Practical Work - Project Report

SemUN: A Semantics-Powered Search Platform for the United Nations' Digital Library

Clément Sicard
csicard@ethz.ch
D-INFK, ETH Zurich

Supervised by:
Dr. Menna El-Assady[1], Dr. Sascha Langenbach[1], Catherine Pysden, MSc.[2]

[1]ETH Zurich [2]United Nations

August 25, 2023

A faire

Keywords:
Natural Language Processing (NLP), Named Entity Recognition (NER), Graph databases, Frontend, Network Visualization

1 Introduction

The United Nations Digital Library (UNDL) is a United Nations (UN) service that provides public access to a diverse range of UN documents: voting data, speeches, maps, and open access publications starting from 1979. All of these documents have been classified according to the UNBIS Thesaurus, a multilingual database of the controlled vocabulary used to describe UN documents and other materials in the Library's collection, with a more or less precise topic label.

The main idea of this project is to create an analysis platform, with a network visualization of documents from a subset of the documents in the UN Digital Library. The analytics platform includes a basic Named Entity Recognition (NER) system to extract mentioned entities from the documents. The visualization part will be a network visualization, leveraging the versatility of the structure of a graph to display the extracted insights. Both parts aim at improving the search of documents by implementing an analytics layer on top of the existing search engine from the digital library.

2 Motivation & Scope

In the short-term, the goal of this project is to provide an MVP of a potential future UN product, in close contact with UN staff to make it conform to their needs. Specifically, this project will focus – as a first iteration – on Catherine Pysden's suggestion, "Women in Peacekeeping".

The project's long-term scope is to be used as a search engine for UN staff, member-state delegates, and members of the public with an interest in UN topics. Potential future work could include extending the project to the whole UN Digital Library. It could, for instance, also suggest Thesaurus-compliant metadata for untagged documents to facilitate the work of Library staff.

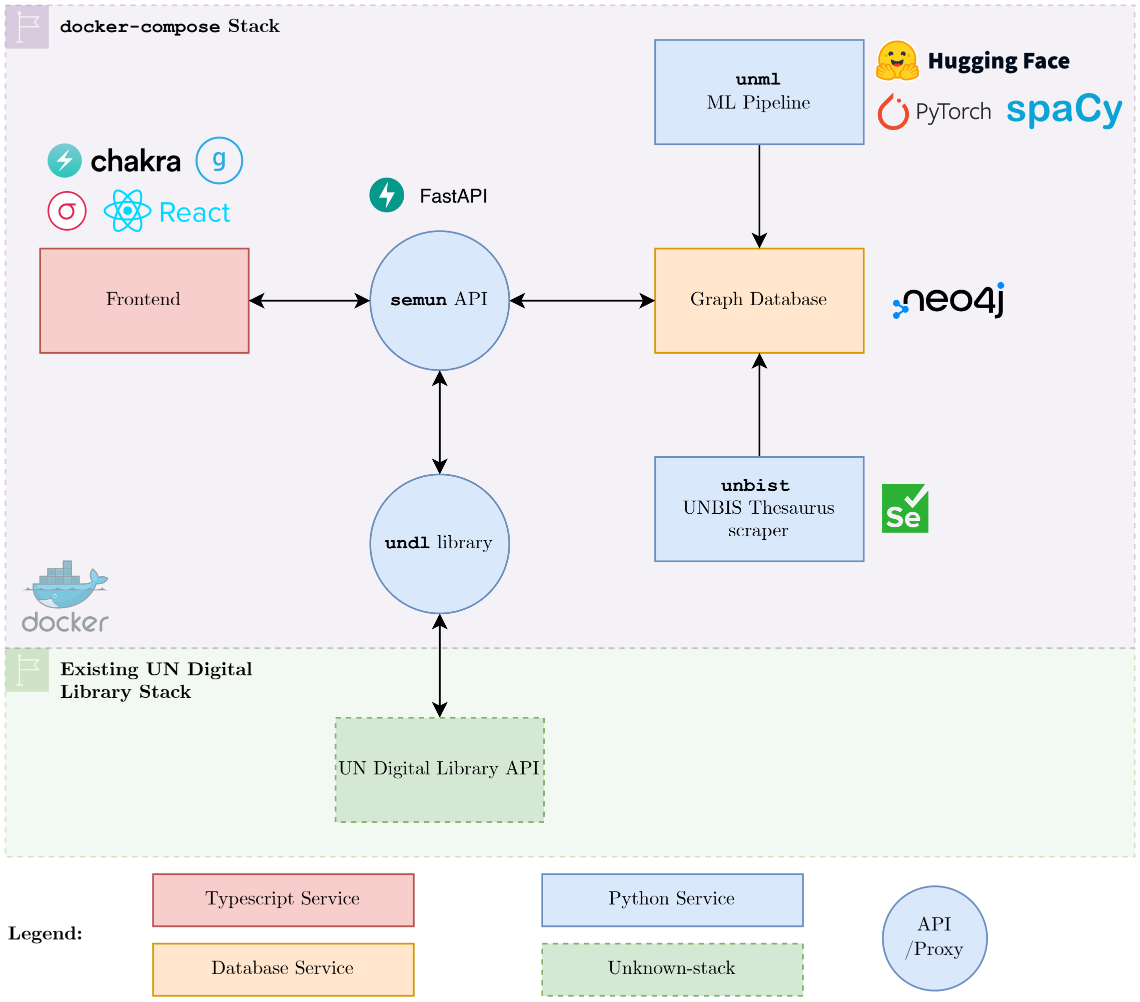


Figure 1: Final stack architecture

3 Final architecture

The final architecture is a full-stack architecture, from the database to the frontend, handed in as a Docker compose stack: `un-semun`.

The project consists of 7 GitHub repos:

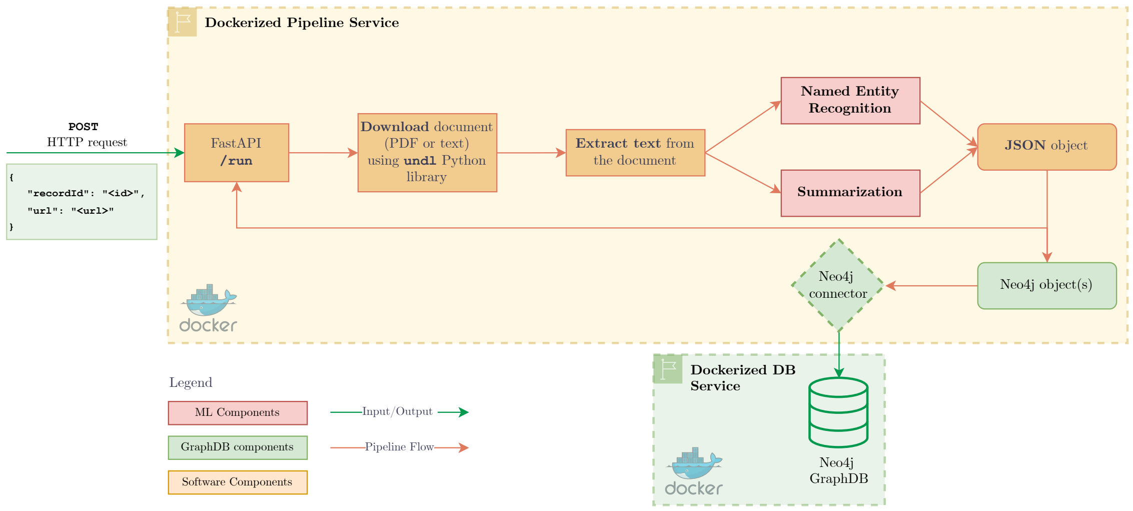
- `un-semun`: The main repository with the `docker-compose` stack declaration.
- `un-semun-frontend`: The frontend.
- `un-semun-api`: The API for the frontend.
- `undl`: The code for `undl`, a Python wrapper around the UN Digital Library API.

- un-unbis-thesaurus-scraper: A scraper for the UNBIS Thesaurus taxonomy website.
- un-ml-pipeline: Machine learning pipeline for UNDL documents.
- un-semun-misc: Diverse scripts used for the project.
- un-semun-paper: The code for this paper.

This paper will go through each of them in detail except for the paper repository.

3.1 `un-semun-frontend`: A React & Sigma.js frontend

I used React combined with Typescript for the UI framework, as well as Chakra UI for the UI components and `Sigma.js` via its React adapter `@react-sigma` for the network map. `graphology` was also used for graph manipulation in the frontend, mostly to iterate over graph elements to perform styling. The code is available here: `un-semun-frontend`.



```
{
    "recordId": "<id>",
    "url": "<url>"
}
```

Figure 2: `un-semun-frontend` library: a dockerized machine learning pipeline for NER & summarization

The frontend was the part I was the least familiar with, but Chakra UI allowed to insert nice-looking components that I could customize based on my use case. It is composed in two panes:

- **The search bar** (on top): the user can enter its prompt, which will be sent to the API (3.2) to retrieve the results.
- **The result list** (on the left). The results are displayed as a scrollable list of `Card` components, with the title, the summary, and the date of publication. The user can click on a card to display the document in the right pane.
- **The network map** (on the right). The results of the search are also displayed as a network map, with the documents, related United Nations bodies, topics from UNBIS Thesaurus taxonomy, and named entities extracted from the documents. The user can click on a node to display the document in the left pane. This map is also fetched using the API (3.2) and is based on the results of the machine learning pipeline (3.5).

3.2 `un-semun-api`: An API for `un-semun-frontend` using `FastAPI`

The `un-semun-api` Python package was developed to provide an API for the frontend. It is a `FastAPI` application, which is a Python framework for building APIs. It is a modern framework, which is fast, easy to use, and well documented. One nice feature is that it is also coupled with `pydantic` to perform data validation and serialization. This is very useful to ensure that the data sent to the frontend is valid, and to avoid having to write boilerplate code for serialization and deserialization.

This package mainly acts as a proxy between the frontend and the UNDL API, and is composed of 2 main endpoints:

- `/search`: This endpoint is used to perform a search query. It takes a `GET` HTTP request with query parameter the prompt. Then, it uses `undl` (3.3) library to perform the search on the UNDL API and returns the results to the frontend.
- `/graph`: This endpoint is used to get the graph corresponding to the above search query, also with an HTTP `GET` request with the prompt as unique query parameter. To optimize the process, it first gets the IDs of all documents returned by the search query on the given prompt, then it queries the Neo4j graph database (3.6) to get the graph corresponding to these documents. Finally, it returns the graph to the frontend as `JSON` structured according to the format expected by the `graphology` Typescript library, which is used to model a graph and manipulate it as a programmatic object.

Note that the API is dockerized and that it offers a basic query caching mechanism to avoid querying the UNDL API too often. This is done using a simple `dict` in memory, which is not ideal but enough for the scope of this project.

3.3 `undl`: A Python library to wrap to the UN Digital Library API

Tech stack of `undl`

For the `undl` library, I used Python 3.10 with packages, with `requests` for the HTTP requests, `pandas` for the data manipulation, and `pydantic` for the data validation.

Intermediary between frontend and Neo4j

Proxying the UNDL API using `undl` library

3.4 `un-unbis-thesaurus-scraper`: the UNBIS Thesaurus scraper

Tech stack of `un-unbis-thesaurus-scraper`

3.5 `unml`: The machine learning pipeline

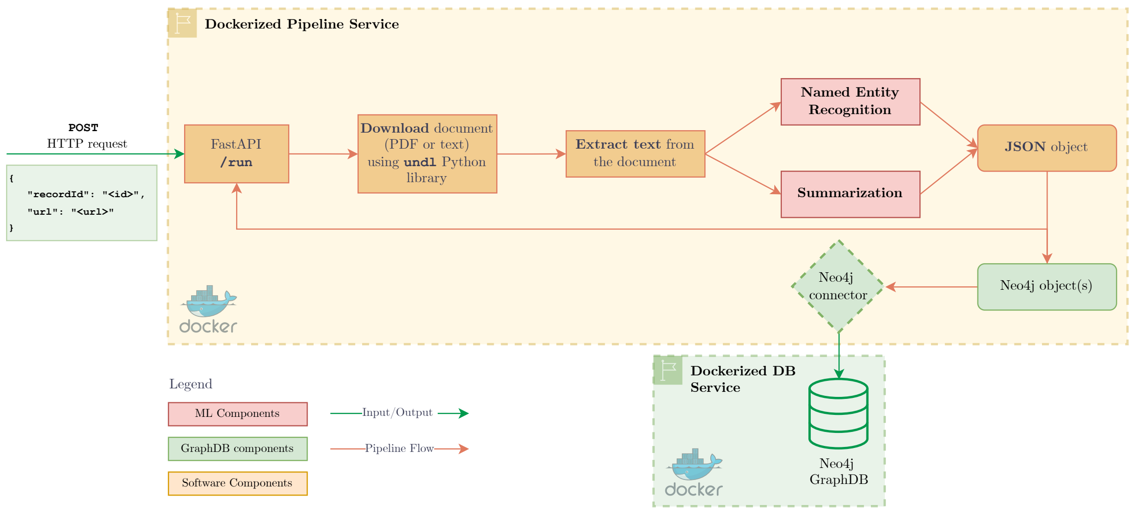
Tech stack of `un-ml-pipeline`

3.5.1 Parts

Parts

3.5.2 Models

Models



```
{
    "recordId": "<id>",
    "url": "<url>"
}
```

Figure 3: `unml` library: a dockerized machine learning pipeline for NER & summarization

3.5.3 API

API

Connection to Neo4j

3.6 Neo4j graph database

3.6.1 Types of nodes

Types of nodes

3.6.2 Types of relationships

Types of relationships

3.7 `un-semun`: The main repository

`un-semun` is a repository that englobes all other repositories as submodules. It also contains the `docker-compose` stack declaration, which points to Dockerfiles in submodules. They are updated using the `Makefile` when new commits are added to the submodules. The port forwardings and environment variables are also declared here. This is the main entry point to run the whole stack.

3.8 `un-semun-misc`

This repository contains scripts for the both the UN member states and the UN bodies: it first extracts the data from an Excel file and a CSV file respectively, then it cleans the data, and finally it inserts it into the Neo4j database after having written a Cypher query. The scripts are written in Python and directly use the `neo4j` Python connector library.

3.9 General tech stack notes

For all the Python components, the dependencies are managed using `poetry`. They are also all dockerized, and the whole stack is orchestrated using `docker-compose`.

4 Limitations

5 Discussion & future work

Conclusion