Metadata Extraction

Members: Abhishek Nadgeri, Salmaan Tariq, Omar Ejjeh, Lei Wang, Amit Mudgal, David Abdelsalam

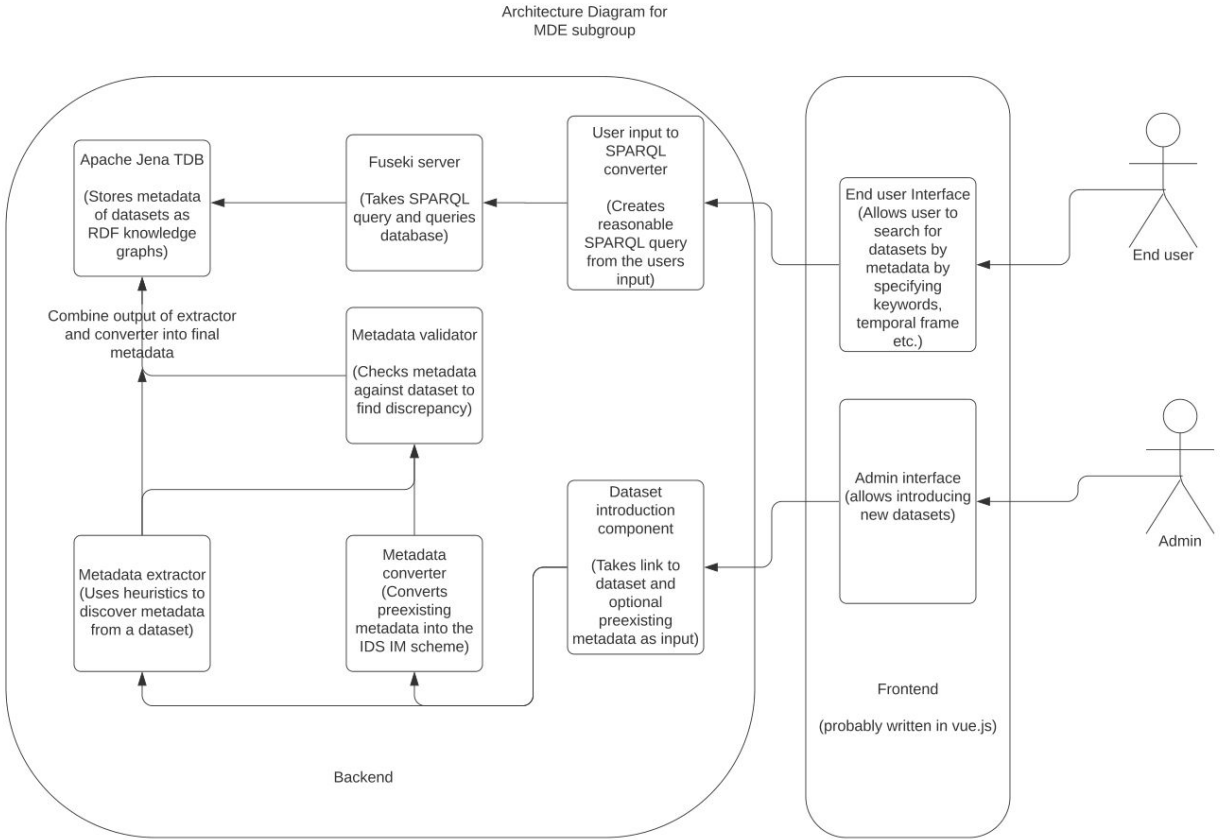
Proof of Concept Presentation, June 2021

Overview

- Recap of the Architecture
- Metadata Extraction
  - Keyword extraction
  - Data Quality
  - Language Extraction
- Convert Metadata Concerns into IDS IM
- Front End Design
- Back End

Recap of the Architecture

Recap of the Architecture



Recap of the Architecture

- Admin introduces datasets into the application
- Application extracts metadata concerns
- Metadata is cast into an RDF knowledge graph using IDS IM
- Knowledge graph is written to backend (Jena/Fuseki)

- User browses and searches for datasets based on their metadata using the UI
- User input is translated into SPARQL Queries
- Queries are sent to Fuseki server
- Fuseki retrieves relevant datasets and sends them back to the user

Extractor Components

Component: Keyword Extraction

The aim is to extract the keywords that describes the entire underlying dataset in an unsupervised manner.

Keywords: For now, we defined the keywords as the higher level concepts that help us to understand the underlying data

Keyword Extraction: Approach

- We divide the approach into two parts
  - Extraction of metadata
  - Filtering of metadata

Approach: Extraction of Metadata

- Filter out the relevant columns from the dataset.
- For each relevant columns:
  - Compute the frequency for each word and select the words which are above the mean/threshold
  - From the words obtained above, use Falcon tool for NER and NED and obtain the relevant Wiki-ID. The knowledge base of choice is Wikidata
  - Using the Wiki-ID use the SPARQL API of Wikidata to get the meta-data of the entities. For now, we only extract instance-of from the KG.

Approach: Filtering of Metadata

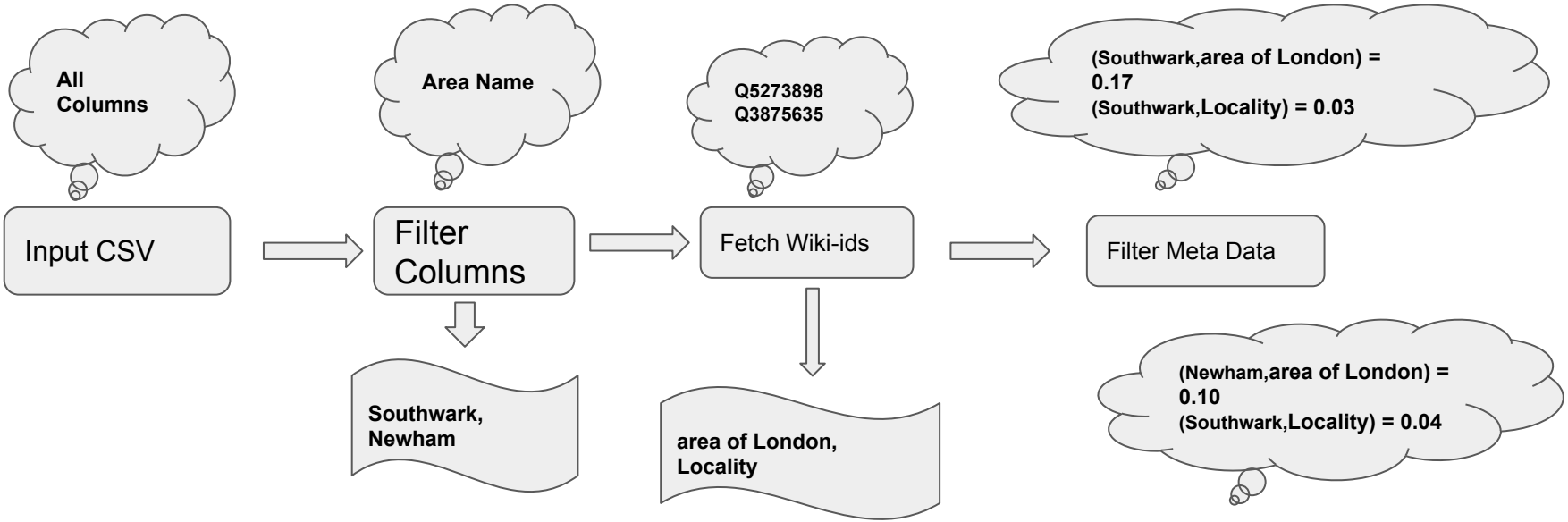
- Aim: Since the information obtained from the previous step will be prone to noise from NER and NED step, the system should be able to select the relevant information and filter out the noise from the metadata.
- Motivation: Each column represents itself and therefore the semantic information encoded in each column will also be similar. For example if a column contains information about cities in Germany, then we expect the column to only contain information related the cities in Germany. We exploit this constraint to weed out the noise from the data.
- For each relevant columns:
  - Obtain the metadata for each column
  - Compute the word embeddings of all the filtered words in the columns and the metadata obtained from NER and NED step.
    - We use a pretrained Word2Vec model which is trained on entire corpus of wikipedia text, the common crawl dataset, the Google News Dataset.
  - For each embedding in metadata:
    - Compute the cosine similarity with embeddings of the filtered words.
    - Average out all the score obtained above to get the final similarity score.
  - Filter the metadata with value based on different threshold.

Keyword Extraction: Example

It consists of the following columns:

- Date: In format YYY-MM-DD
- Area Name: City of London, Westminster etc
- Area Code: E09000001, E09000033 etc
- Retail And Recreation percent change from baseline: -1, 5, 4 etc
- Grocery And Pharmacy percent change from baseline: -2, 3, 8 etc
- Parks percent change from baseline: -4, 6, 9 etc
- Transit stations percent change from baseline: 1, 2, 3 etc
- Workplaces percent change from baseline: 4, 9, 10 etc
- Residential percent change from baseline: NA, 5, 6 etc

Keyword Extraction: Example



Component: Data Quality

- We check average of percentage of unique values in all columns of the file and let's say that is = a
- We check average of percentage of not null values in all columns of the file and let's say that is = b

- We calculate percentage of blank values in the file = c
- We calculate percentage of NaN in the file = d

Data Quality: Approach

- Finally, file quality is measured by the formula as below:

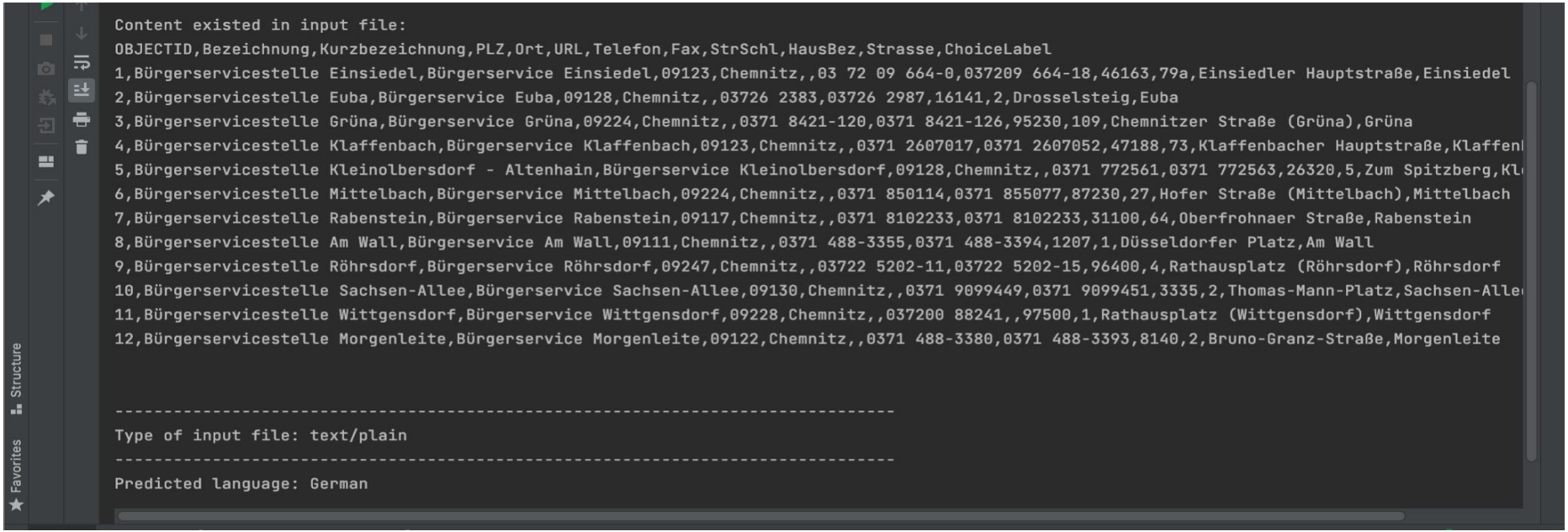
- File Quality = 0,8*(a) + 0,2*(b) - 0,1 (c + d)

- Quality of the file is expressed as below:

  > 85 Excellent
  55-85 Good
  30-55 Sufficient
  < 30 Bad

Language Extraction: Approach

- Use Apache Tika to parse the content of input file (CSV,pdf,docx,etc.)
- Use Apache OpenNLP to detect the language of parsed content



Extractor Todos

- Add more components (timeframe discovery etc.)
- Increase accuracy
- Add data quality metrics / refine formula
- Improve performance
- Let extractor run asynchronously

- Create API to let extractor components communicate with backend (modular design)

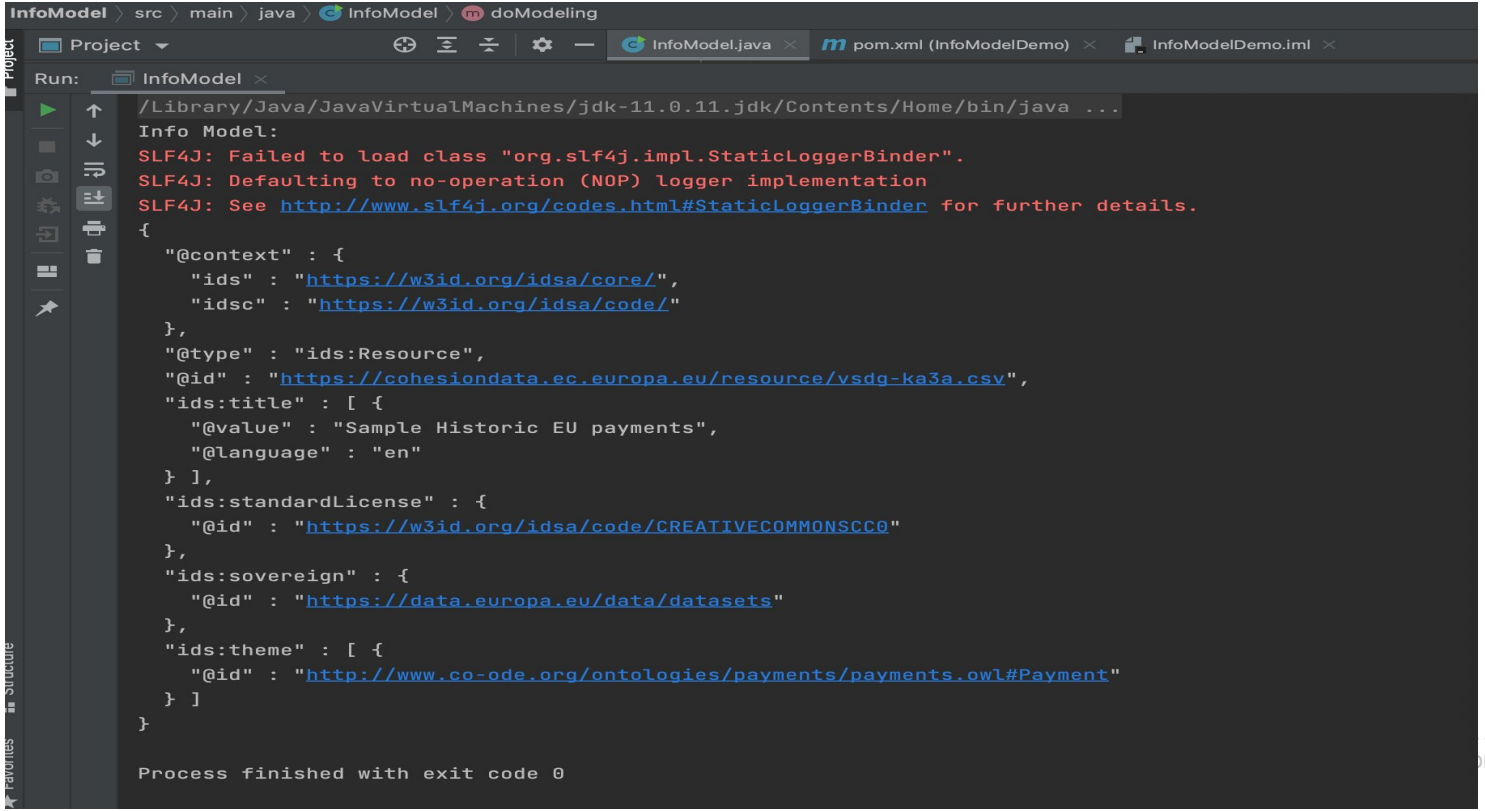
IDS IM RDF Generator

Component: Re-express Metadata into IDS IM

- After Extractor work on CSV file, re-express into IDS IM
- Create (Output) RDF Knowledge Graph (which we will stored in our Database)
- Resources and Properties in RDF are represent in the form of URIs

IDS IM: Output

Using the IDS IM Java API:



Frontend

Front End Tools

- React (Framework)
- CSS
- HTML

MetaData

Type your query here | Search

Home | FAQ | AboutUs | SIGN IN | SIGN UP

Start Your Data Journey With Us

What are you waiting for?

GET STARTED | WATCH TRAILER

List of DataSet!

Data

Lorem lipsum

Join the MetaData Community

You can unsubscribe at any time.

Your Email | Subscribe

About Us

How it works
Testimonials
Careers
Terms of Service

Contact Us

Contact
Support
Destinations
Sponsorships

MetaData

Front End Todo

- Add more Components
- Add Content to rest of pages
- Maps, filters,list
- Connect Frontend and Backend

Backend

Backend

- Work in progress and main concern for future
- Current status: can use Jena/Fuseki library classes
- Can “manually” read/write TDB datasets using library calls
- Can launch local (seperate) Fuseki server and read/write using UI

- Immediate concern: embed Fuseki into main maven project and communicate internally using API

- Main difficulty: Application has to run continuously as a server and handle requests

Thank you

Questions?
Slide theme credit - Isaiah Mulang