARQL

SPARQL⁵⁵ - SPARQL Protocol and RDF Query Language ('SPARQL', 2024) - is a semantic query language for databases used to retrieve and manipulate data stored in RDF format, which is a direct labelled graph for representing information in the Web (see [1.1](#)). SPARQL can be used to express queries across several data sources and contains capabilities for querying graph patterns ('SPARQL Query Language, 2013). The queries can be distributed to multiple SPARQL endpoints, which are services that accept SPARQL queries and return results, sets or RDF graphs.

The SPARQL language specify four query variation for different purposes:

- SELECT, which returns results in a tabular form;
- CONSTRUCT, which transform the result into new RDF graph;
- ASK, returns a boolean (True/False) result.

In the Flying Digital Edition project, the queries have been constructed using the SELECT clause against Wikidata knowledge base (see [1.2](#)). In fact, Wikidata provides both a Query Service

⁵⁵ <https://www.w3.org/TR/sparql11-overview/>

(WDQS)⁵⁶, a tool through which a user can easily construct queries and train with SPARQL language, and a SPARQL endpoint⁵⁷, to which queries can be submitted directly with a GET or a POST request.

GET requests must have the query specified in the URL, while the POST ones can accept the query in the body, avoiding length limits. The results format is XML by default, but can also be set on JSON, TSV, CSV, and Binary RDF.

To explain how to construct a SPARQL query, a simple example is provided from the Wikidata Tutorial (Figure 17). The triple `?child`, `parent`, and `Bach` can be read as a sentence consisting of a subject, a predicate and an object. The `WHERE` block contains restrictions on the variables that the query service will try to fill with values at the run and the variables that will be shown in the results are the ones selected into the `SELECT` clause.

```
SELECT ?child
WHERE
{
  # ?child father Bach
  ?child wdt:P22 wd:Q1339.
}
```

Figure 17: Example 1 of a SPARQL `SELECT` query from Wikidata tutorial.

Wikidata items and properties are not identified by human-readable names, but they have a specific alphanumeric identifier: Q-number for the previous and P-number for the latter, both to be prefixed with, respectively, `wd:` and `wdt:` as in Figure 18.

⁵⁶ <https://query.wikidata.org/>

⁵⁷ https://www.mediawiki.org/wiki/Wikidata_Query_Service/User_Manual#SPARQL_endpoint

```

SELECT ?child
WHERE
{
  # child "has parent" Bach
  ?child parent Bach.
  # (note: everything after a '#' is a comment and ignored by WDQS.)
}

```

Figure 18: Example 2 of a SPARQL SELECT query from Wikidata tutorial.

However, the query so constructed does not return the values of the item, but the item itself, namely its identifier. To overcome this situation, it is possible to add a clause (Figure 19) that allows retrieval of multilingual labels for Wikidata items. To see the result, the user must put the suffix ‘Label’ to the interested variable inside the SELECT clause.

```

SERVICE wikibase:label { bd:serviceParam wikibase:language "[AUTO_LANGUAGE]". }

```

Figure 19:SERVICE clause from Wikidata tutorial.

Turtle

Turtle⁵⁸ (Terse RDF Triple Language) is both a syntax and a file format that can be used to express data in the RDF data model. Similar to the SPARQL language, Turtle provides a way to create triples by grouping three URIs (Figure 20), abbreviating this information (‘Turtle’, 2023). It is a textual representation of an RDF graph, and it is an alternative to the original syntax and standard for writing RDF, namely RDF/XML, because it does not rely on XML and it is more readable and easier to edit.

⁵⁸ <https://www.w3.org/TR/turtle/>

EXAMPLE 1

```
@base <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rel: <http://www.perceive.net/schemas/relationship/> .

<#green-goblin>
  rel:enemyOf <#spiderman> ;
  a foaf:Person ;    # in the context of the Marvel universe
  foaf:name "Green Goblin" .

<#spiderman>
  rel:enemyOf <#green-goblin> ;
  a foaf:Person ;
  foaf:name "Spiderman", "Человек-паук"@ru .
```

Figure 20: Example of a Turtle document describing the relationship between Green Goblin and Spiderman provided by W3C Documentation⁵⁹

2.2 Code organisation

Flying Digital Editions keeps the division between client-side and server-side. The server-side is constituted by a Python API that manages all the backend logic concerning the retrieval of the data, then organised into a JSON object, and obtained by the Angular application that works as client and displays the information.

In the next paragraphs, the organisation of the project code will be briefly reproduced to provide a clearer explanation of the structure.

2.2.1 Server-side

The Python server of the FDE application consists of four files: ‘main.py’, ‘disambiguation.py’, ‘wikiquery.py’ and ‘queries.py’.

To run the server on premises, it is necessary to use the command:

⁵⁹ *Ibidem*

```
waitress-serve --host:localhost --port=8888 main:app
```

main.py

The file `main.py` is the starting point for the retrieval of the data that will be displayed to the final user. By using the `flask` module (see [2.1.1](#)), two routes are defined. The first is the one associated with the address `/getEntities` of the server that responds to POST requests (Figure 21). `getEntities()` is the function associated with the route in charge to retrieve the JSON data from the request by using the key `['text']`. The text retrieved will be analysed with the class `Disambiguation()`, that will be discussed later, and the result of this analysis will be stored into a variable that will be sent back to the client-side.

```
15 @app.route('/getEntities', methods=['GET', 'OPTIONS', 'POST'])
16 def getEntities():
17     if request.method == 'OPTIONS':
18         response = jsonify(success=True)
19         response.headers['Access-Control-Allow-Methods'] = 'GET, POST, OPTIONS'
20         response.headers['Access-Control-Allow-Headers'] = 'Content-Type'
21         return response
22
23     if request.method == 'POST':
24         try:
25             data = request.get_json()
26             text = data['text']
27             findEntity = Disambiguation()
28             results = findEntity.convertText(text)
29             return jsonify(results)
30         except Exception as e:
31             return jsonify(success=False, message="Error processing request")
32
33     return jsonify(success=False, message="Invalid request method")
```

Figure 21: endpoint `/getEntities` responding to POST requests.

The second route defined in this file is associated with the function `getEntityInfo()` that takes two parameters as input: `type` and `id` (Figure 22). This route can respond only to GET requests and, by calling the `WikiQuery()` class's method `findEntityInfo()`, it returns a JSON object containing the information regarding the specific identifier sent as input.


```

35 @app.route('/<string:type>/<string:id>', methods=['GET', 'OPTIONS'])
36 def getEntityInfo(type, id):
37     if request.method == 'OPTIONS':
38         response = jsonify(success=True)
39         response.headers['Access-Control-Allow-Methods'] = 'GET, HEAD, OPTIONS'
40         response.headers['Access-Control-Allow-Headers'] = 'Content-Type'
41         return response
42
43     if request.method == 'GET':
44         info = WikiQuery()
45         result = info.findEntityInfo(type, id)
46         return jsonify(result)
47
48     return jsonify(success=False, message="Invalid request method")
49

```

Figure 22: endpoint '/type/id' responding to GET requests.

disambiguation.py

The `Disambiguation()` class contains one of the most important methods of the project, which is the function that analyses the text for retrieving the entities. `convertText()`, called directly from the `main.py` file, takes as input the text to be processed (Figure 23). Then, the `Refined` class is instantiated and its method `process_text()` is invoked.

The results, stored firstly into a variable 'spans', are iterated with a for-loop. For each span, i.e. the recognized entity, some information are selected among those that the `ReFinEd` tool makes available:

- `text`: this property represents actually the name of the entity recognized by the entity-linking system;
- `wikidata_entity_id`: retrieved from the previous level 'predicted_entity', it represents the Wikidata ID corresponding to item's name;
- `candidate_entities`: it is the list of all the possible entities that could match the text property. From this list will be selected only the items which start with the character 'Q', because it is how an entity identifier is defined in Wikidata (see [2.1.4](#)).
- `coarse_mention_type`: this property represents the type detected by default from `ReFinEd`. It is a general categorization of the entity. Sometimes it cannot be returned.

- `predicted_entity_types`: this property is accessed only if the previous one results 'None'. In this case, the entity type is still filled with a predicted value.

The method performs two types of checks: the first ensures that all the main and necessary information are not None. If this check passes, then a list, `entities`, will be populated with all the entity's information. An entity will have the structure of a key-value object that maps the information retrieved: 'Name', 'ID', 'Type' and 'Candidates'. If the first control is not satisfied, a new check ensures that the entity appended to the list is valued with the name, the id and the type, this time retrieved from `predicted_entity_types`. Finally, the list will be returned.

```

8     def convertText(self, text):
9         refined = Refined.from_pretrained(model_name="wikipedia_model", entity_set="wikidata")
10        spans = refined.process_text(text)
11        entities = []
12        entity_list = []
13        for span in spans:
14            entity_id = span.predicted_entity.wikidata_entity_id
15            candidate_list = span.candidate_entities
16            for item in candidate_list:
17                if item[0].startswith("Q"):
18                    entity_list.append(item[0])
19            entity_name = span.text
20            entity_type = span.coarse_mention_type
21            if entity_id is not None and entity_name is not None and entity_type is not None:
22                entities.append({"Name": entity_name, "ID": entity_id, "Type": entity_type, "Candidates": entity_list})
23            elif entity_id is not None and entity_name is not None and entity_type is None:
24                entity_type = span.predicted_entity_types[0][1].capitalize()
25                entities.append({"Name": entity_name, "ID": entity_id, "Type": entity_type, "Candidates": entity_list})
26        return entities

```

Figure 23: `convertText()` method for entity recognition and linking with ReFinEd.

wikiquery.py

The `WikiQuery` class is the one that manages the construction of the queries against the Wikidata SPARQL endpoint and the transformation of the results into objects that can be easily used in the Angular project. When the user tries to execute a query on the Wikidata Query Service⁶⁰, the system provides code for implementing the GET request according to several programming languages. For the FDE's purposes the Python method `get_results()` have been used for running the queries. It takes as inputs the endpoint url and the text of the query; by using `SPARQLWrapper` (see [2.1.1](#)), the

⁶⁰ <https://query.wikidata.org/>

variable ‘sparql’ encapsules the full SPARQL call, and then both query and return format are set. The function will return the queries’ results converted into the desired format.

`findEntityInfo()` is the method that takes as inputs the type and the id of the searched entity. It is called directly from the `main.py` file and it is constructed as a match-case statement. If the parameter ‘type’ matches one of the cases, another method is invoked in return. Currently, only four types are defined - ‘Person’, ‘Location’, ‘Space’ and ‘Other’ -, and each type is associated with a query to be executed against Wikidata. The invoked method takes as parameter the id input, in turn derived from the route. In case of errors, an exception is returned.

Therefore, each matching case will call a specific function, according to the type of the entity. By way of example, just one method will be explained in detail: `getPersonInfo()`.

The first method (Figure 24), as mentioned above, takes as parameter the entity’s id. The id will be then passed to the `personQuery()` function that properly sets the query for calling the `get_results()` method previously discussed.

```
35 def getPersonInfo(self, id):
36     endpoint_url = "https://query.wikidata.org/sparql"
37     query = queries.personQuery(id)
38     try:
39         results = self.get_results(endpoint_url, query)
40         turtle_graph = Graph()
41
42         for res in results["results"]["bindings"]:
43             person_label = self.checkIfEmpty(res.get("personLabel", {}).get("value", ""))
44             description = self.checkIfEmpty(res.get("personDesc", {}).get("value", "").capitalize())
45             gender = self.checkIfEmpty(res.get("genderLabel", {}).get("value", "").capitalize())
46             birth_date = self.checkIfEmpty(self.formatDate(res.get("dateOfBirth", {}).get("value", "")))
47             death_date = self.checkIfEmpty(self.formatDate(res.get("dateOfDeath", {}).get("value", "")))
48             place_of_birth = self.checkIfEmpty(res.get("placeOfBirth", {}).get("value", "").capitalize())
49             place_of_death = self.checkIfEmpty(res.get("placeOfDeath", {}).get("value", "").capitalize())
50             occupation_label = self.checkIfEmpty(res.get("occupationsLabels", {}).get("value", "").capitalize())
51             image_url = self.checkIfEmpty(res.get("image", {}).get("value", ""))
52             viafID = self.checkIfEmpty(res.get("viafID", {}).get("value", ""))
53             wikipedia_url = self.checkIfEmpty(res.get("wiki_page", {}).get("value", ""))
54
55             subject = URIRef("http://www.wikidata.org/entity/" + id)
56             turtle_graph.add((subject, RDF.type, URIRef("https://schema.org/Person")))
57             turtle_graph.add((subject, URIRef("https://schema.org/givenName"), Literal(person_label)))
58             turtle_graph.add((subject, URIRef("https://schema.org/description"), Literal(description)))
59             turtle_graph.add((subject, URIRef("https://schema.org/gender"), Literal(gender)))
60             turtle_graph.add((subject, URIRef("https://schema.org/birthDate"), Literal(birth_date)))
61             turtle_graph.add((subject, URIRef("https://schema.org/deathDate"), Literal(death_date)))
62             turtle_graph.add((subject, URIRef("https://schema.org/hasOccupation"), Literal(occupation_label)))
63             turtle_graph.add((subject, URIRef("https://schema.org/image"), URIRef(image_url)))
64             turtle_graph.add((subject, URIRef("https://schema.org/birthPlace"), Literal(place_of_birth)))
65             turtle_graph.add((subject, URIRef("https://schema.org/deathPlace"), Literal(place_of_death)))
66             turtle_graph.add((subject, URIRef("https://schema.org/sameAs"), URIRef("https://viaf.org/viaf/" + viafID)))
67             turtle_graph.add((subject, URIRef("https://schema.org/sameAs"), URIRef(wikipedia_url)))
68
69             turtle_data = turtle_graph.serialize(format='turtle')
70
71         return { "type": "Person", "data": { "Name": person_label, "Description": description, "Gender": gender, "Birth Date": birth_date, "Death Date": death_date,
72             "Birth Place": place_of_birth, "Death Place": place_of_death, "Occupation": occupation_label, "Image": image_url, "viafID": viafID, "Wikipedia": wikipedia_url,
73             "Turtle": turtle_data } }
74     except Exception as e:
75         return f"An error occurred: {str(e)}"
```

Figure 24: `getPersonInfo()` method.

The results are iterated according to the structure of the Wikidata SPARQL endpoint response, by using the syntax `results['results']['bindings']`, and a list of variables will be valued correctly only if the returned value is not empty. Otherwise, the `checkIfEmpty()` method assigns a hyphen ('-') to them. Following the list of the data constituting the object that will be displayed into the information panel:

- `person_label`: will be valorized with the entity name;
- `description`: a brief description of the entity;
- `gender`: the gender of the person (if known);
- `birth_date`: the date of the birth (if known);
- `death_date`: the date of the death (if known);
- `place_of_birth`: the place of the birth (if known);
- `place_of_death`: the place of the death (if known);
- `occupation_label`: the list of the professions of the entity;
- `image_url`: an image retrieved from Wikidata (if any);
- `viafID`: identifier for the Virtual International Authority File⁶¹ database (if any);
- `wikipedia_url`: the corresponding Wikipedia page for the entity.

Concurrently, the method instantiates the `Graph` class from `rdflib`. The triples are constructed using the subject defined with the Wikidata ID, the ontology - specifically `Schema.org`⁶² - and the value. The turtle graph is then serialised. After the end of the iteration, `getPersonInfo()` returns an object with two keys: 'type' and 'data'. The first defines the general categorization of the entity, while the second receives all the variables previously valued, including the turtle data.

queries.py

This file contains the 'core' of the creation of the information panel, the queries. Inside `queries.py` there are four simple functions - `personQuery`, `locationQuery`, `spaceObjQuery` and `defaultEntityQuery` - that takes as parameter an id and returns the text of the query that will be run against Wikidata (Figure 25). The first three queries reflect the types that the system can recognise

⁶¹ <https://viaf.org/>

⁶² <https://schema.org/>

and assign to the entities, while the last one is specially created to be a ‘default case’, if the type is not detected as known. It retrieves just the basic information such as the name of the entity, the description, the image (if any), the VIAF id and eventually the Wikidata page.

The text of the query is written in SPARQL language (see [2.1.4](#)) and shows some specific clauses, such as the `BIND` for binding the id to a variable or the `OPTIONAL`, which allows to extend the results if there is a match.

```

103 def spaceObjQuery(id):
104     return """
105         PREFIX wd: <http://www.wikidata.org/entity/>
106         PREFIX wdt: <http://www.wikidata.org/prop/direct/>
107         PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
108         PREFIX schema: <http://schema.org/>
109
110         SELECT ?spaceObjLabel ?spaceObjDesc ?partOfLabel ?mass ?image ?viafID ?wiki_page
111         WHERE {
112             BIND(wd:""+id+"" AS ?spaceObj)
113
114             OPTIONAL {
115                 ?spaceObj rdfs:label ?spaceObjLabel.
116                 FILTER(LANG(?spaceObjDesc) = "en")
117             }
118             OPTIONAL{
119                 ?spaceObj schema:description ?spaceObjDesc.
120                 FILTER(LANG(?spaceObjDesc) = "en")
121             }
122             OPTIONAL{
123                 ?spaceObj wdt:P361 ?partOf.
124                 ?partOf rdfs:label ?partOfLabel.
125                 FILTER(LANG(?partOfLabel) = "en")
126             }
127             OPTIONAL{
128                 ?spaceObj wdt:P2067 ?mass.
129             }
130             OPTIONAL{
131                 ?spaceObj wdt:P18 ?image.
132             }
133             OPTIONAL {
134                 ?spaceObj wdt:P214 ?viafID.
135             }
136             OPTIONAL {
137                 ?wiki_page schema:about ?spaceObj.
138                 ?wiki_page schema:inLanguage "en".
139                 FILTER (SUBSTR(str(?wiki_page), 1, 25) = "https://en.wikipedia.org/")
140             }
141             SERVICE wikibase:label { bd:serviceParam wikibase:language "en". }
142         }
143         LIMIT 1
144     """

```

Figure 25: `spaceObjQuery()` function query for retrieving data from Wikidata.

2.2.2 Client-side

The client side of this tool consists of the Angular project. As explained in [2.1.3](#), this framework works with components, meaning blocks with an HTML template, a TypeScript class defining the behaviour and a CSS selector to style the template.

In this project, there are four main components that manage the interaction with the user and perform minimal elaborations with the data retrieved from the Python APIs.

To run the client on premises, two commands can be respectively used:

```
npm start  
ng serve -o
```

loading-mode

The loading-mode component represents the first interaction between the tool and the user. The HTML template includes the two inputs for uploading the text to analyse and a button that will send the request. Two approaches have been developed depending on which input has been chosen from the user.

The first detects a change inside the text-area that triggers the method `onTextChange()`, which works like a ‘guard’ to enable the button for sending the text. The value of the input area is saved inside the variable `searchText`, that will be checked directly when the button is clicked.

The second one allows the upload of a document, both in `.txt` or `.doc` format. The change will trigger the `onFileChange()` method, which converts the `.doc` into a format that can be processed by the API.

The method `sendText ()` handles the call to the server for retrieving the text annotated (see Figure 26). For a better performance of ReFinEd, the input text is divided into blocks of sentences, which length can be customised, and for each block the api-service - namely, the service that manages the request to the Python API - is called to annotate the text. When all the text blocks have been iterated, the response is ready to be saved into a variable through another call to the api-service. At the end of this process, the user is directly redirected to another component, which is the edit-mode.

```

32  sendText() {
33      const textToAnalyze = this.fileContent !== null ? this.fileContent : this.searchText;
34      if (textToAnalyze) {
35          this.apiService.setText(textToAnalyze);
36          const maxSentencesPerBlock = 20;
37          const blocks = this.splitTextIntoBlocks(textToAnalyze, maxSentencesPerBlock);
38
39          const allResponses: any[] = [];
40
41          const sendRequest = (index: number) => {
42              if (index < blocks.length) {
43                  const block = blocks[index];
44                  this.apiService.getAnalyzedText(block).subscribe(response => {
45                      allResponses.push(response);
46                      sendRequest(index + 1);
47                  });
48              } else {
49                  this.handleCombinedResponse(allResponses);
50              }
51          };
52          sendRequest(0);
53      } else {
54          this.fileUploaded = false;
55          this.emptyFile = true;
56      }
57  }

```

Figure 26: *sendText()* method

edit-mode

The edit-mode component has the aim to provide the possibility of handling the response retrieved from the entity-linking annotation tool. Because the Flying Digital Editions is a project thought also for teaching and educational purposes, the need of allowing the user to modify the result and be able to change - or add or delete - the entities seemed to be important. The interface will be described in the next section (see 2.3), while in this one will be provided a little overview of the code behind it.

The main method of this component is the method that recreates the text by substituting the named entity with a span. This function, `formatText()`, takes as inputs the original text and the response

provided by the API. The response is essentially an array that will be iterated: for each item of the array, a span will be created with the following characteristics (Figure 27):

- `textContent`: the named entity of the text;
- `className`: two classes will be added to the span, 'mySpan' and the item ID.

```
67 formatText(text: string, response: any) {
68   const flatResponse = response.flatMap((array: any) => array);
69   const sentencesBetweenNewLines = 5;
70
71   flatResponse.forEach((item: any) => {
72     const spanElement = document.createElement('span');
73     spanElement.textContent = item.Name;
74     spanElement.className = 'mySpan ' + item.ID;
75     const spanHtml = spanElement.outerHTML;
76
77     const regex = new RegExp(item.Name, 'g');
78     text = text.replace(regex, spanHtml);
79   });
80   const sentences = text.match(/^[^!?]*((?:[.!?]"')*(?:$))/g) || [];
81
82   let sentenceCountProcessed = 0;
83   let modifiedText = "";
84
85   sentences.forEach((sentence) => {
86     sentenceCountProcessed++;
87     if (sentenceCountProcessed % sentencesBetweenNewLines === 0 && sentenceCountProcessed < sentences.length) {
88       modifiedText += `<br/><br/>`;
89     }
90     modifiedText += sentence;
91   });
92
93   this.editableText = modifiedText;
94   this.fragList = this.response;
95   return this.editableText;
96 }
```

Figure 27: `formatText()` method.

After the creation of the span, all the occurrences will be matched through a regex that substitutes the plain text of the entity with the span itself. To add a little style, the text is divided by some new lines every five sentences. The number can be customised according to the developer.

The other main method of this component is the `handleSpanClick()`. When the user clicks on the span for displaying the information retrieved from the API, he will generate an editing box, provided by another component, which is the clickable-span component. This component is entirely realised with the data that the edit-mode component sets. Specifically, the box valorises the input with the retrieved id and the supposed class (Figure 28). Firstly, the user can check if the ID corresponds to the correct entity by using the Wikidata page linked; eventually, he can choose the correct ID by selecting among the dropdown list of Candidates provided. The method uses the api-service for

setting the new data and updating the existing spans, hence if the user clicks again on the same span, he will immediately see the new data inserted.

```

98   handleSpanClick(spanData: any) {
99     console.log('Span clicked:', spanData);
100    this.selectedSpanData = spanData;
101
102    const factory = this.componentFactoryResolver.resolveComponentFactory(ClickableSpanComponent);
103    this.dynamicComponentRef = this.spans.createComponent(factory);
104
105    if (this.dynamicComponentRef) {
106      const dynamicComponent = this.dynamicComponentRef.instance;
107      dynamicComponent.text = spanData.Name;
108      dynamicComponent.dataId = spanData.ID;
109      dynamicComponent.dataClass = spanData.Type;
110      dynamicComponent.dataList = spanData.Candidates;
111      dynamicComponent.openCard();
112
113      dynamicComponent.updateSpan.subscribe((data: any) => {
114        if (this.componentRef) {
115          this.componentRef.instance.dataList = data.dataList;
116          this.componentRef.instance.selectedValue = data.dataList.length > 0 ? data.dataList[0] : '';
117          this.componentRef.instance.dataId = data.dataId;
118          this.componentRef.instance.dataClass = data.dataClass;
119        }
120        spanData.ID = data.dataId;
121        spanData.Candidates = data.dataList;
122        spanData.Type = data.dataClass;
123
124        this.cdr.detectChanges();
125        this.editableText = this.formatText(this.inText, this.response);
126      });
127
128      this.spans.insert(this.dynamicComponentRef.hostView);
129      if (this.componentRef) {
130        this.componentRef.destroy();
131      }
132      this.componentRef = this.dynamicComponentRef;
133    }
134  }

```

Figure 28: *handleSpanClick()* method.

For what concerns the classification (i.e., the type) of the entity, the `dataClass` is automatically set with what was retrieved with the text analysis process. For helping the user to avoid errors, a dropdown selection list is provided with the class currently supported by the FDE tool (see [2.3](#)).

Finally, another important feature consists in the possibility of deleting the unnecessary entities. When the bin icon is clicked inside the summary panel of the entities, the `deleteEntity()` method is called, taking as input the data of the span. Specifically, the ID is stored in a variable, and it is used inside the div that contains the text for searching the class' span that matches the ID (Figure 29).

Once found, a new document node is created with the `textContent` of the span; then, the span element is replaced with the new text node and the list showing the spans is updated with the new values, by calling the `popOutFromArray()` method.

```
161 deleteEntity(spanData: any) {
162     var myDiv = document.getElementById('book');
163     var targetId = spanData.ID;
164
165     if (myDiv) {
166         if (myDiv.querySelector(`span.${targetId}`)) {
167             var spanEl = this.el.nativeElement.querySelector(`span.${targetId}`);
168
169             if (spanEl && spanEl.textContent !== null) {
170                 console.log("Im in the if");
171                 var textNode = document.createTextNode(spanEl.textContent);
172                 spanEl.parentNode?.replaceChild(textNode, spanEl);
173                 this.popOutFromArray(spanData);
174             } else {
175                 console.log("Span element not found or has no text content.");
176             }
177         } else {
178             console.log("Span element not found in the DOM.");
179         }
180     } else {
181         console.log("Div element not found.");
182     }
183 }
```

Figure 29: `deleteEntity()` method.

When the entity editing process is done, the user can also provide the title and author of the document using the 'Add metadata' button. This information, which is not mandatory, will be shown in the view-mode at the top of the text. To proceed, the user must save the changes by clicking a button that triggers the function `saveText()`. This method performs two operations: firstly, it groups all the different possible types in four general classes that are supported for creating the customisable summary panels. Then, it uses the api-service to send to the following component the text, the list of the spans and the metadata.

view-mode

The view-mode component is the last page of the FDE tool. It represents the result of the previous steps, because it shows a facet that summarises the entities of the text grouped by the type; a box that contains the text, and, when displayed, an information panel for the clicked entity.

```
83
84     spanElement.textContent = item.Name;
85     spanElement.className = 'entitySpan';
86     spanElement.id = item.ID;
87     spanElement.setAttribute('pos', positions.toString())
88     spanElement.setAttribute('type', item.Type);
89
```

Figure 30: Detail of the function `formatText()`.

The text is reconstructed using the same `formatText()` of the edit-mode component, but there are some differences in the creation of the span elements. In this component, the entity span is characterised by two more attributes: the position and the item type (see Figure 30). The type is a necessary information because it is one of the parameters of the function that executes the query for that specific entity. Instead, the position allows the user to click on the arrow inside the facet and scroll directly to the wanted entity. `goToPos()` is the method that manages this feature: it takes in input the position previously calculated when creating the span and then it searches for it inside the div containing. Once found, the whole span is selected and stored into a constant and the method `scrollIntoView` will perform the scrolling (Figure 31).

```
125     goToPos(position: string) {
126         const elements = this.el.nativeElement.querySelectorAll(`[pos="${position}"]`);
127         if (elements.length > 0) {
128             const container = this.el.nativeElement.querySelector('#file');
129             const element = elements[0] as HTMLElement;
130             const entityId = element.id;
131
132             const entityElement = this.el.nativeElement.querySelector(`[id="${entityId}"]`);
133             if (entityElement) {
134                 entityElement.scrollIntoView({ behavior: 'smooth', block: 'center', inline: 'nearest' });
135             }
136         }
137     }
```

Figure 31: `goToPos()` function.

The spans inside the text are characterised by a click event, added dynamically when each span is created. At the click, the function `getIdInfo()` is triggered: this function takes as input the type and the ID of the entity. These two parameters are used for setting, through the info-service, `getEntityInfo()` that performs the call to the server at the `'/type/id'` route for executing queries against Wikidata. Once a response is loaded, the type of the entity is checked and then the service is recalled for setting the entity data to be displayed inside the information panel, with the instruction for opening the panel card.

panel-data

The panel-data component represents the heart of this project. It is the summary of the information retrieved from Wikidata and has the task of displaying them. Actually, the logic behind this component is truly simple and leverages on the data retrieved using the previous component. At the initialization of the component, the `getEntityInformation()` method is called: it fetches the data from the info-service api and stores them inside a constant. If the `entityData` exists, the actual data values an object (`entityInfo`) and the `createPanel()` function is invoked (Figure 32).

```
getEntityInformation(){
  const entityData = this.infoService.getEntityData();
  if(entityData){
    this.entityInfo = entityData.data;
    this.createPanel();
  }
  else
  {
    alert("No information provided for this entity");
  }
}

createPanel(){
  this.keys = Object.keys(this.entityInfo);
  this.entityName = this.entityInfo['Name'];
  return this.keys;
}
```

Figure 32: `getEntityInformation()` and `createPanel()` functions.

The `createPanel()` method keeps the keys of the `entityInfo` object, because the keys will be iterated inside the HTML template to display all the information as a table.

If, for whatever reason, the entity does not return any data, an alert will inform the user of the lack of information.

In addition, the information panel is characterised by the possibility of navigating the Wikipedia page of the entity and downloading a turtle file. The turtle format is frequently used in the context of the semantic web and RDF graphs, because it facilitates data interchange, visualisation and analysis due the structured data representation that makes comprehension easier for machines.

The method `downloadTurtleFile()` is called when a specific field of the panel is clicked. It takes the information stored into the key 'Turtle' and creates a new Blob⁶³ file for which a new URL is created for downloading it.

Services

This section is briefly dedicated to the two services created for handling the passing of data among the components.

The `api.service.ts` is the file dedicated to manage all the information that regards the text analysis and the data of the spans. Specifically, it performs the call to the API endpoint for retrieving the annotations of the texts. It defines the URL, the headers and the request body and returns a POST request (Figure 33).

⁶³ https://it.wikipedia.org/wiki/Binary_large_object

```

55   getAnalyzedText(textBlock: string): Observable<any> {
56       const apiUrl = 'http://127.0.0.1:8888/getEntities';
57       const headers = new HttpHeaders({ 'Content-Type': 'application/json' });
58       const requestBody = { text: textBlock };
59       return this.http.post(apiUrl, requestBody, {headers});
60   }
61

```

Figure 33: `getAnalyzedText()` function.

The `info.service.ts` deals with the request of the specific entity information. In detail, it uses the `getEntityInfo()` function to perform the GET request to the endpoint, taking as input parameters two strings representing the type and the ID of the searched entity (Figure 34).

```

13   getEntityInfo(type:string, id:string){
14       const apiUrl = `http://127.0.0.1:8888/${type}/${id}`;
15       const headers = new HttpHeaders({ 'Content-Type': '*/*' });
16
17       return this.http.get(apiUrl, {headers})
18   }

```

Figure 34: `getEntityInfo()` function.

2.3 A new reading experience

Flying Digital Editions is a project with the aim of giving a new reading experience to the users. The tool, in its first version, is characterised by a simple user-friendly interface that has its strength in the possibility to be used by anyone, also with no specific expertise. In fact, the currently existing tools that provide an entity-linking annotation are also extremely difficult to set if the user has no competence in the matter. As seen in [1.4](#), few existing tools are designed to be used online, indeed most are designed as APIs or libraries for annotating texts or for implementing the analysis response within other projects, and an example can be found in ReFinEd, which is the entity-linking library used in this one.

Flying Digital Edition is instead created to be easy to understand both for people who want to annotate their texts in an accurate way and users who are to benefit from the analysis itself, such as students or researchers. These latter categories often have to analyse texts for their studies or understand journal articles for retrieving information, and the process of selecting one entity at a time and searching for it in a search engine can be long, tiring and stressful. FDE tries to help in these kinds of situations by providing an in-page summary panel, with the main information on the entities. For a complete overview, the tool also makes available the direct link to the Wikipedia page of the entity, so that if the summary is not enough, the user can easily find all the information he wants. The use of Wikidata as a knowledge base guarantees data is always up to date, because the base is maintained by the efforts of the entire community, and reliable, because the source is nearly always provided.

A first version of the project has been developed, and the source code can be found in the GitHub⁶⁴ repository of the project. This version is not currently available online due to some technical issues that will be explained in the next sections. Anyway, the code can be downloaded and run on local machine (see [2.2](#)).

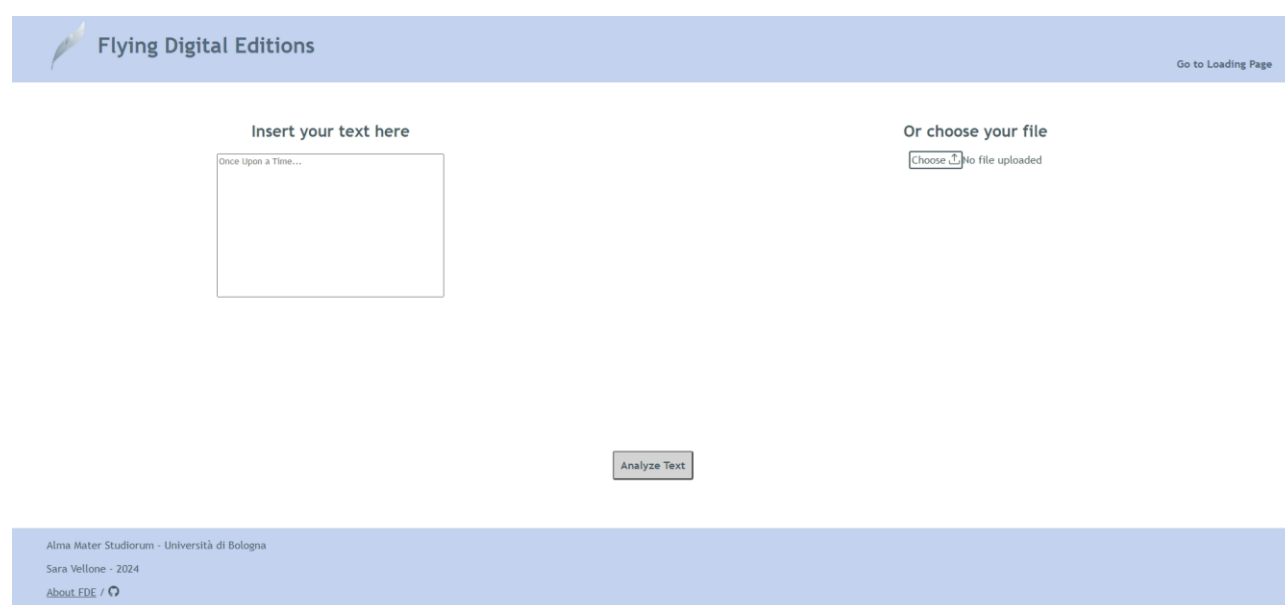


Figure 35: Loading Page of the demo.

The starting page - namely, the loading-mode component - has been thought to be minimal and easy to use. The style of the demo is based on a light blue colour, which should smooth what usually seems complicated and to calm (Cohen 2014).

⁶⁴ <https://github.com/SaraVell1/FDE-project>

The logo represents a leather that inspires a sense of lightness such as the Flying Digital Editions want to, by helping users to create a light version of a digital edition but with all the main features. The header has a direct link to the first page, which is visible from any component of the tool, while the footer groups all the information about the project (Figure 35). The ‘About FDE’ opens a little modal that summarises the aims of this project and its functionalities, and the link to the repository can also be reached by clicking the GitHub icon.

As anticipated in the previous sections, the text can be uploaded through two inputs: a text area and a button for choosing the file from the local archive of the device. The ‘Analyse Text’ button, that leads the user to the next page, can be enabled only if a file is uploaded or the text area is valued. Once the button is clicked, a text with a spinner appears to let the user know that the document is being analysed, and soon the result will be displayed (Figure 36).

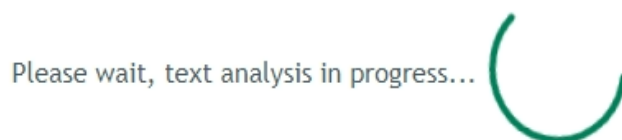


Figure 36: Loading spinner component.

When the process is completed, the user is automatically redirected to the page for allowing the editing of the entities. This page maintains the same simplicity as the previous one, with only the essential features for customising the experience. Specifically, this editing page is designed for those who want to produce an output as precise as possible, such as, for example, a teacher. At the centre of the page there is the text, on which the entities already detected are underlined with a purple-red colour (see Figure 37 and Figure 38).

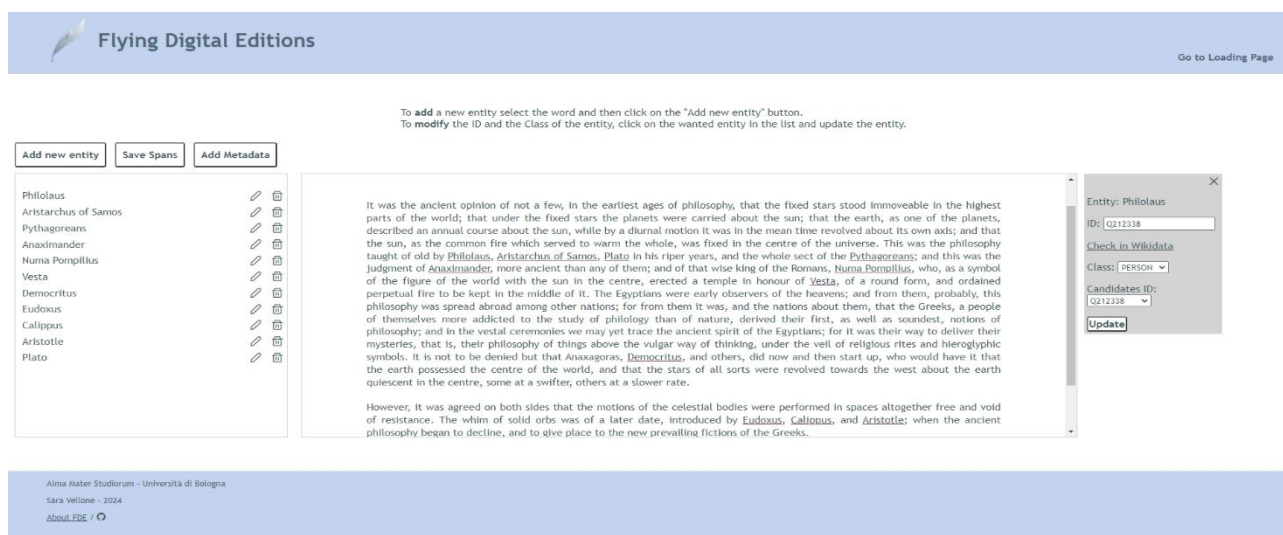


Figure 37: Edit-mode page, an example of entity editing.

It was the ancient opinion of not a few, in the earliest ages of philosophy, that the fixed stars stood immoveable in the highest parts of the world; that under the fixed stars the planets were carried about the Sun; that the Earth, as one of the planets, described an annual course about the sun, while by a diurnal motion it was in the mean time revolved about its own axis; and that the sun, as the common fire which served to warm the whole, was fixed in the centre of the universe. This was the philosophy taught of old by [Philolaus](#), [Aristarchus of Samos](#), Plato in his riper years, and the whole sect of the [Pythagoreans](#); and this was the judgment of [Anaximander](#), more ancient than any of them; and of that wise king of the Romans, [Numa Pompilius](#), who, as a symbol of the figure of the world with the sun in the centre, erected a temple in honour of [Vesta](#), of a round form, and ordained perpetual fire to be kept in the middle of it. The Egyptians were early observers of the heavens; and from them, probably, this philosophy was spread abroad among other nations; for from them it was, and the nations about them, that the Greeks, a people of themselves more addicted to the study of philology than of nature, derived their first, as well as soundest, notions of philosophy; and in the vestal ceremonies we may yet trace the ancient spirit of the Egyptians; for it was their way to deliver their mysteries, that is, their philosophy of things above the vulgar way of thinking, under the veil of religious rites and hieroglyphic symbols. It is not to be denied but that [Anaxagoras](#), [Democritus](#), and others, did now and then start up, who would have it that the earth possessed the centre of the world, and that the stars of all sorts were revolved towards the west about the earth quiescent in the centre, some at a swifter, others at a slower rate.

Figure 38: Detail of the text after the automatic annotation process in the edit-mode.

On the left, there is a panel that includes all the entities retrieved: for each entity, two icons allow the user to modify the entity or delete it (Figure 39).

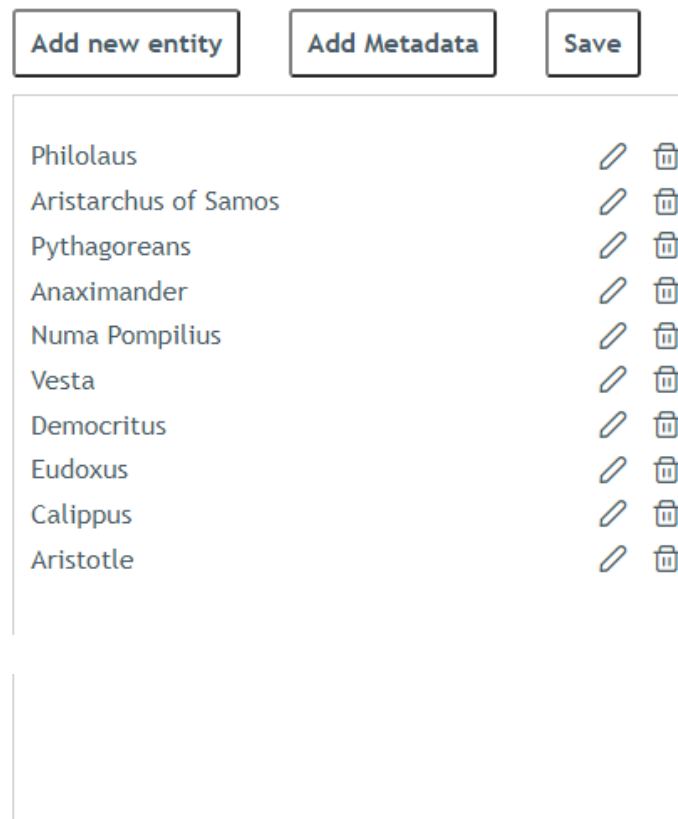


Figure 39: Detail of the summary panel in the edit-mode.

When the pencil icon is clicked, a little box appears on the right of the text. This informative box displays all the main information needed for retrieving the data of the entity in the Wikidata knowledge base. In particular, the name represents the entity selected and cannot be modified. On the contrary, the ID can be both inserted manually or, eventually, can be selected through the Candidate ID list. This list shows all the other possible matched identifiers for that named entity, and to check if the one retrieved is correct, the user is facilitated by the presence of a link to the Wikidata page item, that will open another tab in the browser window so that the editing of the text is not interrupted (Figure 40).

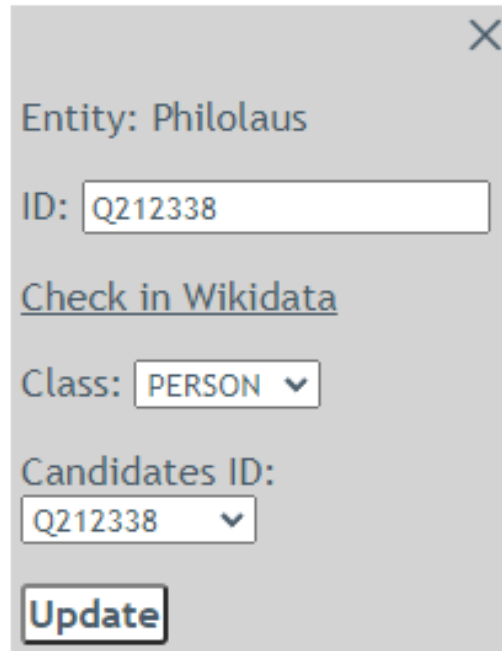


Figure 40: Detail of the card for editing the detected entity in the edit-mode.

For ‘Class’, in this case it is meant the ‘Type’ of the entity selected. By default, the class displayed is the one retrieved by the entity-linking annotation process, but sometimes the types can be extremely specific (for example if the `coarse_mention_type` returns `None`). To generalise as much as possible – so that it is easier to create a template for the query and hence retrieve the data –, the supported classes options are listed in the dropdown, so the user is guided to avoid possible conflicts with the implementation.

If an entity is not considered relevant, for any reason, or the annotation process has made some types of mistakes, it is also possible to delete the unnecessary entity. In this case, the user can simply click on the bin icon and the underline of the text will disappear as well as the named entity in the panel.

Obviously, such as it is possible that an entity is not needed, also it is possible that an entity is it. In this case, by following the few guiding lines at the top of the page, the user can add a new entity. First, he must select the word that he wants to mark as entity and then click on the button ‘Add new entity’. Immediately, a new box will appear (Figure 41). For a correct handling of the entities, it is necessary that they have at least the ID and the type so that they can be searched in Wikidata through the execution of the queries.

In this situation, the Candidates ID list is not displayed, the ID must be added by hand and the Class can be chosen among the available ones.

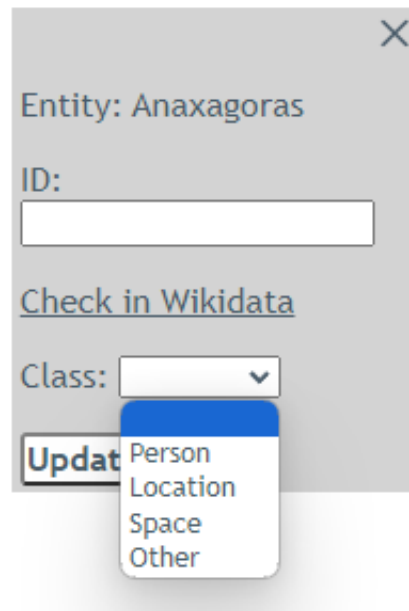


Figure 41: Example of inserting a new entity in the edit-mode.

If the text derives from articles or it is a part of a bigger document, the tool gives the user the possibility to add some metadata. In this first development, the metadata that can be inserted are the *Author* name and the *Title*. At the click of the corresponding button, a little modal, styled in a pastel yellow, appears on the text and allows the input of the information (Figure 42).

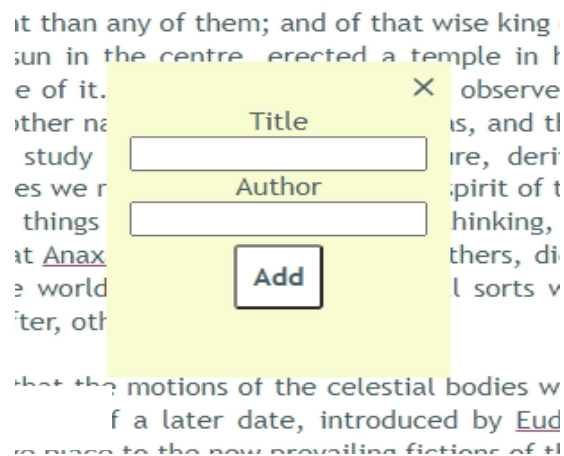


Figure 42: Metadata modal.

Finally, the user arrives on the last page, which is the page that is the most important one, because here is where the Flying Digital Edition completes its realisation. The text is always maintained in the centre to guarantee a visual continuity for the user (Figure 43).



⁶⁵ <https://tccd.libanswers.com/faq/176920>

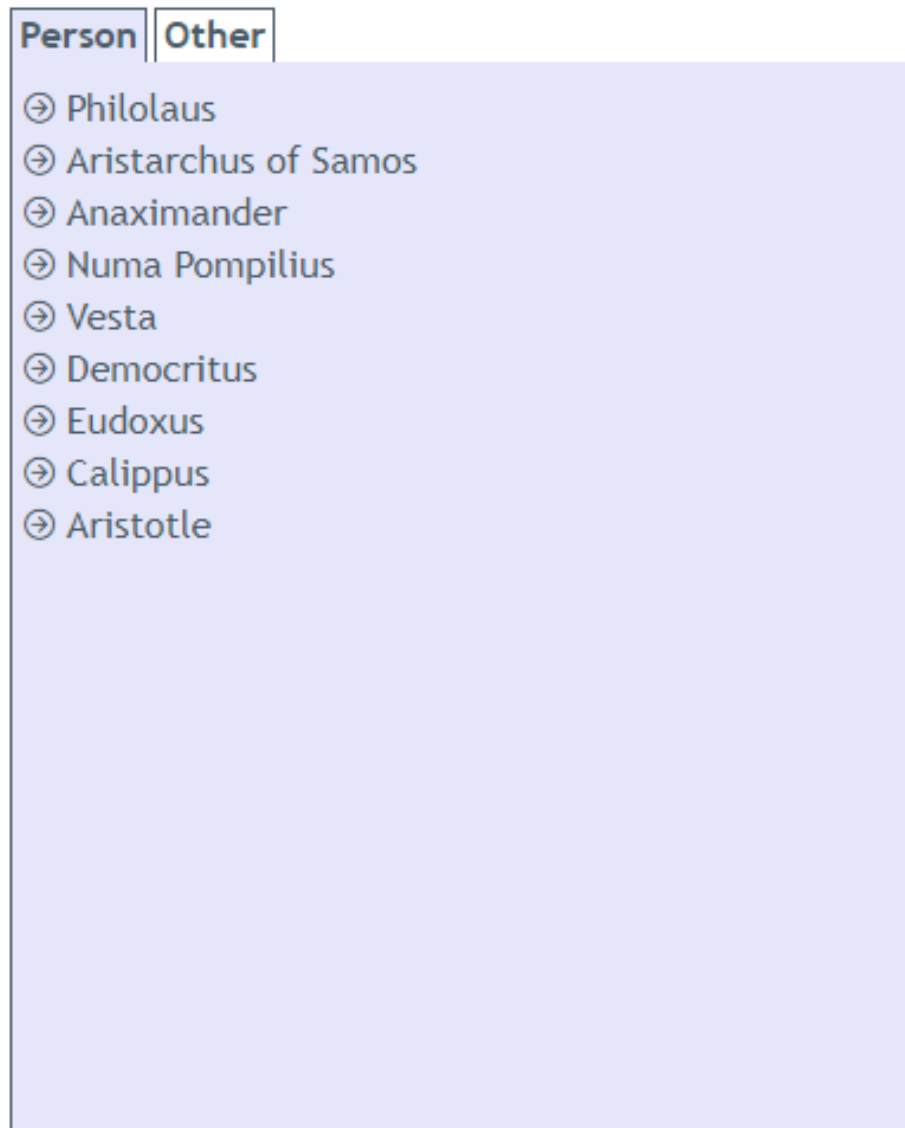


Figure 44: Example of facets in the view-mode.

In the view-mode, the entities are not just underlined, but they are highlighted, so it is easier for the user to understand that he must click on them to visualise some information (Figure 45).

It was the ancient opinion of not a few, in the earliest age immoveable in the highest parts of the world; that under the the Sun; that the Earth, as one of the planets, described a diurnal motion it was in the mean time revolved about its c fire which served to warm the whole, was fixed in the centi taught of old by Philolaus, Aristarchus of Samos, Plato in l Pythagoreans; and this was the judgment of Anaximander, n wise king of the Romans, Numa Pompilius, who, as a symbol the centre, erected a temple in honour of Vesta, of a roun kept in the middle of it. The Egyptians were early observers this philosophy was spread abroad among other nations; for them, that the Greeks, a people of themselves more addicte

Figure 45: Detail of the sample text in the view-mode.

Once found the wanted entity, if the user clicks on the name on the text, a new panel data will appear on the right of the screen. Taking its inspiration from the Google Knowledge Panel, this box includes all the basic data concerning the entity. As already anticipated, for each class there are different types of responses and different information displayed.

For example, for the 'PERSON' type, the main data that are retrieved are those concerning birth and death, in addition to the occupations of the entity (Figure 46).

Numa Pompilius

Image

Description

Legendary second king of rome, succeeding romulus

Birth Date

13 Apr 752 CE

Birth Place

Cures

Death Date

1 Jan 670 CE

Death Place

-

Gender

Male

Occupation

Ancient roman politician, ancient roman priest

Turtle

[Download Turtle](#)

Wikipedia

[Go to Wikipedia](#)

viafID

1460159248511004870003

Figure 46: Example of a "PERSON" entity's information panel.

For what concern all those entities grouped under the umbrella class ‘SPACE’, they will provide information about their mass and if they are part of some super-structure (in Figure 47), the Earth is part of the Earth-Moon system).



Figure 47: Example of a “SPACE” entity’s information panel.

If the entity fell inside the class “Other”, FDE is anyway able to provide few information such as the Wikipedia page and a brief description of the item (Figure 48).

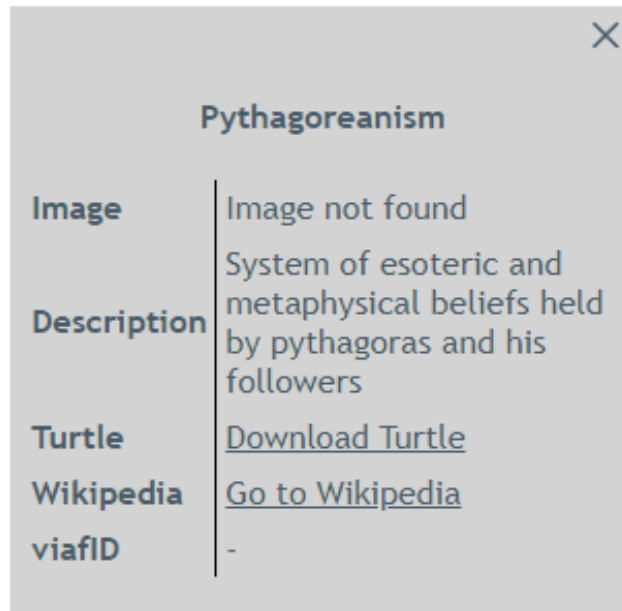


Figure 48: Example of "Other" entity's information panel.

In case any of the information is not available, the corresponding value will be automatically valued with a hyphen. If the Image is not provided, a placeholder will substitute the image itself stating 'No Image found'. Similarly happens if the Wikidata link cannot be retrieved, because the entity is not existent in the Wikidata database.

For each summary panel is possible to download a Turtle file (see Figure 49). Even if it is out of the scope of this project, the Turtle file has been provided to allows the maximum reuse of data. It can also allow some analysis on them, or banally just to store data in a structured and interoperable way, allowing an easy access and retrieving of the data when necessary.

```

1  @prefix schema: <https://schema.org/> .
2
3  <http://www.wikidata.org/entity/Q212338> a schema:Person ;
4      schema:birthDate "1 Jan 469 CE" ;
5      schema:birthPlace "Taranto" ;
6      schema:deathDate "1 Jan 389 CE" ;
7      schema:deathPlace "Thebes" ;
8      schema:description "Greek philosopher, astronomer and pythagorean (c.470-c.385 bce)" ;
9      schema:gender "Male" ;
10     schema:givenName "Philolaus" ;
11     schema:hasOccupation "Music theorist, mathematician, politician, philosopher, writer, astronomer" ;
12     schema:image <-> ;
13     schema:sameAs <https://en.wikipedia.org/wiki/Philolaus>,
14         <https://viaf.org/viaf/50021495> .
15
16

```

Figure 49: Example of downloaded turtle file of the entity.

The entity is defined as the subject with the URI of the Wikidata page of the item, using the type ‘Person’. Several properties are set according to the data retrieved from the queries, and finally the ‘sameAs’ defines the Wikipedia URL and the VIAF ID, if any.

2.3.1 FDE mocked demo

As mentioned in the previous section, it was not possible to put the official project code online. For this reason, a lighter mock version has been created. Although the actual entity-linking process cannot be used, through the demo it was possible to obtain feedback on the visualisation part of the project.

For this demo two GitHub repositories have been created: the first is a fork of the official one and contains the mock server. It is called *FDE-demo-service*⁶⁶ and it has been made available online using Render⁶⁷, which is a hosting platform for web services and static sites. The difference with the original code, lies in the JSON file that provides the ‘entity recognition and entity linking process’ without actually analysing the text. By reproducing a similar structure of the ReFinEd model response, the `convertText()` method can construct the list of entities to be send back to the client.

The client, namely the Angular project, is instead stored in another repository, called *FDE-Angular*⁶⁸. This one has been deployed using GitHub Pages⁶⁹ service. The changes in the project concern mostly the disabling of the buttons that allows the upload of the text by the user (Figure 50). This is because three more buttons have been implemented to select among three texts which one to analyse, according to the mocked entity linking process created server-side. Few modifications have been made to the logic of the loading component, to send the request directly with a numeric identifier corresponding to the list of entities to be returned.

⁶⁶ <https://github.com/Flying-Digital-Editions/FDE-demo-service>

⁶⁷ <https://render.com/>

⁶⁸ <https://github.com/Flying-Digital-Editions/FDE-Angular>

⁶⁹ <https://pages.github.com/>

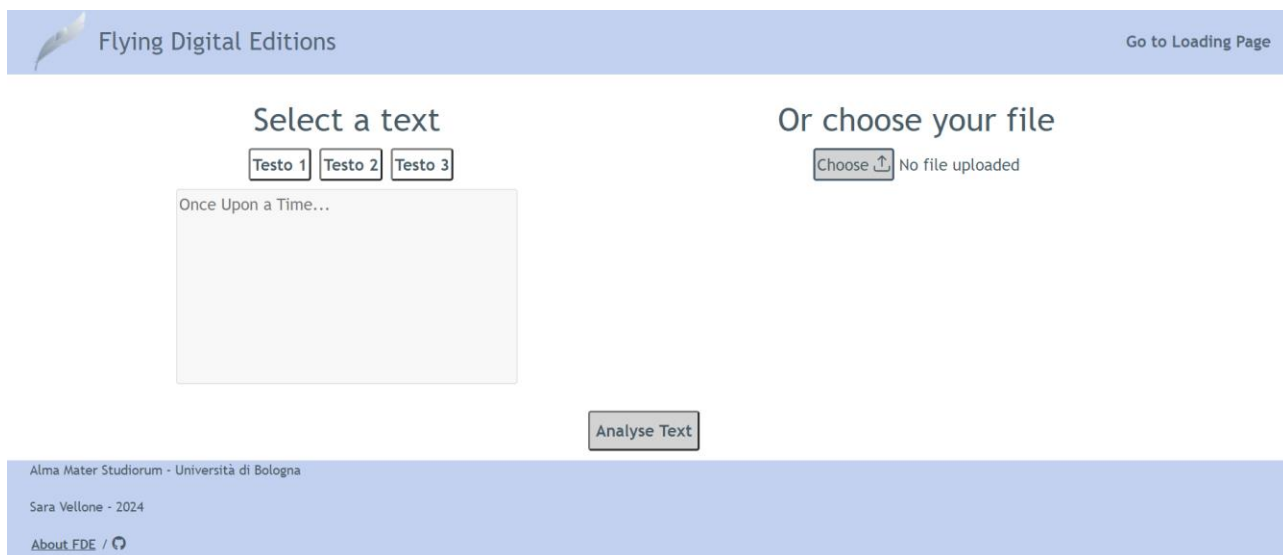


Figure 50: The loading-mode page of the mocked demo

The creation of the demo has demonstrated the high flexibility of the tool, which can adapt to different conditions without stop working. In fact, by mocking just the API response regarding the text analysis of the entities, the tool continues to be able to add, delete or modify the retrieved entities, and then to visualise the ‘final text’ and explore the entities without any kind of problem. The collecting process of the data from Wikidata is completely separated from the actual text annotation procedure, hence it still works just by providing the Wikidata identifier of the wanted item and the class defined in the preliminary categorisation.

For what concerns the three texts, they have been chosen considering the idea of providing a tool that can analyse any type of text. For this reason, the three texts are all about scientific or astronomic subject matter. In particular, the sample of the Newton’s *Principia* used during the testing of the existing tools is proposed as the first text that can be analysed.

The second text is an online article found on ‘The Teaching Astrophysicist’ website. Written by Oliver Shearman, ‘The Most Massive Stars: Elemental Engines of Our Universe’⁷⁰ is a text that about the largest known stars in our universe.

Finally, the third text is an article retrieved from the NASA archives about the moon landing of the 1969. ‘July 20, 1969: One Giant Leap for Mankind’⁷¹ summaries the incredible human mission on

⁷⁰ <https://www.theteachingastrophysicist.com/post/the-most-massive-stars-elemental-engines-of-our-universe>

⁷¹ <https://www.nasa.gov/history/july-20-1969-one-giant-leap-for-mankind/>

the Moon, in which the three astronauts took their first step -and with them, all the mankind- on our satellite.

As already explained, except for the above-mentioned alterations, the rest of the code works exactly as before, hence, the demo has been an useful tool for understanding if the idea of having all the processes included in just one application may be an interesting resource for different kind of people and for different purposes.

3. Results

This chapter will analyse various aspects of the project, by starting from the results obtained from the development of the tool, and by explaining all the features achieved. Each issue pointed out in [1.5](#) will be investigated in depth by describing the solutions adopted by the FDE tool and the outcome of these implementations. Finally, a briefly discussion on the mocked demo's feedback is here presented for giving validity to a part of this research concerning the experience of editing and visualising the annotated text. Some concrete ideas about future improvements will be analysed instead in [Chapter 5](#).

Regarding the performance of the code, the outcomes are good but not great yet. To find an entity linking annotation tool that executes a punctual and accurate text analysis is not simple. As seen in [1.4](#), there are a lot of attempts to produce a good NER process, but with a percentage similar to the evaluation scores, some entities are not correctly recognised and linked correctly.

With the tests conducted during the development of this tool, ReFinEd, with its high F1 scores (Ayoola et al. 2022), is relatively more accurate in linking the entities than in recognising them within the text. Even if it can be considered a negative aspect, it can be a strength for the purposes of this project. In fact, in the context of the project, it is overall better to have a robust entity linking tool than a NER one. As of now, for improving the ReFinEd entity detection, the text is split into chunks of a customisable number of sentences, to reduce the context for a more accurate semantic analysis. This approach tends to slow the execution, because for each block a request is sent to the backend and the analysis has some response time, but it is an effective solution in most cases. With discretely long texts and a maximum of 20 sentences for block, the execution of the analysis is estimated to be 45-55 seconds, which is a good result if compared to Falcon 2.0, that has long response times if the text is too long – with common errors in the response handling.

Another found limitation lies the fact that the types retrieved are extremely specific in some cases or too generic in others. ReFinEd, specifically, categories the entities usually as 'GPE' for countries and 'PERSON' for humans, but sometimes it is not able to correctly identify the type. To avoid errors while retrieving data, the FDE maps all the types and divides them inside bigger groups having a query template designed for retrieving specific information. It may happen that a type is not supported in the current implementation, but due to the modularity of the tool with small additions of the most used classes, it is possible to support more entity types in the knowledge panels. With four main

classes configured in the original version of the tool, it is possible to analyse different texts successfully.

To solve the problem revealed by the other tools of the lack of control over the results of the annotation process, a system for updating entities' data has been created. The management of all these operations has been implemented through a summary panel, which shows all the list of the entities retrieved from the first analysis, and by a button that allow the creation of a new entity. Near to the name, two icons allow editing and deletion of the entity.

The first action guarantees the possibility of modifying the major information that are needed to perform the queries against Wikidata, always giving the possibility to check the inputs by providing the automatic generation of a link connected to the Wikidata item, when another identifier is selected from the list of candidates' entities. The updates are immediately effective and, by clicking on the icon for modifying the item again, the panel with the new data is displayed.

The integration of the possibility of deleting and adding novel items seemed fundamental to guarantee a full experience of text analysis and exploration. Precisely because this tool is designed to be used for several purposes, having the power of enhancing manually the performance of the entity detection process is considered crucial for building a complete working instrument. Thus, more entities can be added to the recognised ones if the users provide the corresponding Wikidata ID item and the category, which can be chosen from a dropdown list. This choice was made to prevent the user from inserting classes not supported by the queries templates. Finally, the user can delete the entities not wanted by just clicking the bin icon.

All these processes take few seconds to be completed, and for each change both the text and the summary panel on the left are constantly updated.

For what concern the retrieving process of the information on Wikidata, the use of queries template ensures a coherence among the entities of a same class. In fact, to request data, all is needed is the Wikidata ID of the item and the type. While the identifier is necessary to retrieve the correct information about the item, the type is used for properly setting what must be visualised about that entity. The collecting process and the creation of the panel takes few seconds to be completed, because for each click on the desired entity a request is sent to the Wikidata SPARQL endpoint. If the user clicks on another entity, the previous panel closes automatically, guaranteeing a smooth navigation.

Differently from the Google Knowledge Panel, for some kinds of entities, the FDE panel tries to be more accurate in choosing the information to show. For example, for what concerns entities marked as 'human', the Knowledge Panel usually shows a brief description from Wikipedia and information related to the family (such as 'Parents', 'Spouse', 'Children'), which sometimes are not interesting to know. Flying Digital Editions tries to point out just the main details about the entity and provide the possibility to compare the entities inside one class because all the fields exposed are the same. For example, for what concerns the 'humans' entities, one can make a comparison between two people's occupations. This is also a consequence of the fact that Wikidata contains 11,600 properties⁷², which means that two items belonging to the same class could be described differently one from the other. By creating a set of queries, FDE tries to focus the attention just on those which could be interesting for a generic public.

A difficulty emerged when creating the panel with fields that requires the unit of measurement. In fact, it looks like the Wikidata does not automatically returns the unit of measurement of an information such as the mass of an object. Unfortunately, for this problem this project could not find a solution now. Same happens for the dates: when the item has a birth or death date year such as '300' (AC or BC), the query returns a string as '1 January 0300', which is unpleasant to display. For this reason, some formatting adjustments were made for ensuring better readability.

Another problem detected concerning Wikidata is that for some types of novels there are not included the fictional characters as separated items in the knowledge bases. By executing some tests, it has been noted that the characters of the literary work 'From the Earth to the Moon'⁷³ of Jules Verne are not available Wikidata items. On the contrary, more famous works as 'Anna Karenina'⁷⁴ of Lev Tolstoj has more considerations. In fact, the fictional character of Anna Karenina itself has a corresponding Wikidata item ID⁷⁵.

For guaranteeing a tool that can be used for any purposes, even for the more specialistic ones, the panels offer the possibility of downloading a Turtle representation of the entities. This feature allows the creation of graphs and the reuse of the retrieved information in other contexts regarding the semantic web and structured data. This addition has been made by thinking at the possibility that the creation of these kinds of editions may be useful in the field of the digital humanities, where the

⁷² https://www.wikidata.org/wiki/Wikidata:List_of_properties

⁷³ https://en.wikisource.org/wiki/From_the_Earth_to_the_Moon

⁷⁴ [https://en.wikisource.org/wiki/Anna_Karenina_\(Garnett\)](https://en.wikisource.org/wiki/Anna_Karenina_(Garnett))

⁷⁵ <https://www.wikidata.org/wiki/Q4066531>

interoperability of the data and its visualisation is one of the most important aspects. And, again, to achieve the creation of a complete tool for digital editions.

The division between the Angular project and the Python one allows developers to work with both the sides. Just by connecting to the endpoints, the APIs can be reused in other newly interfaces, and vice versa, the front-end can be adapted to other endpoints, by changing the calling methods and adapting the structures of the objects to the new data. This guarantees a maximum flexibility and creative freedom for combining different technologies and tools, as demonstrated with the demo.

An important limitation and warning for those who want to use this tool with ReFinEd as entity-linking model on premises is the need to have at least 40GB of memory to store pre-trained models. This is because this Python library downloads all the models at the first call of the method that execute the annotation of the text, using a cache folder. If wanted, the user can also indicate a different folder to store the models, but the files must be easily accessible by using a directory path. It is not possible to pass these files using an URL and preserving them somewhere else. This has been one of the key issues encountered in developing this project.

As already mentioned, one of the main critical points for what concern the entity-linking tools already existing is the fact that they do not consider non-expert users as consumers of these resources.

FDE tries to overcome this problem, by implementing an interface that is as simple as possible. It is characterised by few elements that have a consistent style to facilitate the user to understand their functionalities. In particular, the tool gives the possibility to upload both in a .txt or .doc format, which are the most used format for plain texts, but also offers a text area in which the users can paste the paragraph they want to analyse.

The editing mode requires necessarily a minimum of competence because the user needs to know or be taught about Wikidata and the use of the identifiers and types. Other than that, FDE tries to guarantee familiarity by using standard icons for identifying the possible tasks that can be performed, such as the ‘pencil’ for editing and the ‘bin’ for deleting the entity. The result is a page in which the text is at the centre as the most important feature; all the entities are underlined, so that they can be immediately noted inside the text. Already mentioned and discussed is the left panel, which is always visible on the page, while the right card – that appears only when the user is adding or modifying an entity – keeps the simplicity of the overall design, without unnecessary decorations.

For a visual continuity, the viewing mode reproduce the aspects of the previous page, by keeping the text in the centre part of the page as the main element. This time, the entities are highlighted by using a greyish colour that should invite the users to click on it. On the left side, another panel works as a filter system to search the entities. It separates the entities according to the type and list all of them associating them with an arrow icon. When clicked, the icon scrolls the text to the position where the entity is.

Hence, all these features have been implemented to provide a tool that allow users to interact with the system itself, but above all, to give something that everyone can use, for any kind of purpose.

Feedback on the mocked demo

Due to the technical issues regarding the size of the ReFinEd models, an online version of the tool is not currently available, hence, a real user testing has not been conducted in the scope of this project. But, for experiencing the idea of having a complete and comprehensive tool and to understand if this kind of resource can actually have some soundness, a demo has been created (see [2.3.1 FDE mocked demo](#)) and supplied to few people with a questionnaire of 13 questions.

The questionnaire does not follow a particular protocol, such as the Discount Usability Testing⁷⁶, because the analysis on this demo would be conditioned by the lack of real experience with regard to daily utility, namely the possibility to upload and annotate any type of text.

For this reason, the questionnaire contains more generic questions about the navigation in its entirety, and it has been useful for receiving precious feedback about the editing and visualisation part, without any type of external influences. A brief description of the project has been provided as introduction of the questionnaire and twelve people answered to it after using the demo. Following, all the questions will be listed with the corresponding responses.

Q1: What is your occupation?

The 41% of participants states they were students, while the 25% is represented by developers. The rest of the participants do other jobs.

Q2: What is your age?

The participants are equally divided into three age ranges, which are 20-25, 26-31, 31-35.

⁷⁶ Van Veenendal, Erik. 2017. 'Discount Usability Testing'. *Quality Matters*. pp. 14-17

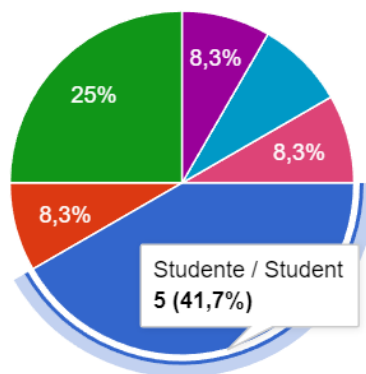


Figure 51: Q1 answers

Q3: Based on description, what do you think about this project?

All the participants chose the ‘Sounds interesting’ option.

Q4: Did you use the demo?

All the participants stated to have used the demo.

Q5: Did you find the demo intuitive and user-friendly?

The 83% of the participants answered positively to this question, while the 16,7% responded that they found it not very intuitive.

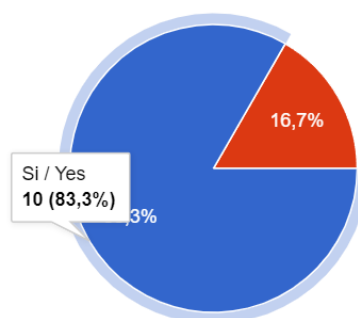


Figure 52: Q5 answers.

Q6: On the editing page, did you have trouble figuring out what to do?

The 50% of the participants stated they did not have problems in understanding what to do. The 33% answered ‘Few’. The remaining 16,6% answers both that they would like to have a ‘go back’ button for having the possibility to return on the editing page and that they did not quite understand what they could do.

Q7: Does the view page seem suitable for displaying content?

The 83,3% of the participants answered 'Yes'; an 8,3% stated that it is but can be improved and another 8,3% claimed did not seem intuitive.

Q8: Does the information panel on the view page contain interesting and useful information in your opinion?

The 91,7% of the participant answered 'Yes' to the question, while an 8,3% responded 'Yes', but with the suggestion of giving to the panel more space inside the page.

Q9: From 1 to 5 how useful do you think this tool is?

Participants were given a scale ranging from 'Indifferent' to 'Very'. The 50% of them responded 'Enough' while the other 50% answered 'Very'.

Q10: Do you think it could be useful in the educational field?

The 100% of the participants answered 'Yes'.

Q11: Select who could use it:

This question proposed to the participants a multi-option checkbox. Figure 46 shows the results, with 100% agreements on the usefulness of this tool for 'Teacher'. In second place were 'Student', while in third the 'Personal use'.

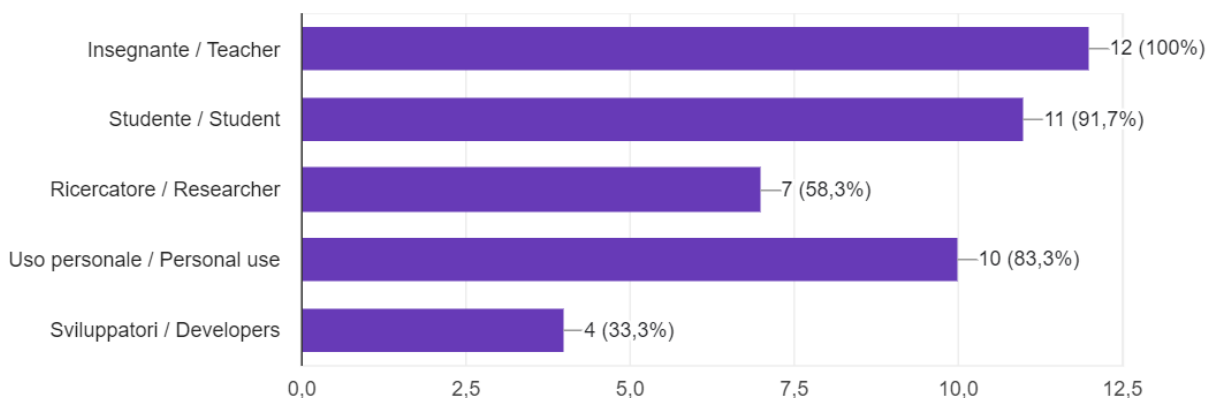


Figure 52: Q11 answers

Q12: If the official version came out, would you use it?

The 100% of the participant responded ‘Yes’.

Q13: Suggestions.

In this section, participants were left free to express their opinions and leave useful suggestions for the development of the project. In particular, a participant commented that he would like to have a tool guide available, a sort of user manual explaining the potential of the tool and how to use it, especially for the editing part, in which the user has to perform some actions.

On the same idea seems to be another participant, which suggest not only using some tooltip for guiding the user over the different functionalities, but also an additional page for previewing the changes before the view-page and as already proposed, a button for returning on the editing page.

Independent of the questionnaire is some feedback received from people who used the demo without being asked specific questions about it. The responses were quite in line with what the survey participants said, getting just a criticism regarding the choice of the tool’s name, as they expected the texts to be more historical than scientific.

As already pointed out, this section does represent a real user testing phase, but it can be considered as a part of the experiment procedure to realise whether the project approached its goals– namely, to have a complete tool for creating digital editions on the fly which can be useful for also non-expert users – by focusing on the features that the other entity-linking tools do not have. The results are, anyway, good if considered that the participants did not have access to the option of uploading a wanted text, and, consequently, they were not able to test the efficiency of ReFinEd.

Some of the suggestions received will be discussed in the next chapter that concerns the future developments.

4. Conclusions

This project started with three questions: ‘Can a Digital Edition be produced from a text automatically? At what expense and in which contexts is it possible to do so? Is the output of this process useful?’ and at the end of the first implementation of this tool, the response could be:

‘Yes, but...’.

Yes, it is possible to produce a Digital Editions from a plain text, and yes it can be done also in a short amount of time, but there are some considerations that one must make.

For this reason, the following paragraphs will provide a summary of each feature that this project was able to achieve, and a brief description of the problems encountered.

1. The implementation of a Knowledge Base

After the comparison between the three main knowledge bases freely accessible (Yago, DBpedia and Wikidata), the choice fell on Wikidata. The incredible number of data contained in this base and the reliability of its information, make it one of the best candidates for the purposes of this project. Unfortunately, even if it is the knowledge base with the most items currently available, it is fallible, and some items cannot be found, e.g. the fictional characters for certain novels. This problem can lead to less customisation because if the item is not contained in Wikidata, or the entity cannot be linked, thus the text could be not perfectly annotated. Nevertheless, the community effort to guarantee constantly updates gives the hope that it will slowly cover those items that it has not so far inserted. Another problem is the soft standardisation of entity information in Wikidata: two different entities of the same type can have a vastly different amount of information.

2. The text annotation system

For what concern the text annotation, one of the main limitations is that there is no tool or model which can perform precise annotations yet. The average of errors is still high, both for the named entity recognition and the entity linking process, hence the result of the analysis cannot be taken for sure without a check. This is why the FDE project implements not only one of the best entities linking models currently available (see [1.4.9](#)), but also adds a ‘middle layer’ to allow users to check the entities one by one. Trying to make this process as easy as possible and to reduce the time employed

in executing this task, it provides a direct link to the Wikidata item page to check the detected entity before performing any kind of change.

Another limitation is the impossibility to guarantee an instrument which is ‘context-free’, meaning that it can effectively analyse any type of text from any historical period and in any language. There are certain linguistic restrictions for texts with forms no longer used, which makes it challenging to work with this tool. For this reason, one of the ‘sacrifices’ that needs to be made is to have a more generic annotation.

3. The navigability of the text

‘Navigability’ refers to the ability to explore a text by using tools that help users to move towards the digital content for finding specific information and for having access to the desired resources. For implementing this aspect, Flying Digital Editions employs the so-called *facets* (2.3) which are a particular type of filters for searching inside a text. In fact, in the ‘view-mode’ of the tool, the users find a panel that represents an overview of the annotation process’ results, because it lists all the entities by categorising them according to their classification (i.e., their type) and allow users to easily find each entity by scrolling the text to its position.

4. The visualisation of the data

The visualisation is the final and probably the most important part in the creation of a digital edition. How to display data and information is a task constantly discussed, because from it depends also the target to which a tool is directed. FDE has chosen to display information in a way as simple as possible, by selecting the facts on which the general public is usually more interested and by constructing on them the queries against Wikidata. Once retrieved, the data are organised in a panel which maintain the same structure for all the entities belonging to one class. The main issue about this phase was to create a structure that is generic but precise at the same time, choosing from the many properties of Wikidata only those suitable for the users of this tool.

In the development of this tool, we sacrificed the creation of the critical apparatus of the text, for two reasons:

1. This type of feature is mostly used for people who are inside the field of the digital humanities, hence it would have been too specialised, and it would have given more complexity.

2. The main goal was to create a tool complete as much as possible, so we have temporarily left out this type of annotation to concentrate on entity detection and linking.

Despite the problems encountered, Flying Digital Editions is a project that can surely offer a whole experience for those who deal with texts. The simple and easy interface, in addition with the near absence of skills required, make it a powerful tool for different purposes, that is not just limited to the mere text annotation. FDE provides a more comfortable exploring approach, while enhancing the whole reading experience by catching the attention of the users and giving them the possibility to not be ‘passive’ but interactive. The entire life of a digital edition is conveniently manageable from a single application, accessible both for the public and the field’s specialists.

For instance, the educational uses are endless: teachers could use this tool to prepare a lecture on a paper – a scientific one, because it is usually perceived as a difficult subject. They can easily upload the paper, edit the entities on which they want to focus the lesson and then provide the final page to the students, using - for example - the interactive whiteboard.

Otherwise, the students might want to use this tool to better understand texts or parts of texts given by their professors. It is easy to imagine the situation in which a - university, secondary, whatever it is - student tries to figure out what the assigned text is communicating. Sometimes, the papers contain references to figures that one does not know well or never had heard, and having a tool that retrieves this data can help to improve the performance of the student, reduce the search time, and provide punctual information that can be reused.

Also, researchers and writers can benefit from the creation of such a tool. They could be interested in providing their own texts with a certain grade of knowledge depth, by allowing the tool to analyse the document and link the entities. The huge data information that Wikidata contains allow FDE to be adequate for any type of text analysis.

Flying Digital Editions is not less useful for expert users. The high flexibility and its development in separate modules, makes it easy to adapt to any needs. It can also be seen as a challenging field on which a programmer can unleash his creativity and expertise, for example by training the model already implemented for more accurate results in the text annotation or by changing the management of the lists of entities in the various parts of the tool. With the appropriate developments, the entire view-mode can be exported and framed in other pages. Some publishing houses or journal editors

might provide their texts making the inner knowledge explicit and avoiding manually linking all the main references.

Digital Humanists could be helped by the employment of this tool too. Having all the different tools that they usually use in their work would lead to improved performance and reduced working time.

As already discussed, it was not possible to conduct a user testing phase, although the tool is fully functional locally.

Despite this, the results of the demo feedback show that all the participants see the potential of this tool for educational purposes, which means that the target group of users was correctly identified. Some adjustments must be made to make it more efficient and user-friendly, as pointed out in the previous chapter, but the 83.8% of participants claimed they would use it also for personal use. This means that the tool can be helpful not only for particular tasks – i.e., a school homework – but also for everyday business, such as trying to better understand a journal article on the web on a matter in which we don't have expertise. The feedback shows an interest in the complete project if ever made available and the participants found the information retrieved and displayed as interesting and useful, thereby confirming the enhanced experience with the tool.

In conclusion, Flying Digital Editions aims to be simple and complete aid directed to the public for experiencing and managing the whole life cycle of a digital edition. From the plain text to its visualisation, this tool includes all the technologies currently available to produce a single and comprehensive output. To all intents and purposes, it is designed to give people the opportunity to fly through the text, creating digital editions that are immediately available and simple to handle, and to discover all the wonders still hidden beyond the words.

5. Future Works

As with any project, improvements are possible. The architecture of FDE is open and more tools are easily implementable as add-ons to manage the whole life cycle of a (simple) digital edition.

Some of the following improvements include:

1. Tools/functions/features that are common in SDE⁷⁷, but have not been implemented at the current state of the project.

The integration of annotation and critical apparatus could significantly enhance the value of FDE as a comprehensive tool, both for the public and the Digital Humanities community. Currently, there are tools that can help automate the process of annotating critical texts, but they are often very generic and rely mostly on regular expressions (which are language dependent), identification of the start and end of a page, a paragraph, or a direct dialogue. Using Natural Language Processing models and pipelines could help further annotate the text with more information, especially semantic information (such as indirect dialogue, which is not easily automatically annotated through regular expressions or rules). A proposed development that was already thought of but could not be fully implemented was the automatic annotation of text using TEI-XML⁷⁸ with the headers (with metadata) of the documents and identifiers of the detected entities during document export. Importing text directly in TEI-XML is also an easily implementable add-on. This feature, coupled with the possibility to download a Turtle file containing all the graphs of the entities and the metadata of the document, facilitates the construction a full graph of the entire text. This improvement would promote data exchange and the reuse of annotated data in different applications and research contexts. By enabling the creation of a full graph, FDE can enhance the reusability of the information inside the text and can be expanded further with more models to extract more information.

⁷⁷ Scholarly Digital Editions

⁷⁸ <https://tei-c.org/release/doc/tei-p5-doc/en/html/SG.html>

2. Tools/functions/features that have been suggested from the demo feedback.

The demo feedback revealed a need for a more streamlined tool for people not interested in the editing part. For this reason, an authentication system could be implemented to distinguish and separate the edit-mode from the view-mode. By additionally providing a database to store the uploaded text, a teacher or an editor can upload a text, edit the entities, and save their works for later viewing; hence, the students or the generic users, by inserting their credentials, may access the view-mode and explore independently the text without getting involved into the underlying logic. In this way, the tool could be more usable and easier for end-users that are not at all interested in the complete experience.

Instead, for those who need the edit-page, a detailed guide could be provided for exactly understanding what to do and how to do it. A third option as ‘temporary user’ could provide a service of instant analysis of a text for those who are not interested in a continuous use of the tool but want to exploit it occasionally. In this case, the edit-mode is not needed and the user inserting the text can be directly redirected to the view page.

To enhance the structure of the knowledge panes, the implementation of dynamic templates could be a compelling feature addition. This functionality aims to provide users with further information about recognised entities by generating templates based on the most important entity’s property contained in Wikidata. The dynamic nature of this approach allows the tool to always be able to return information about recognised entities and decreases the likelihood of having panels with sparse or insufficient data to display. This improvement would make the design of the panels tailored to the entity, and consequently the user would have more accurate data to rely on. Moreover, as new properties and information become available on Wikidata, the system can automatically update the templates (hence, not just the data) to reflect these changes, ensuring constant updates and current information.

3. Tools/functions/features that are implementable using available models (especially to improve NER, Entity Linking, Relationship extraction, distant reading techniques)

Given larger datasets and tools, fine-tuning the ReFineD model with more data can be a first solution to improve the EL part of the model, to not only identify named entities but also classes, unnamed entities of interest and so on. To provide more interesting semantic structured information to the user, a *relation extraction* model could be implemented to find relations inside the text itself. *Relationship*

extraction consists in discerning the relationships that exist among the detected entities (Martin and Jurafsky 2024, 416), thereby providing additional information that may or not be explicit. This would help in generating not only richer knowledge graphs, but also be used to visualise the relationships between entities in a text, adding another automatic tool that, as of now, does not exist in any published Scholarly Digital Edition. This is already implementable using models such as LLama 2 or GPT-3.5 through the spacy-llm library⁷⁹ but would require dependency to a Large Language Model, solution that poses many additional questions that are, as of now, out of the scope of this work.

Furthermore, exploring the integration of other models capable of recognising lemma definitions or phrases, presents an opportunity to expand the functionalities already developed. Also executing a multiple entity-linking process and performing an accurate annotation of texts in different languages could be compelling additions for improving the annotation process.

The incorporations of these advanced features, which are highly specific and not for any type of users, ensures that it remains an adaptable tool to diverse research needs and user preferences.

In the perspective of exploiting the use of some distance reading techniques⁸⁰, the integration of a dedicated page for visualising all the extracted entities and their relationship can be a feature enhancement that allows to perform several analyses, including quantitative analysis and topic modelling⁸¹, by examining the occurrences of specific entities and their relationships within the text. These inclusions offer several benefits, because it provides users with a comprehensive overview of the entities and their interconnections, facilitating the analysis and exploration of the text. For instance, a new modal panel gives users the possibility to visually inspect the most common term within the text, to identify patterns or calculate the frequency of relationships between entities.

Moreover, for better grouping the entities, a potential future improvement could involve highlighting them with different colours based on their classification. For example, all the entities classified as ‘PERSON’ will be marked in yellow, with entities marked as ‘LOCATION’ could be highlighted in green. This visual differentiation enhances the interpretability of the entities on the page, by enabling users to quickly identify and choose the entities in which they are interested. Facets, which are common in SDE, can also be added to navigate through highlighted mentioned in the text.

⁷⁹ <https://spacy.io/api/large-language-models>

⁸⁰ https://en.wikipedia.org/wiki/Distant_reading

⁸¹ <https://www.datacamp.com/tutorial/what-is-topic-modeling>

The positive response received from the demo feedback demonstrates that such a tool could be powerful if fully implemented, validating the idea that stands behind this project. Despite the challenges encountered, such the issues regarding the entity linking process, the concepts remain robust to for further refinement and development. The tool's innovative approach to text exploration, facilitated by the creation of information panels, makes the knowledge acquisition a fast and straightforward process, without neither requiring the users to leave the page in which they are nor obliging them to make use of external resources and platforms. The versatility of FDE extends beyond the academic contexts, with potential application daily life as well as in education and research fields, hence confirming its being an affordable tool for everyone. The possibility to improve and add new features to the project while maintaining the core stable is one of its main strengths, because it means can be adapted to any need.

The ambition of Flying Digital Editions (FDE) may prompt discourse within the scholarly community due to its forward-looking propositions. Nonetheless, FDE epitomizes the potential evolution of Digital Editions (DEs) into a harmonious collaboration between artificial intelligence and domain experts in the foreseeable future. This iteration of DE, while potentially broader and less nuanced than editions crafted entirely by human effort, heralds a new era for the democratization of digital project creation, particularly for individuals or entities constrained by limited resources. Such DEs offer an accessible entry point for students and broader audiences, fostering enhanced collaboration between academic circles and the public to foster the dissemination of meticulously curated and reliable information online. In conclusion, FDE represents a significant step towards bridging the gap between advanced digital technologies and traditional scholarly practices. It promises a future where the creation and curation of digital editions are more inclusive, collaborative, and aligned with the principles of open knowledge and accessibility.

TABLE OF FIGURES

FIGURE 1: EXAMPLE OF WIKIPEDIA DATA MODEL, PROPERTY-VALUE PAIRS.	12
FIGURE 2: AN EXAMPLE OF A GOOGLE KNOWLEDGE PANEL FOR THE ENTITY "GALILEO GALILEI". ...	14
FIGURE 3: EXAMPLE OF API REQUEST FOR THE TITLE ‘JOHN BAUER’ WITH ACTION WBGETENTITIES.	16
FIGURE 4: EXAMPLE OF RESPONSE AFTER API REQUEST FOR THE TITLE ‘JOHN BAUER’ WITH ACTION WBGETENTITIES.	17
FIGURE 5: EXAMPLE OF API REQUEST FOR THE TITLE ‘JOHN BAUER’ WITH ACTION QUERY.....	18
FIGURE 6: EXAMPLE OF RESPONSE AFTER API REQUEST FOR THE TITLE ‘JOHN BAUER’ WITH ACTION QUERY	18
FIGURE 7: EXAMPLE OF TEXT ANNOTATION WITH STANZA	20
FIGURE 8: EXAMPLE OF WIKIFIER’S ANALYSIS RESULT ON A TEXT.	22
FIGURE 9: PARTS OF SPEECH ANNOTATION ON A TEXT WITH WIFIER TOOL.	22
FIGURE 10: EXAMPLE OF TEXT ANNOTATION WITH OPENTAPIOCA.	24
FIGURE 11: EXAMPLE OF ENTITY LINKING WITH FALCON 2.0.....	27
FIGURE 12: EXAMPLE OF TEXT ANNOTATION WITH TAGME.....	29
FIGURE 13: EXAMPLE OF TEXT ANNOTATION USING DBPEDIA SPOTLIGHT.....	31
FIGURE 14: EXAMPLE OF TEXT ANNOTATION USING ENTITY-FISHING.....	33
FIGURE 15: EXAMPLE OF ReFinEd ANNOTATION PROCESS RESULTS.	35
FIGURE 16: EXAMPLE OF COMPLETE RESPONSE OF ReFinEd ANNOTATION PROCESS.	35
FIGURE 17: EXAMPLE 1 OF A SPARQL SELECT QUERY FROM WIKIDATA TUTORIAL.	44
FIGURE 18: EXAMPLE 2 OF A SPARQL SELECT QUERY FROM WIKIDATA TUTORIAL.	45
FIGURE 19:SERVICE CLAUSE FROM WIKIDATA TUTORIAL.	45
FIGURE 20: EXAMPLE OF A TURTLE DOCUMENT DESCRIBING THE RELATIONSHIP BETWEEN GREEN GOBLIN AND SPIDERMAN PROVIDED BY W3C DOCUMENTATION	46
FIGURE 21: ENDPOINT ‘/GETENTITIES’ RESPONDING TO POST REQUESTS.....	47
FIGURE 22: ENDPOINT ‘/TYPE/ID’ RESPONDING TO GET REQUESTS.	48
FIGURE 23: CONVERTTEXT() METHOD FOR ENTITY RECOGNITION AND LINKING WITH ReFinEd.	49
FIGURE 24: GETPERSONINFO() METHOD.....	50
FIGURE 25: SPACEOBJQUERY() FUNCTION QUERY FOR RETRIEVING DATA FROM WIKIDATA.	53
FIGURE 26: SENDTEXT() METHOD	55
FIGURE 27: FORMATTEXT() METHOD.	56
FIGURE 28: HANDLESPANCLICK() METHOD.	57
FIGURE 29: DELETEENTITY() METHOD.	58

FIGURE 30: DETAIL OF THE FUNCTION <code>FORMATTEXT()</code>	59
FIGURE 31: <code>goToPos()</code> FUNCTION.	59
FIGURE 32: <code>GETENTITYINFORMATION()</code> AND <code>CREATEPANEL()</code> FUNCTIONS.	60
FIGURE 33: <code>GETANALYZEDTEXT()</code> FUNCTION.	62
FIGURE 34: <code>GETENTITYINFO()</code> FUNCTION.	62
FIGURE 35: LOADING PAGE OF THE DEMO.	63
FIGURE 36: LOADING SPINNER COMPONENT.	64
FIGURE 37: EDIT-MODE PAGE, AN EXAMPLE OF ENTITY EDITING.	65
FIGURE 38: DETAIL OF THE TEXT AFTER THE AUTOMATIC ANNOTATION PROCESS IN THE EDIT-MODE.	65
FIGURE 39: DETAIL OF THE SUMMARY PANEL IN THE EDIT-MODE.	66
FIGURE 40: DETAIL OF THE CARD FOR EDITING THE DETECTED ENTITY IN THE EDIT-MODE.	67
FIGURE 41: EXAMPLE OF INSERTING A NEW ENTITY IN THE EDIT-MODE.	68
FIGURE 42: METADATA MODAL.	68
FIGURE 43: THE VIEW-MODE.	69
FIGURE 44: EXAMPLE OF FACETS IN THE VIEW-MODE.	70
FIGURE 45: DETAIL OF THE SAMPLE TEXT IN THE VIEW-MODE.	71
FIGURE 46: EXAMPLE OF A "PERSON" ENTITY'S INFORMATION PANEL.	71
FIGURE 47: EXAMPLE OF A "SPACE" ENTITY'S INFORMATION PANEL.	72
FIGURE 48: EXAMPLE OF "OTHER" ENTITY'S INFORMATION PANEL.	73
FIGURE 49: EXAMPLE OF DOWNLOADED TURTLE FILE OF THE ENTITY.	73
FIGURE 50: THE LOADING-MODE PAGE OF THE MOCKED DEMO	75
FIGURE 51: Q1 ANSWERS	82
FIGURE 53: Q11 ANSWERS	83

REFERENCES

Papers

Ayoola, Tom, Shubhi Tyagi, Joseph Fisher, Christos Christodoulopoulos, and Andrea Pierleoni. 2022. ‘ReFinED: An Efficient Zero-Shot-Capable Approach to End-to-End Entity Linking’. arXiv. <https://doi.org/10.48550/arXiv.2207.04108>.

Berners-Lee, Tim, Tom Heath and Christian Bizer. 2009. ‘Linked Data: The Story so Far’. *International Journal on Semantic Web and Information Systems*. 5. 1-22. <http://dx.doi.org/10.4018/jswis.2009081901>.

Brank, Janez, Gregor Leban, and Marko Grobelnik. 2017. ‘Annotating Documents with Relevant Wikipedia Concepts’. *Proceedings of the Slovenian Conference on Data Mining and Data Warehouses*. Ljubljana, Slovenia. <https://api.semanticscholar.org/CorpusID:52236149>.

Daiber, Joachim, Max Jakob, Chris Hokamp, and Pablo N. Mendes. 2013. ‘Improving Efficiency and Accuracy in Multilingual Entity Extraction’. In *Proceedings of the 9th International Conference on Semantic Systems*, 121–24. Graz Austria: ACM. <https://doi.org/10.1145/2506182.2506198>.

Delpuch, Antonin. 2020. ‘OpenTapioca: Lightweight Entity Linking for Wikidata’. arXiv. <http://arxiv.org/abs/1904.09131>.

Driscoll, Matthew James, and Elena Pierazzo. 2016. *Digital Scholarly Editing: Theories and Practices*. Open Book Publishers. <https://doi.org/10.11647/obp.0095>.

Färber, Michael, Basil Ell and Carsten Menne. ‘A Comparative Survey of DBpedia , Freebase, OpenCyc , Wikidata , and YAGO.’ (2015). <https://api.semanticscholar.org/CorpusID:34478646>

Ferragina, Paolo, and Ugo Scaiella. 2010. ‘TAGME: On-the-Fly Annotation of Short Text Fragments (by Wikipedia Entities)’. In *Proceedings of the 19th ACM International Conference on Information*

and Knowledge Management, 1625–28. CIKM '10. New York, NY, USA: Association for Computing Machinery. <https://doi.org/10.1145/1871437.1871689>.

Lehmann, Jens, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, et al. 2015. 'DBpedia – A Large-Scale, Multilingual Knowledge Base Extracted from Wikipedia'. *Semantic Web* 6 (2): 167–95.
<https://doi.org/10.3233/SW-140134>.

Lemus-Rojas, Mairelys, and Jere D. Odell. 2018. 'Creating Structured Linked Data to Generate Scholarly Profiles: A Pilot Project Using Wikidata and Scholia'. *Journal of Librarianship and Scholarly Communication* 6 (1). <https://doi.org/10.7710/2162-3309.2272>.

Lopez, Patrice. 2022. 'Entity-Fishing - Entity Recognition and Disambiguation — Entity-Fishing 0.0.6 Documentation'. <https://nerd.readthedocs.io/en/latest/index.html>.

Martinelli, Luca. 2016. 'Wikidata: la soluzione wikimediana ai linked open data'. *AIB studi* 56 (1), 75-85. <https://doi.org/10.2426/aibstudi-11434>.

Qi, Peng, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. 2020. 'Stanza: A Python Natural Language Processing Toolkit for Many Human Languages'. arXiv.
<https://doi.org/10.48550/arXiv.2003.07082>.

Sakor, Ahmad, Kuldeep Singh, Anery Patel, and Maria-Esther Vidal. 2020. 'Falcon 2.0: An Entity and Relation Linking Tool over Wikidata'. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 3141–48. Virtual Event Ireland: ACM.
<https://doi.org/10.1145/3340531.3412777>.

Suchanek, Fabian, Mehwish Alam, Thomas Bonald, Pierre-Henri Paris, and Jules Soria. 2023. 'Integrating the Wikidata Taxonomy into YAGO'. arXiv.
<https://doi.org/10.48550/arXiv.2308.11884>.

Vrandečić, Denny, and Markus Krötzsch. 2014. 'Wikidata: A Free Collaborative Knowledgebase'. *Communications of the ACM* 57 (10): 78–85. <https://doi.org/10.1145/2629489>.

Sitography

‘API Sandbox’. Wikidata. Accessed 20 September 2023.

<https://www.wikidata.org/wiki/Special:ApiSandbox> .

‘Angular (Web Framework)’. 2024. Wikipedia. Accessed 12 November 2023.

[https://en.wikipedia.org/w/index.php?title=Angular_\(web_framework\)&oldid=1194326623](https://en.wikipedia.org/w/index.php?title=Angular_(web_framework)&oldid=1194326623).

Google. ‘About Knowledge Panels’. Accessed 20 September 2023.

<https://support.google.com/knowledgepanel/answer/9163198?hl=en#>.

Cohen, Diana. 2014. ‘6 Ways Color Psychology Can Be Used to Design Effective E-Learning - Shift’. Accessed 6 February 2024.

<https://www.shiftelearning.com/blog/bid/348188/6-ways-color-psychology-can-be-used-to-design-effective-elearning>.

Coppola, Lucia Maria. 2023. ‘Exploring Named Entity Disambiguation’. 29 August 2023.

<https://datavid.com/blog/named-entity-disambiguation>.

‘Entity Linking’. 2021. Devopedia. Version 6, June 28. Accessed 12 November 2023.

<https://devopedia.org/entity-linking> .

‘Entity Linking’. 2023. In *Wikipedia*.

https://en.wikipedia.org/wiki/Entity_linking .

‘Edition’. Merriam-Webster.com Dictionary. Accessed 14 January 2024.

<https://www.merriam-webster.com/dictionary/edition>.

‘Flask (web framework)’. Wikipedia. Accessed 28 January 2024.

[https://en.wikipedia.org/wiki/Flask_\(web_framework\)](https://en.wikipedia.org/wiki/Flask_(web_framework)) .

‘Getting Started | Yago Project’. 2024. Accessed 13 February 2024.

<https://yago-knowledge.org/getting-started>.

‘Knowledge Base’.2023. In Wikipedia. Accessed 14 January 2024.

https://en.wikipedia.org/w/index.php?title=Knowledge_base&oldid=1189427043.

Martin, James H., and Dan Jurafsky. 2024. ‘Speech and Language Processing’. Accessed 13 February 2024. <https://web.stanford.edu/~jurafsky/slp3/>.

Mancini, Gennaro. 2023. ‘Google Knowledge panel: cos’è, come si attiva e a cosa serve’. *SEOZoom* (blog). Accessed 22 September 2023.

<https://www.seozoom.it/google-knowledge-panel/>.

‘MediaWiki’. MediaWiki. Accessed 23 January 2024.

<https://www.mediawiki.org/wiki/MediaWiki>.

‘MediaWiki API Help - Wikidata’.Wikidata. Accessed 23 January 2024.

<https://www.wikidata.org/w/api.php>.

‘PrimeNG’. PrimeFaces. Accessed 29 January 2024.

<https://www.primefaces.org/primeng-v8-lts/#/>.

‘Python (Programming Language)’. 2024. Wikipedia. Accessed 28 January 2024.

[https://en.wikipedia.org/wiki/Python_\(programming_language\)](https://en.wikipedia.org/wiki/Python_(programming_language)) .

‘RDF - Semantic Web Standards’. W3C Semantic Web. Accessed 22 January 2024.

<https://www.w3.org/RDF/>.

Sanjay, Vaishali. 2023. ‘The Importance of Google Knowledge Panels: Benefits and Guidance’. LinkedIn. Accessed 20 September 2023.

<https://www.linkedin.com/pulse/importance-google-knowledge-panels-benefits-guidance-vaishali-sanjay/>.

‘SPARQL’. 2024. In *Wikipedia*. Accessed 29 January 2024.

<https://en.wikipedia.org/wiki/SPARQL> .

‘SPARQL 1.1 Query Language’. W3C. Accessed 29 January 2024.

<https://www.w3.org/TR/sparql11-query/> .

‘TypeScript’. 2024. Wikipedia. Accessed 8 January 2024.

<https://en.wikipedia.org/w/index.php?title=TypeScript&oldid=1194795281>.

‘Turtle (syntax)’.2023. Wikipedia. Accessed 30 January 2024.

[https://en.wikipedia.org/wiki/Turtle_\(syntax\)](https://en.wikipedia.org/wiki/Turtle_(syntax))

Vellone, Sara. 2023. ‘FDE – Flying Digital Editions’.

<https://flying-digital-editions.github.io/FDE-Angular/>

‘What Is Google Knowledge Panel?’ . Ahrefs. Accessed 23 January 2024.

<https://ahrefs.com/seo/glossary/google-knowledge-panel>.

‘Wikidata’. 2024. Wikipedia. Accessed 22 December 2023.

<https://en.wikipedia.org/w/index.php?title=Wikidata&oldid=1195024865>.

‘Wikidata:Introduction’. Wikidata. Accessed 14 January 2024.

<https://www.wikidata.org/wiki/Wikidata:Introduction>.

Repositories

‘Amazon-Science/ReFinED’. (2022) 2024. Python. Amazon Science.

<https://github.com/amazon-science/ReFinED>.

Brank, Janez and Artificial Intelligence Laboratory, Jožef Stefan Institute. ‘Wikifier - Semantic annotation service for 100 languages’ (2017).

<https://wikifier.org/>

Lopez, Patrice. (2016) 2024. ‘entity-fishing’. Java.

<https://github.com/kermitt2/entity-fishing>.

‘Opentapioca’. (2018) 2024. ‘Opentapioca’. Python.

<https://github.com/opentapioca/opentapioca>.

‘SDM-TIB/falcon 2.0’.2020. Scientific Data Management Group. Python.

<https://github.com/SDM-TIB/falcon2.0> .

‘Stanfordnlp/stanza’. 2020. Stanford NLP Group. Python.

<https://github.com/stanfordnlp/stanza> .

‘TagMe’. 2012. D4Science. Python.

<https://sobigdata.d4science.org/web/tagme/tagme-help> .

Vellone, Sara. 2023. ‘FDE-project’. Python/Typescript.

<https://github.com/SaraVell1/FDE-project> .

Vellone, Sara. 2023. ‘FDE-demo-service’. Python.

<https://github.com/Flying-Digital-Editions/FDE-demo-service> .

Vellone, Sara. 2023. ‘FDE-Angular’. Typescript.

<https://github.com/Flying-Digital-Editions/FDE-Angular> .