WikiIndexer

➢ Use case

1. The user enters an article name or chooses a file to upload and then click submit.
2. The formValitator check if at least one field is filled before submitting the request
3. The application does its magic
4. The list of ArticleDTO is returned as 'articles' to 'chart.html'
5. The list is used by 'chart.js' to create the charts
6. The words chart modifies (to show words from the article) if the user mouses over an article
7. The articles char modifies (to show articles containing the word) if the user mouses over a word

➢ Dependencies

The application was created with spring boot

For persistence it uses spring JPA and MySql.

For UI it uses tymeleaf, css, javascript, CanvasJS

➢ Implementation

MainController calls MainService to:

- get the words from an article given as a title
- get the words from all articles given by titles in a file

MainService:

- calls WordsCountService to count the words from an article
- calls FileParserService to parse the file and then instances of TextParserThread (who use WordsCountService to count the words from an article)
- keeps track of the time for each WordsCountService call

WordsCountService:

- uses ArticleDatabaseService to get the articel from the database(if it exists)
- uses WikiRequestService to get the article from Wikipedia
- uses javax.xml.parsers.DocumentBuilderFactory to parse the article
- uses CommonWordsCheckService (to check if the words are not 'a', 'the', ...) + it doesn't accept words that are shorter than 2 chars
- counts the number of apparitions for each word
- creates an ArticleDTO
- uses ArticleDatabaseService to save the article in the database

ArticleDatabaseService uses:

- ArticleRepository to save/interrogate the database for an article

OBS:

- Files are not saved because we don't see any need to do it (it fills the memory for nothing)
- We tried to use Java 8 as much as possible but because of that we had cases where we had to iterate through a list 3 times to get all we needed from it
- The controlled saves a list of ArticleDTO in the model because we wanted to make a more interactive graph

Classes:

1. **Controller**

MainController - Single controller of the application

- Responds to GET and POST on "/"

2. **Model**

ArticleEntity - entity that is mapped on table "article"

- Contains: id, title, wordList (from OneToMany)

WordEntity - entity that is mapped on table "word"

- Contains: id, word, articleEntity (from ManyToOne)

3. Data Transfer Objects

ArticleDTO - object representing an article

- Contains: time, title, wordList, fromDatabase(true if it came from a database)

UserFormDTO - object representing a form

- Contains: articleName, fileName(MultipartFile)

WordDTO - object representing a word

- Contains: word, nrAppar

4. Repository

ArticleRepository - JPARepository for ArticleEntity

- Added method: findByTitle

5. Services

ArticleDatabseService - Saves/queries the database for articles

CommonWordsCheckService - Checks if a word is in the top 100 common words in English (used to skit this words)

- Implementation has a @PostConstruct method to create a set of top 100 common words

FileParserService - Reads the titles file and returns titles

WikiRequestService - Sets a connection to wikipedia and asks for an article

WordsCountService - Uses ArticleDatabaseService to check if the article is in the database

- Uses WikiRequestService to get the article, and then parses it

- Counts the words and returns a ArticleDTO with all the words in the article

MainService

- Uses WordsCountSercvice to get the words of the articles
- Finds top 10 words in the articles
- Finds top 10 words from all articles put together
- Returns a list of ArticleDTO (one for every title + one ArticleDTO for top 10 from all articles)
- If multiple articles are requested, it uses an Executor that starts Callable objects (threads)

6. **Threads**

TextParserThread

- Callable object that uses WordsCountService to count the words from an article

➢ HTML pages

mainPage - Contains the form

- Uses 'formValidator.js' to validate the form before sending the POST

chart - Displays the charts and other information

- Uses 'chart.js' to create the charts, update them and display other article parameters