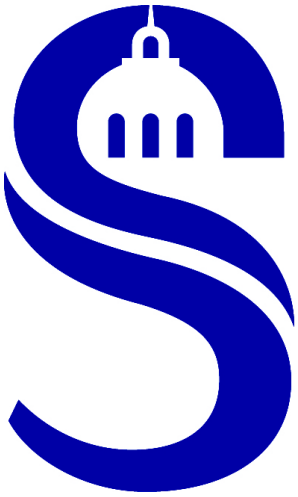

SCIENCES SORBONNE UNIVERSITÉ

Project_online-library-search-engine

DAAR Project choice A: online search engine for a library

(front + back + indexing + suggestions + no-DB : using Gutenberg online DB)

Teacher : Binh-Minh Bui-Xuan

"buixuan" <buixuan@lip6.fr>

Members (binôme)

Thibault ROCHE 3677376

<thibault.roche.1@etu.sorbonne-universite.fr>

thibault.roche.fr@gmail.com

Paul NGUYEN DIT SYVALA 3876651

<paul.nguyen_dit_syvala@etu.sorbonne-universite.fr>

pnds.5791@gmail.com

Github repo : https://github.com/throche/project_online-library-search-engine

Table of content

Project summary

This project is about creating a website with a search engine capable of researching through thousands of books in a near instant response time. The user is also given suggestions of other books based on certain algorithms studied during the semester.

At the time of delivery we have implemented the following :

- a front-end with Angular Framework
- a server made in Python3 with fastapi package
- multiple scripts in Bash and Python3
- multiple features explained with [more details here](#use-cases-features)

1- Installation

1.1- Back-end API in python (using fastapi package)

Open a terminal in project/src/server then enter :

pip install fastapi

pip install uvicorn

pip install -r requirements.txt

1.2- Front-end Client in Angular Typescript

Open a terminal in project/src/client then enter :

sudo apt install nodejs

sudo apt install npm

sudo npm install -g n

sudo n stable (update node to latest version)

hash -r

sudo npm install -g @angular/cli

2- How to start

2.1- (Only run once) : pre-indexing data is the offline work

Open a terminal in `project` then enter :

`python -m src.main_extractor` this generates indexes, you can open the file to toggle in/out certain function, see more in the [offline work : scripts](#offline-work-&-scripts)

2.2- Server

Open a terminal in `project` then enter :

`uvicorn src.server.main:app --reload`

2.3- Client

Open a terminal in `project/src/client` then enter :

npm install

npm run start

then open a browser and go to http://localhost:4200

You're good to go!

3- Use cases : features

3.1- Client side

- the user can use the website to search for a specific book using keywords (separated by whitespace)
- the research's results give the book's basic info and link to the Gutenberg Project book page
- the research's results are ranked
- the results from the search function is very fast (<1s)
- the user is given suggestions of books based on its last research (graph algorithms used)

3.2- Server side (online)

- we use a score system to rank our search results and suggestions
- the search results and suggestions are obtained very quickly (< 1sec)

3.3- Back-end (offline)

- we use scripts to crawl and download books from the Gutenberg Project (respectfully)
- we have 2000 books in English language registered, we can add more easily
- we run scripts to index all books and filter out lots of junk
- we run scripts to pre-run suggestions, using the Jaccard graph algorithm (proximity of lexical fields)

4- Exemple of usage

1) follow the [How to start] procedure
2) search for `mozart`
- displays several results
3) search for `mozart piano`
- displays less results by using multiple keywords search
4) search for `mozart piano jazz`
5) search for ` fffffff`
- displays no results as this word doesn’t belong to the index

5- Program architecture

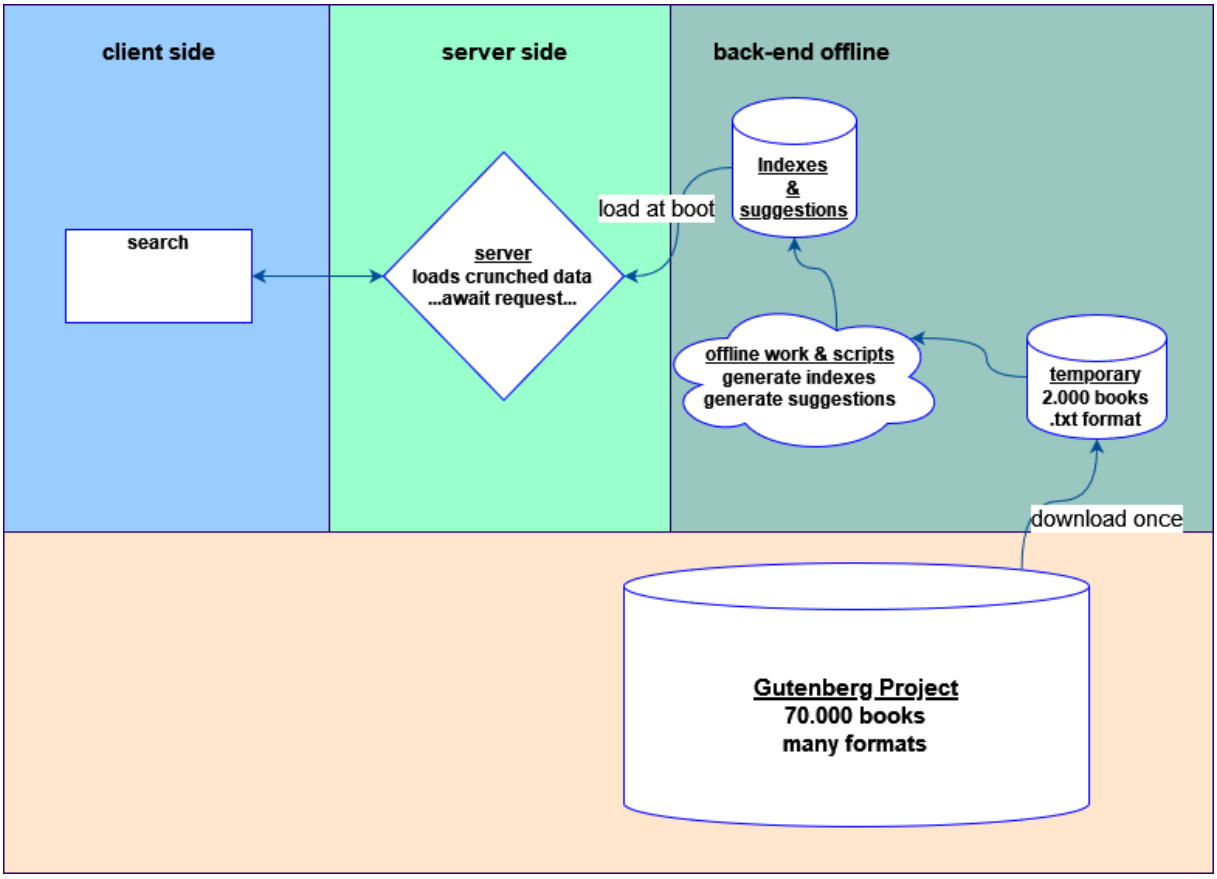

Figure 1- Diagram of program architechture

5.1- Client side

There is no data on the client side. When performing a search request, the client will receive small json files containing : book_id, title, author, release date, score.

The score is used to rank the results on screen.

The book_id is used to generate a link to the proper book section in the Gutenberg Project.

5.2- Server side (online)

When the server is booted, it will load three files in Python dictionaries then it won't read any other files. The loaded files are :

- `books_meta_data.csv` which contains meta-data for all books

- `index_global_unique_word_to_id.csv` which contains all unique words and book_ids which have them
- `books_distance.csv` which contains suggestions for every books

From `project/src/server/main.py` :

The server side uses a Rest API structure using the fastapi python package.

The function `read_item(q: str)` are the endpoint to access for the search function and the suggestion function.

They access the 3 indexes made during the offline pre-treatment operations.

The indexes are loaded into dictionaries, implemented using HashMaps.

When several words are searched, the query string is broken into single words and only books containing all words will be returned as results.

6- Offline work & scripts

Scripts are used to ease the work load on the server when it's online.

6.1- Download and unzip books

From `project/scripts` :

Gutenberg provides books in different format, we chose to download books in ASCII format to generated our indexes, we use :

wget -w 2 -m -H "http://www.gutenberg.org/robot/harvest?filetypes[]=txt&langs[]=en"

to download books in english and `.txt` format and :

find . -name "*.zip" | while read filename; do unzip -o -d "./data/" "$filename"; done;

to unzip all the files into a single `./data` folder. Some books are US-ASCII instead of ASCII, we filter them out with :

xargs rm -f <<< $(find . -regex "./[0-9]+-[0-9].txt")

6.2- Extracting meta-data and generating indexes

Functions are defined in `project/src/main_extractor.py` there are three functions, which can be commented out in the main. By default they are run one after the other when performing the following :

Open a terminal in `project` then enter :

python -m src.main_extractor

6.2.1- Extracting meta-data

From `project/src/main_extractor.py` :

We use `Extractor.extract_meta_data()` to extract meta-data from all books into single csv file containing all books' meta-data : book_id, title, author, release date

It generates the file : `project/data/meta/books_meta_data.csv`

Each line is : `book_id;title;author;release_date;` where book_id is sorted

6.2.2- Unique word index for every book

From `project/src/main_extractor.py` :

We use `Extractor.index_unique_word_for_all_books()` to generate one index per book in the foler `project/data/index/unique_word/`.

Each file name is : `index_unique_word_#.csv` where # is the book id

Each line is : `word;nb_occurrences;` where word is sorted and is a unique word from the book which was not filtered out as junk

6.2.3- Global index of unique words

From `project/src/main_extractor.py` :

We use `Extractor.global_index_unique_word()` to generate a single global index which contains all unique words in every book's index with the list of book_ids using said word along with number of occurrences.

It generates the file : `project/data/index/global/index_global_unique_word_to_id.csv`

Each line is : `word;book_id1;nb_occurences;book_id2;nb_occurences;...;` where word is sorted and unique, there can be one or more `book_id;nb_occurences;` following a word per line.

6.3- Comparing book indexes and generating neighbours for suggestions

6.3.1- Graph of neighbours

From `project/src/main_distance_books.py` :

We use `create_distance_index()` to generate an index file which contains for each book index the list of indexes of neighbours based on a closeness calculation algorithm.

It generates the file : `project/data/distance/books_distance.csv`

Each line is : `id;id_neighbour_1;id_neighbour_2;id_neighbour_3;...;` where id is sorted and unique.

6.3.2- Calculating closeness between 2 books

From `project/src/main_distance_books.py` :

We use `compare_unique_indexes(idx1, idx2)` to compare 2 books indexes file which contains for each book index all unique words. If the 2 books are similar, we define them as neighbours for the suggestion feature. It uses for each word a calculated score based on the rarity of the word.

6.3.3- Calculating words scores

From `project/src/main_distance_books.py` :

We use `create_word_scores()` to define for each word appearing in all the books of the database, a score based on its rarity. A word with a higher score means that the word appears in less books than a word with a lower score.

It generates the file : `project/data/distance/words_scores.csv`

Each line is : `word;score;` where word is sorted and unique, and score a float representing the word's rarity in the book database.

7- Suggestions : Graph algorithms

The algorithm for determining if 2 books are related or not is based on the calculation of a Jaccard distance with a closeness score using the unique index of each book.

The unique index of a book lists every word and its number of occurrences.

Our first approach for determining if 2 books were related was to count the number of different words they had in common. 2 books on the same topic should have a lot of words in common.

After trying this implementation, we decided to improve our algorithm by attributing different weight to the different words in common that 2 books had. If 2 books have a specific word in common, they have a much higher chance to be related (for example 2 books with the word 'algorithm') than 2 books with a very common word (for example 2 books with the word 'England').

This implementation was inspired from the Term Frequency-Inverse Document Frequency ranking method (TF-IDF).

We calculated a rarity score for each word based on the number of books in which the word could be found, among all the books in our database. The calculated score is higher for words that are present in fewer books.

When comparing 2 book indexes, if a word is present in both books, the closeness score is increased by the amount of the rarity score of the found word. If the total closeness score is higher than a threshold, we consider the 2 books as neighbours for suggestions.

We decided for each book to keep the 50 highest 'closeness scores' in the file books_distance.csv

8- Credits

Book database

https://www.gutenberg.org/

Jaccard distance

https://www-apr.lip6.fr/~buixuan/daar2021

TF-IDF ranking method

https://kmwllc.com/index.php/2020/03/20/understanding-tf-idf-and-bm-25/