Author: HoKang Yu
Date: 02/14/2021

Assignment 01: Tokenizing and Indexing

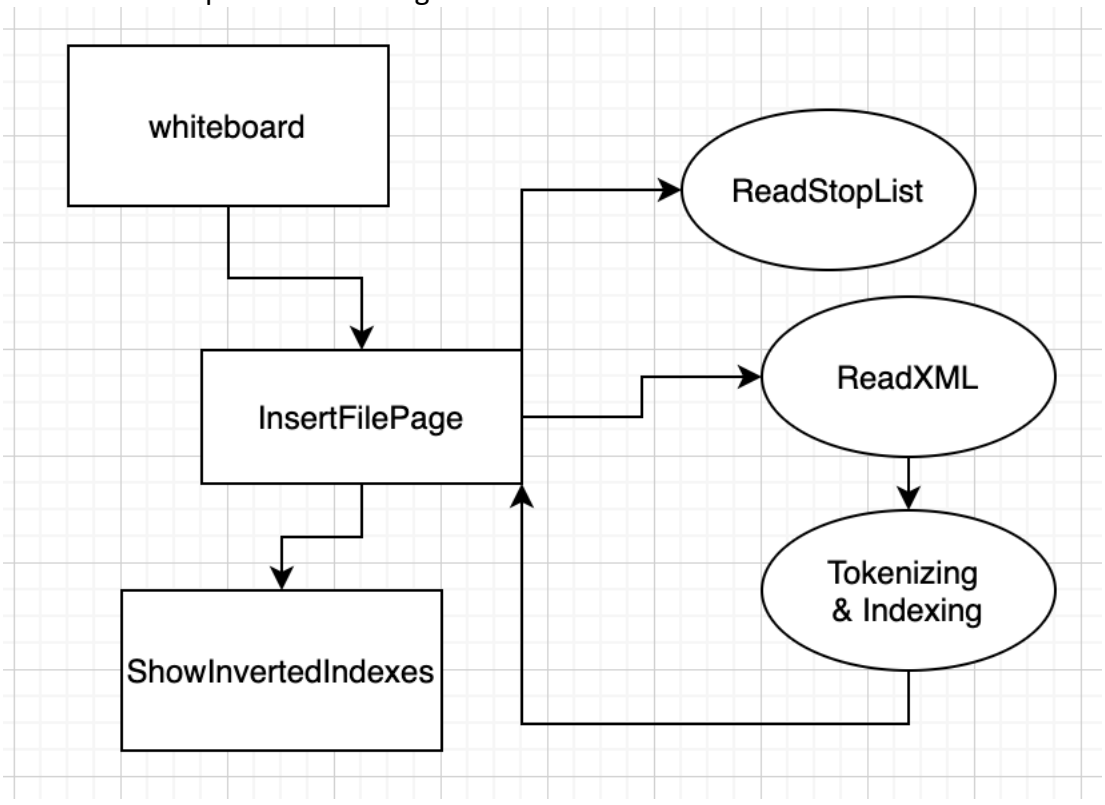
Access Information and Instruction on how to run:
Web: https://cryptic-lake-70423.herokuapp.com/
Github: https://github.com/ImCityHunter/InfoRetreivalWeb

Access Online:
Click choose file, choose a file, then click submit.

Access Locally:
Download the file from github, use terminal and access the folder. Possibly need to install react and install react bootstrap (npm install react-bootstrap bootstrap); otherwise, the styling will be off. Then just type in npm start; the web shall be running localhost:3000 in the browser.

Code Design:

Use React.js to create a website. The following diagram is the flow diagram I drew using Draw.IO that represent the design of this website



Brief Explanation:
There are three main class within this webpage: *whiteboard*, *insertfilepage* and *showInvertedIndex*. *Whiteboard* has nothing but just headers of the page. *InsertFilePage* is where insert file is implemented. Before the file was inserted, the stoplist.text has been read and formed a set in the background. Then once the file is inserted, I have written functions to parse the XML and tokenizing and indexing. The *ShowInverdexes* class will only show result if file has been successfully tokenized and inverted indexes created.

Main Functions:

ReadStopList: By the default, stopList is already read and stored. The stopList.txt will immediately be used to create a set which will be used later in tokenizing.

Reject not ".xml" file: this function is not needed, but I still wrote a function to check if an inserted file is xml or not. If the file is not xml, it will alert.

ParseXML: By default, we know only cf74-79 will be used for this homework, thus, we know all relevant information is stored in each "RECORD" tag. Thus, we just need to find out what tags are needed for each record tag. I have selected "Abstract", "Extract", and "Title" for tokenizing. These are the tags that contain the most information regarding to "words". And if a record does not have RECORDNUM, the codes will give it a number based on the sequence it was read. For each record, it will go through for loop to send abstract, extract, and title to tokenizing if the record has them.

Tokenizing and indexing:

Sentence Separators, punctuation marks: when a text is being tokenized, I first split them by punctuation marks. Any punctuation mark that is NOT ' or – (apostrophe and hyphen) will be used in the split function. Simply because apostrophe and hyphen are not sentences enders. Punctuation marks such as semi-colon, any shape of brackets, space, and etc are used to separated words.

Apostrophe: Any word with apostrophe will be removed for now, because there are too many possibilities. In the stoplist.text, "should" and "can" are stop words but shouldn't/shouldnt/cant/can't are not in the stoplist. In order to prevent having words that are not actual words to be included in the final result, I decided to remove any word with apostrophe for now. Some open libraries will be needed if wanting to maximize this.

Lowercase: Once the text is splitted by word, each word will be converted to lowercase

Hyphen: hyphen will be replaced by empty space; thus, "x-ray" becomes "xray".

For each word that is read, it will be added to a dictionary (key, value pair). Key is the word, value is an array of record_id. But before a record_id is added to the array, the function will also check if the record_id is in the array or not; if the id already existed, it will not be added again.

Sort: After all the inverted index has been created, read through all keys/words and store the keys in an array, and use default function to sort the array of keys. Then when printing out, all keys/words will be printed out in alphabetic order. When each key/word is printed, use default function to sort its value (array of indexes).

Calculate time: Before parseXML starts, I set a time stamp, when all indexes are done, I set a time stamp again. The difference divides by 1000 is the estimate time this function takes.

Calculate Memory Space: Use standard byte size for number, string, boolean and use recursion to find the byte size for Object and Array. The result will be estimated in kb.