CSCI 5408

DATA MANAGEMENT AND WAREHOUSING

ASSIGNMENT - 2

Banner ID: B00952865

GitLab Assignment Link:
https://git.cs.dal.ca/apurohit/CSCI5408_F23_B00952865_AdityaMaheshbhai_Purohit/-/tree/main/A2

Table of Contents

Problem 1A: Reuter News Data Reading & Transformation and storing in MongoDb.

Pre-Requisite: Install MongoDb

I have first installed MongoDb database [1] in my machine, so that my java program can store the new articles in MongoDb later on. Following are the steps that I followed:

Step 1: Go to the MongoDb website and download the setup for MongoDb Community edition. https://www.mongodb.com/try/download/community

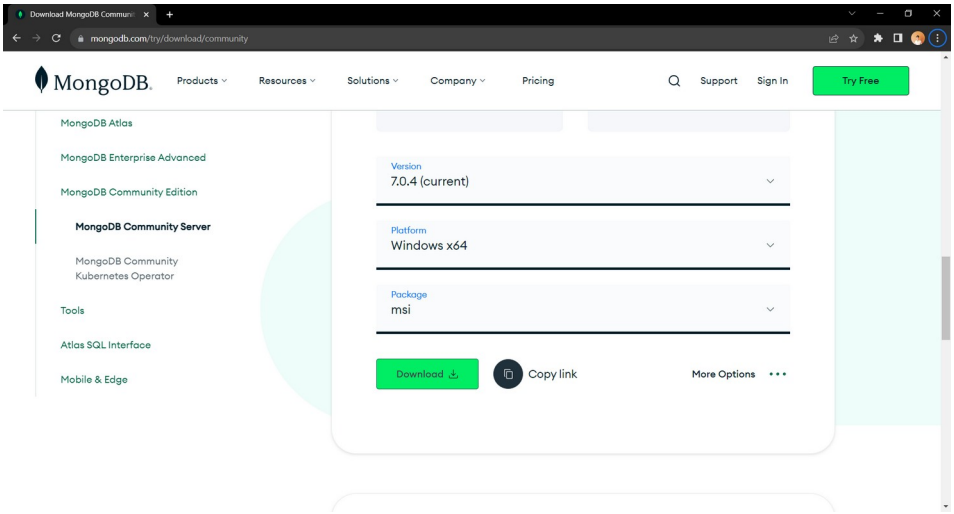


Figure 1: MongoDB Community Edition download page [1].

Step 2: Execute the MongoDB setup and click on next.

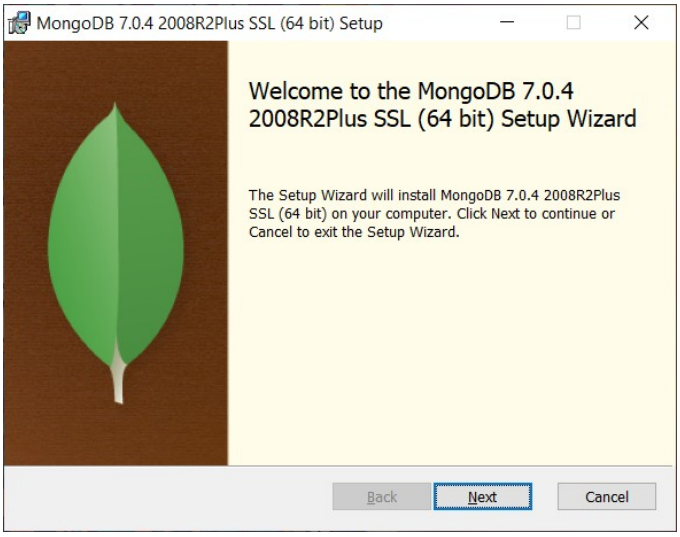


Figure 2: MongoDB Setup wizard welcome page [1].

Step 3: Accept the terms and click on next.

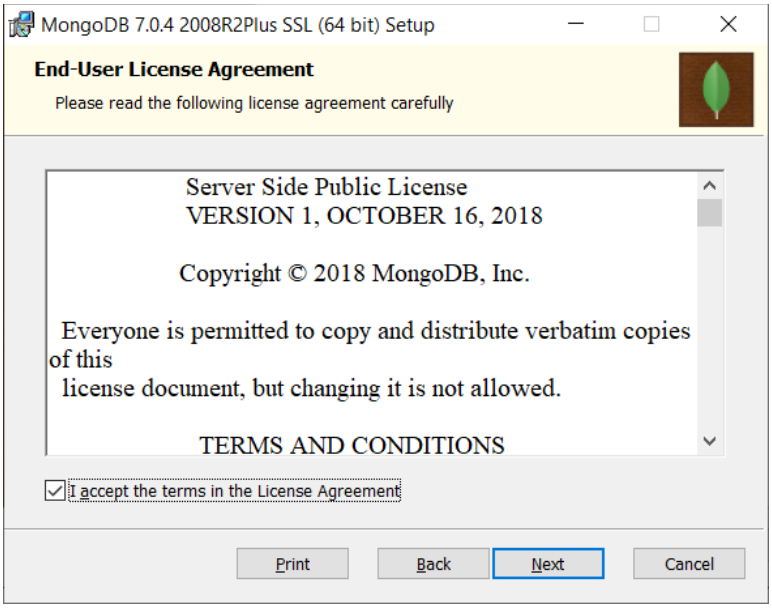


Figure 3: Accept the terms of MongoDB [1].

Step 4: Choose complete installation option, keep the default settings and click on next as show in figure 4 and figure 5.

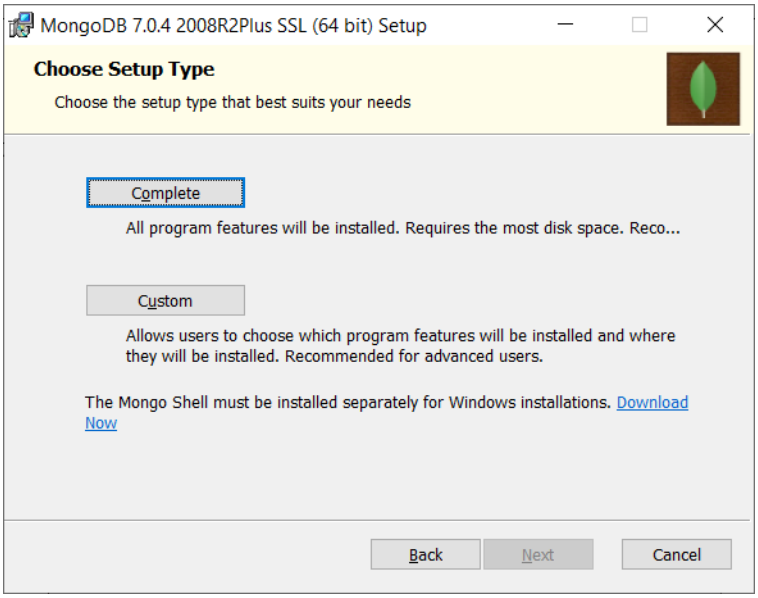


Figure 4: Choose complete installation [1].

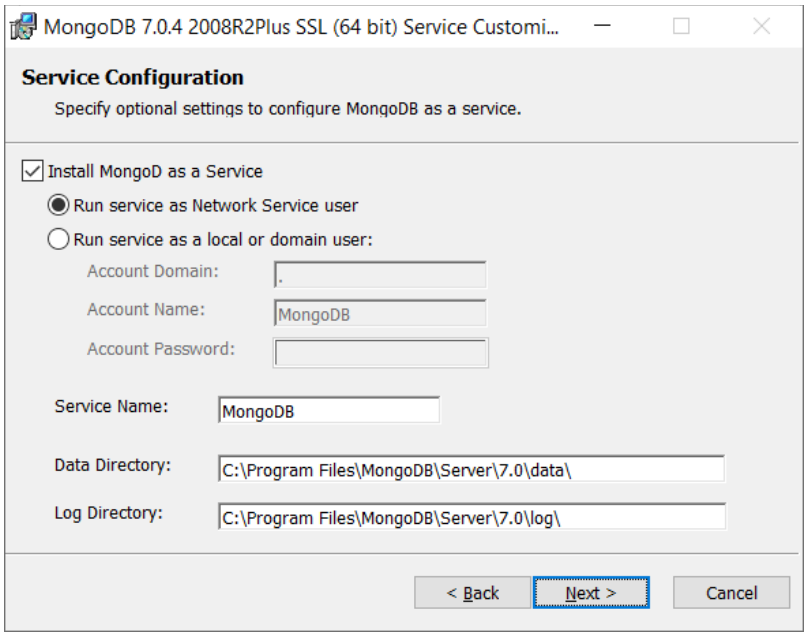


Figure 5: Default service configuration of MongoDB [1].

Step 5: Select MongoDB Compass installation and click on next. This is a graphical interface for MongoDB, which will be useful later on.

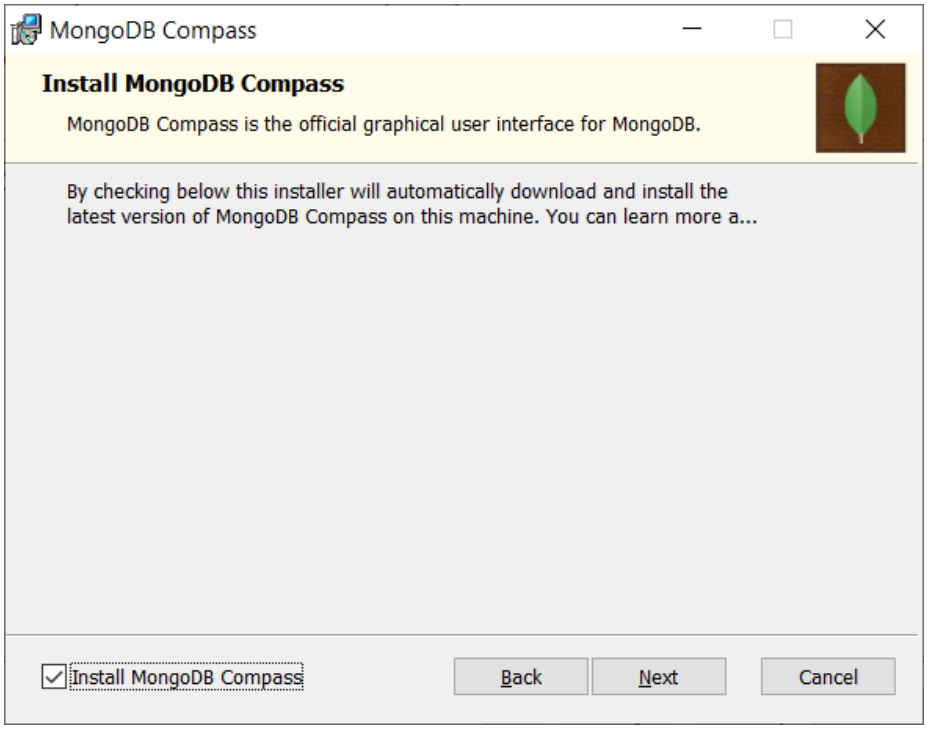


Figure 6: MongoDB Compass option selection [1].

Step 6: Click on install button and wait for the installation to complete.

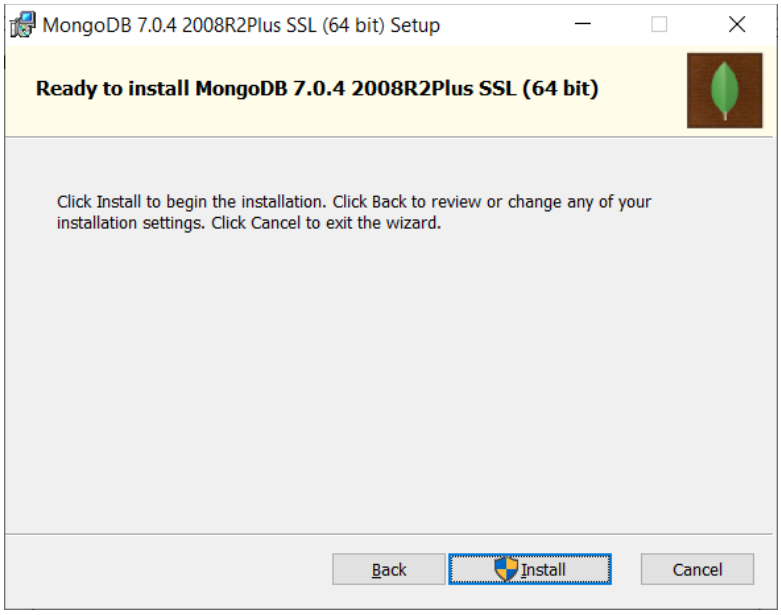


Figure 7: Final installation step for MongoDB [1].

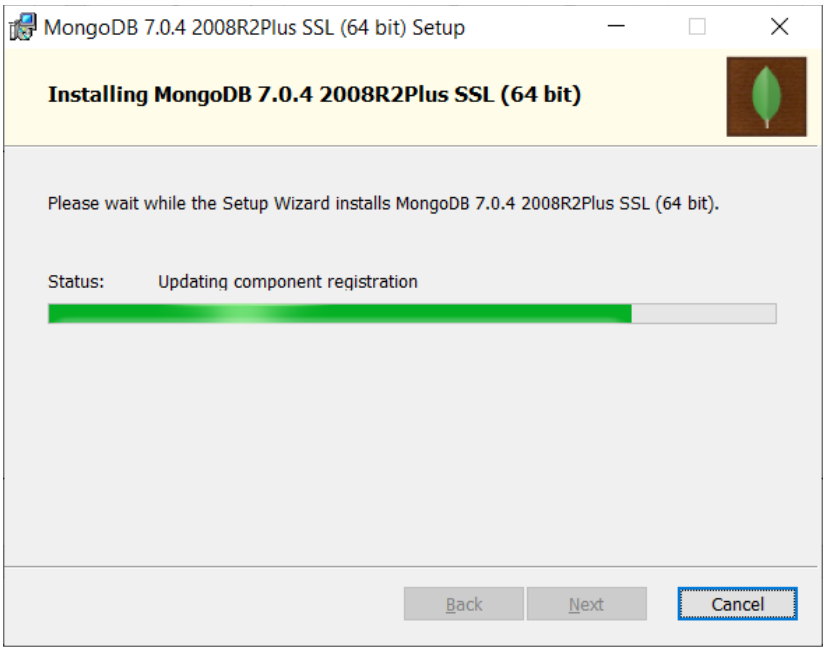


Figure 8: MongoDB installation in progress [1].

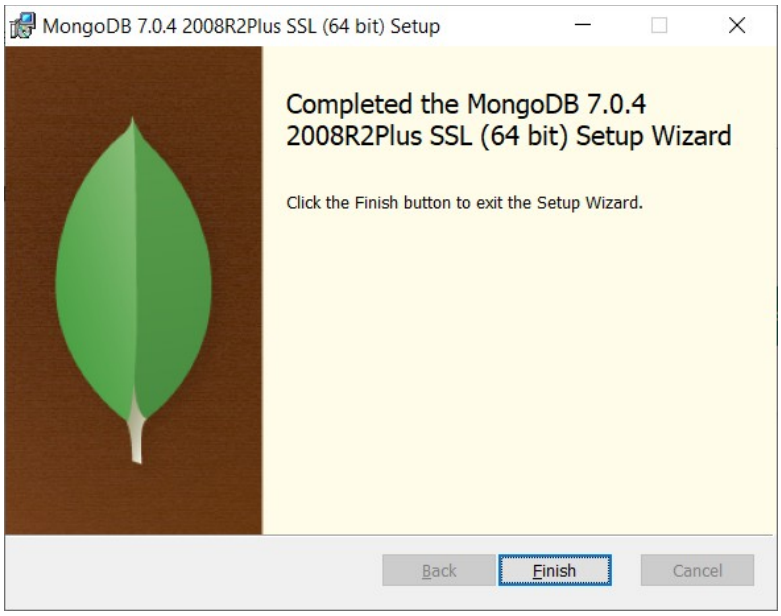


Figure 9: Installation completed for MongoDB [1].

Project Setup and Structure

I have a created a java maven project which contains a main method class called "**ReutRead**" as show in figure 10. Whole java code is commented as per JavaDoc commenting standards.

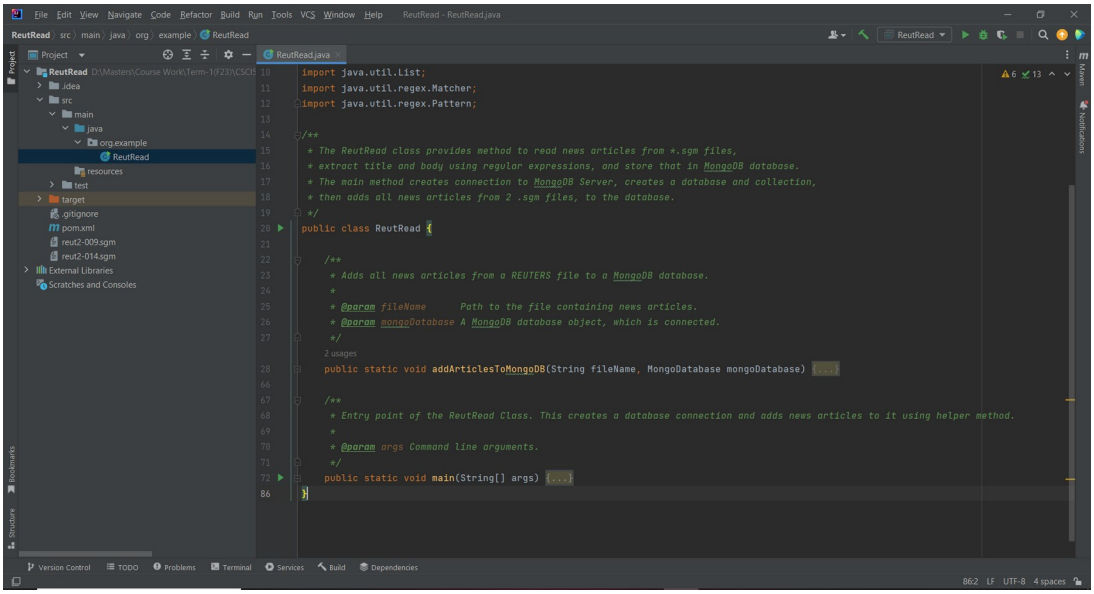


Figure 10: ReutRead.java file and ReutRead Class [2] [3] [4] [5].

In this folder, I have kept both the .sgm files in the root directory as show in figure 10. Moreover, in the ReutRead java class I have 2 methods: main() and addArticlesToMongoDB(). Here, the main method creates a MongoDB server connection, database and then invokes the addArticlesToMongoDB() method twice with 2 different .sgm files.

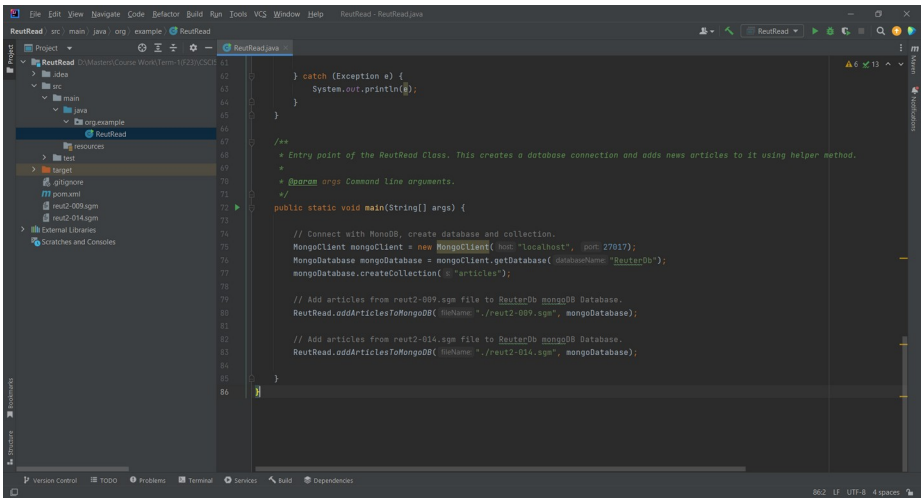


Figure 11: main() method logic of ReutRead Class [2] [3] [4] [5].

The addArticlesToMongoDB(..) contains the core logic of parsing title and body of news articles, as well as adding them to the ReutersDB named MonogDB database as shown in figure 12.

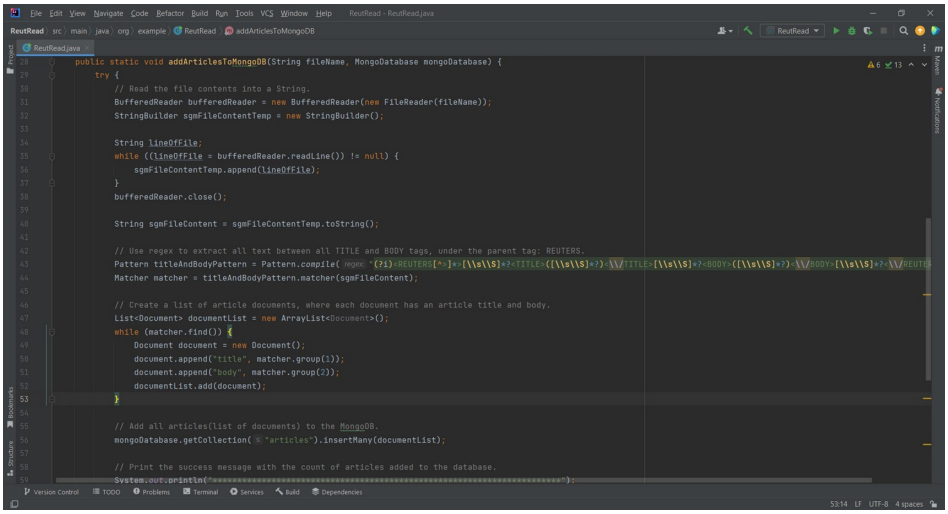


Figure 12: addArticlesToMongoDB method's code overview [2] [3] [4] [5].

Explanation of addArticlesToMongoDB method: The lines 30 to 40, are reading the contents of file from the particular .sgm file using BufferedReader and then storing the whole content into a string variable. After that, a single Regular Expression pattern is used to extract the content between <TITLE> and </TITLE> as well as between <BODY> and </BODY> tags. These tags should be under the <REUTERS>…</REUTERS> tags to get parsed properly. In the regex, the *(?i)* is used for case insensitive comparisons. Then we look for <REUTER (some attributes for this tag)> kind of a pattern using *<REUTERS[^>]*>*, then we look for a starting tag <TITLE> with some optional space characters(i.e. space, tab or newline character.) or non-space characters (i.e. a-z, A-Z, etc) which maybe present before the start of this tag. This is achieved using the *[\s\S]*?<TITLE>* portion of the regex. Then we look for the end tag </TITLE> and extract whatever is before </TITLE> tag and after the <TITLE> tag. This is achieved using ***(***[\s\S]*?***)***<\/TITLE> part of regular expression. The round parenthesis, will help us to extract the text between these <TITLE> </TITLE> tags.

Then the same extraction is performed with the body tags as well using the *[\s\S]*?<BODY>([\s\S]*?)<\/BODY>* part of the regex. Finally, we check for the ending tag </REUTERS>, to conclude the extraction process for a single news article. This matching process will happen for whole string, until we extract all the news articles.

For each article parsed, each title and body is added as a MongoDB document and then this document is then added into a list of documents. Finally, the whole list of documents is added into the articles collection of MongoDB database ReuterDb. A message is also displayed in the end to show how count of articles parsed from the current file and added to database.

The program explanation is also explained in terms of flowchart and algorithm later on in this report.

Functional Testing

I have kept the reut2-009.sgm and reut2-014.sgm files in the root directory of my java maven project before the execution of program. Then, I execute the main method of my java program.

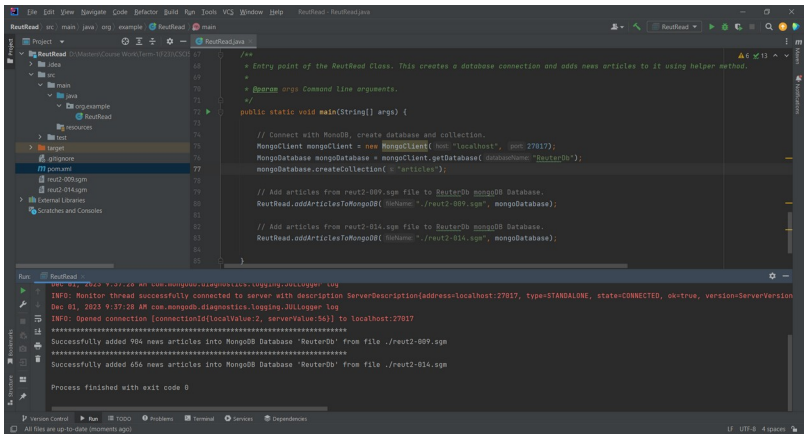


Figure 13: Execution of ReutRead program [2] [3] [4] [5].

Expected Result: Some articles should appear in the MonogDB database “ReuterDb”.

Actual Result: 1560 articles are added to the MongoFB database “ReuterDb”.

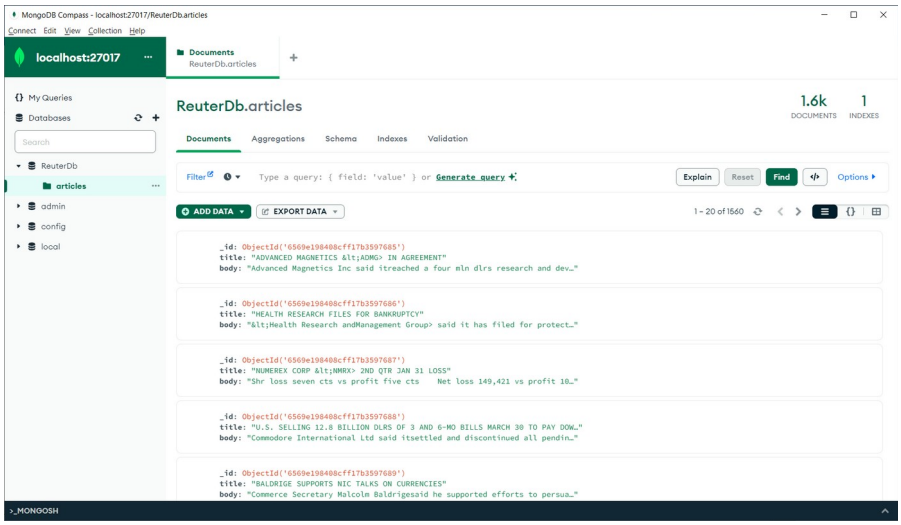


Figure 14: MongoDB Compass showing the 1560 articles added by program [1].

Test Conclusion: Passed

Flowchart

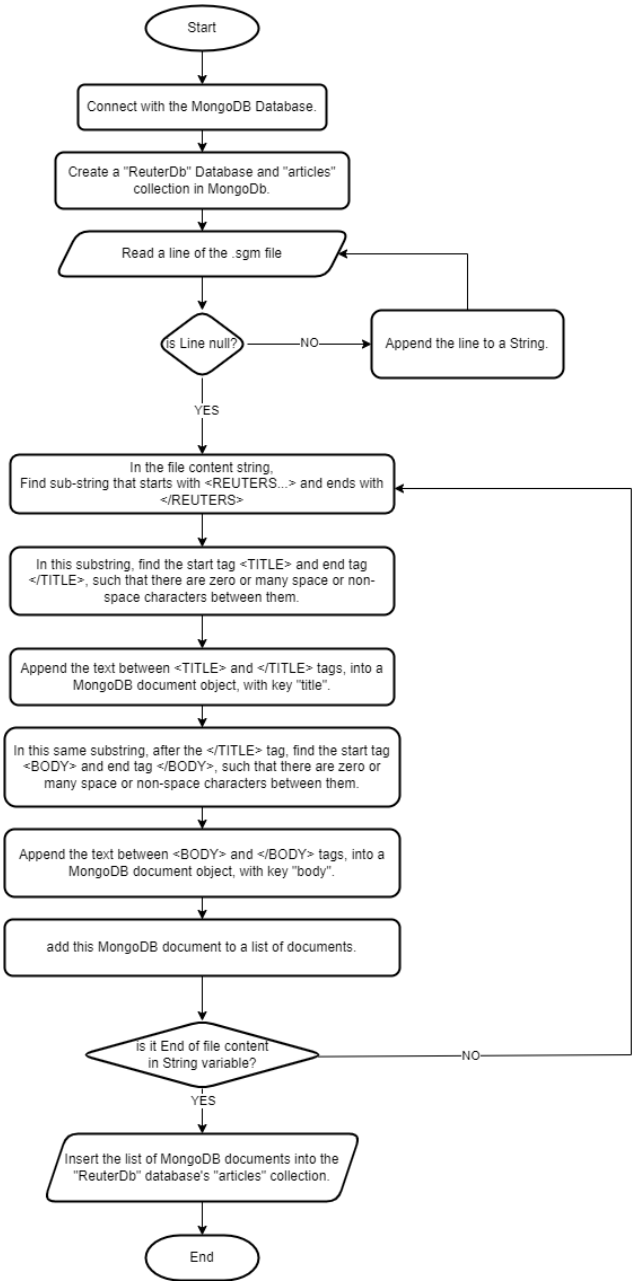


Figure 15: Flowchart of ReutRead program - Problem 1A [6].

This flowchart logic can be repeated for each .sgm file, whose content needs to be added to MongoDB. Moreover, in program, there is just a single regular expression that extracts both tile and body contents from the reuter new articles. However, for better understanding that regex logic is explained in multiple steps in the flowchart. From code perspective, regex pattern was explained in depth, in the “Explanation of addArticlesToMongoDB method” part of Project Setup and Structure section.

Algorithm

Step-1: Start.

Step-2: Connect with the MongoDB Community Server.

Step-3: Create “ReuterDb” database and “articles” collection in MongoDB.

Step-4: Read a single line from .sgm file.

Step-5: Check if the line is null or not, if not, append it to a String and go back to step-4.

Step-6: In the file content string, find sub-string that starts with <REUTERS...> and ends with </REUTERS>

Step-7: In this substring, find the start tag <TITLE> and end tag </TITLE>, such that there are zero or many space or non-space characters between them.

Step-8: Append the text between <TITLE> and </TITLE> tags, into a MongoDB document object, with key "title".

Step-9: In this same substring, after the </TITLE> tag, find the start tag <BODY> and end tag </BODY>, such that there are zero or many space or non-space characters between them.

Step-10: Append the text between <BODY> and </BODY> tags, into a MongoDB document object, with key "body".

Step-11: add this MongoDB document to a list of documents.

Step-12: If the file content is fully read, add list of documents to the MonogDB, else go back to step-6.

Step-13: If having more files, start again from Step-4, else go to Step-14.

Step-14: End

Problem 1B: Reuter News Data Processing using Spark.

Pre-Requisite: Configure and initialize Apache Spark cluster in GCP

Step-1: Go to Cloud Dataproc service in Google cloud platform using the search option and click create cluster.

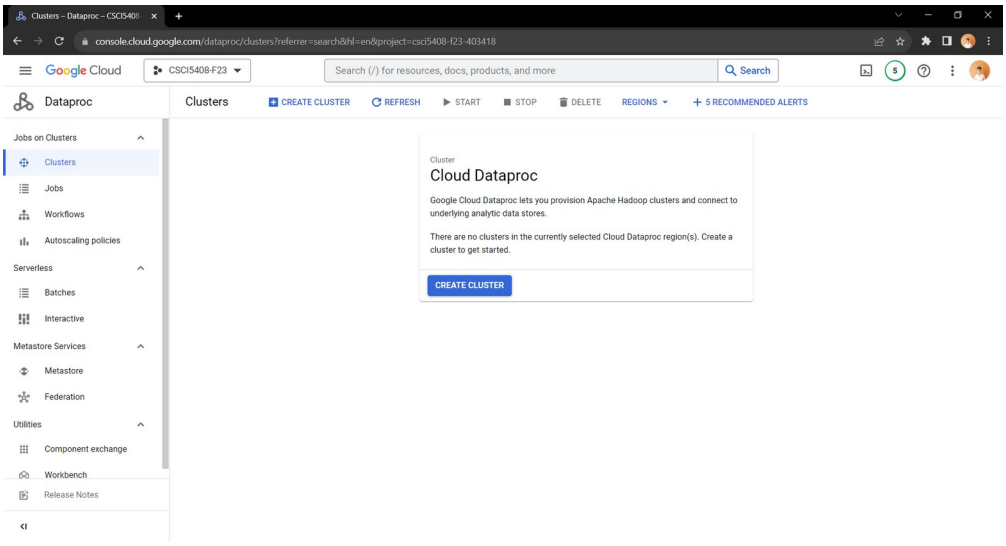


Figure 16: GCP Dataproc home screen [7].

Step-2: Give any convenient name to the cluster.

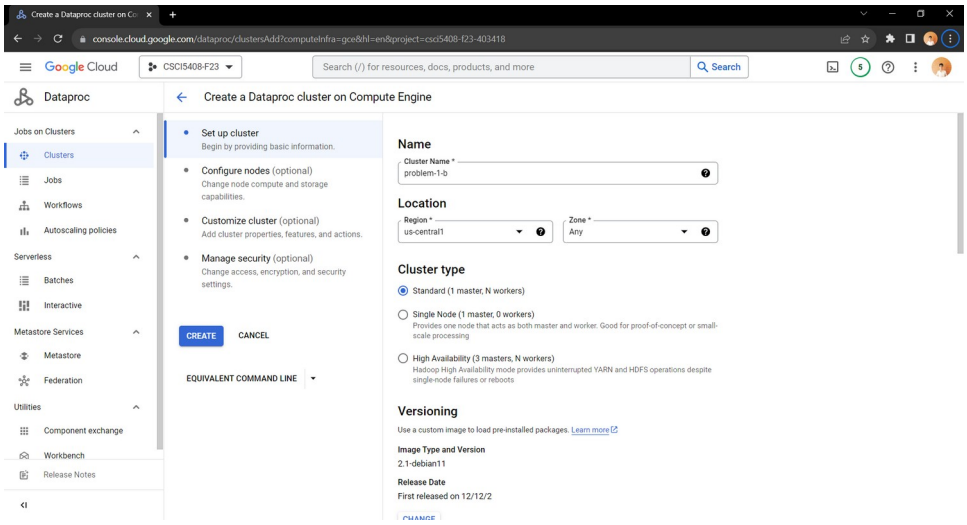


Figure 17: Creating cluster [7]

Step-3: Reduced the vCPUs of worked nodes to 2, due to usage limits of my account. Click create.

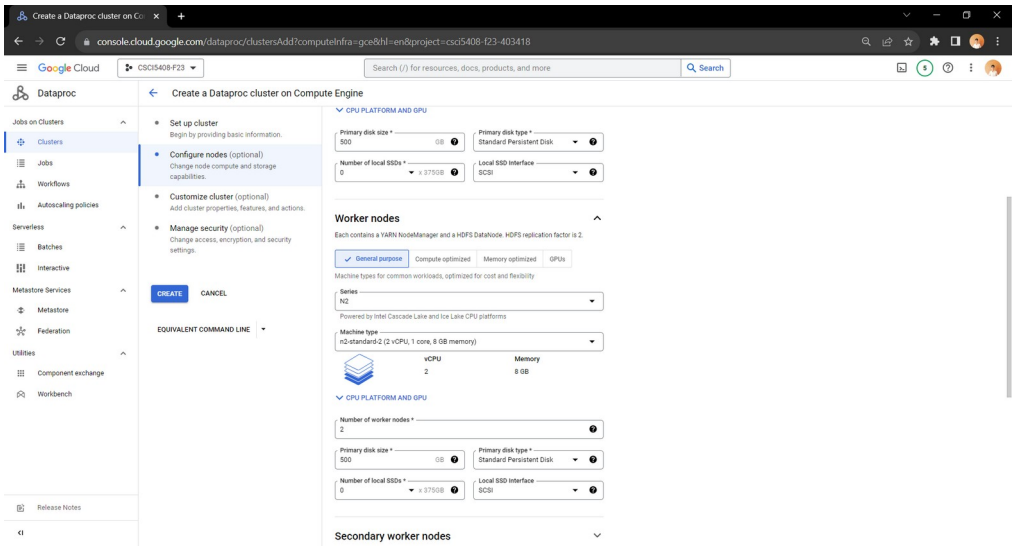


Figure 18: Worker nodes vCPU reduced [7]

Step-4: After a while, you should see all the nodes of cluster running.

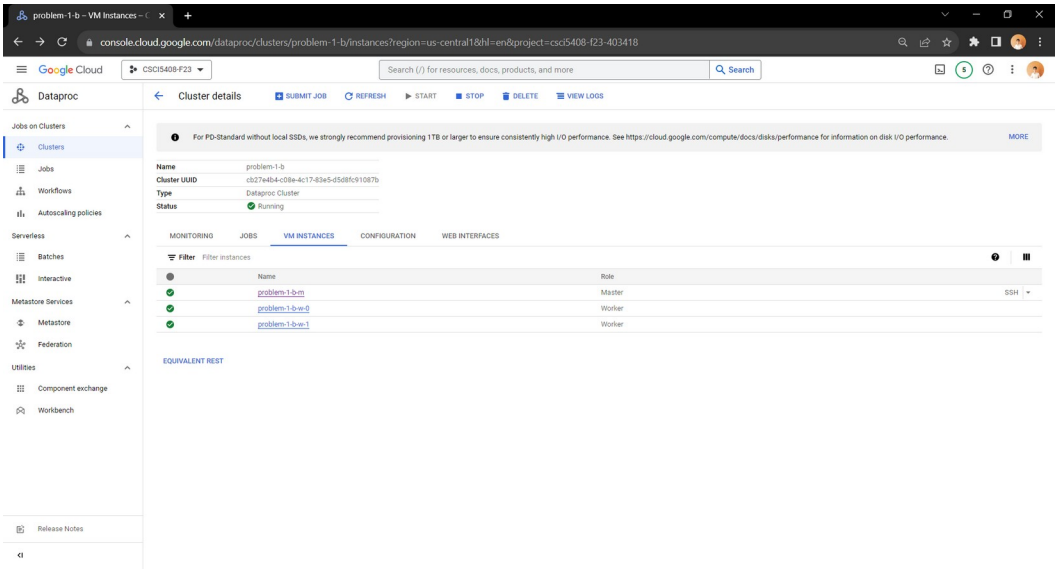


Figure 19: Cluster running [7]

Step-5: Connect to master node using ssh.

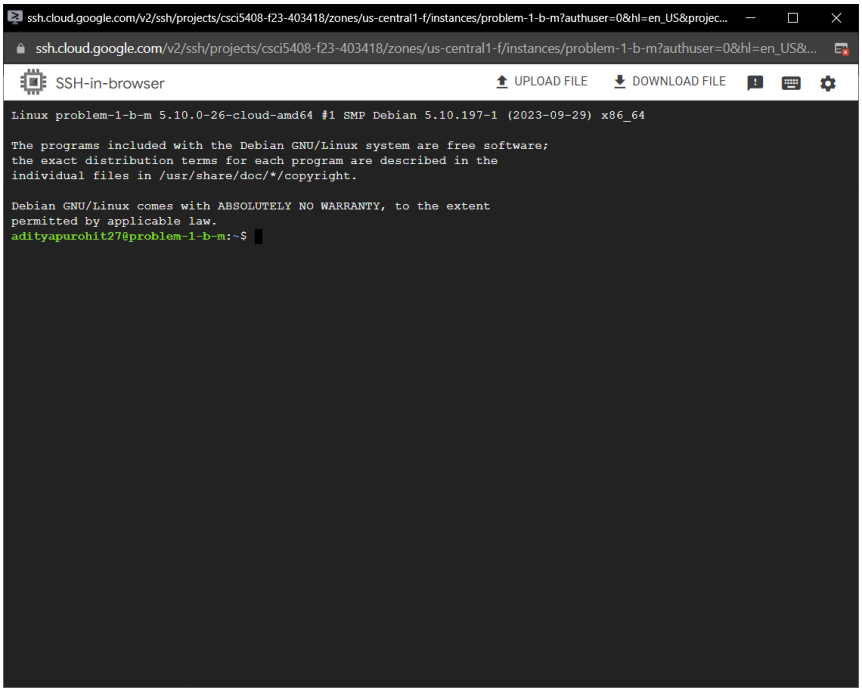


Figure 20: Connected to master node using SSH [7]

Explanation of task completion.

I logged into my GCP account and searched for the Dataproc service, so that I can create my Apache Spark cluster. I gave a name to my cluster as “problem-1-b” and reduced the count of vCPUs to 2 for the worker nodes, due to some account limitations. After that, I clicked on create cluster and waited for sometime till all nodes in the cluster started running. To test the connectivity with the master node, I used the ssh option, which worked fine.

Then, I created a java maven project, with a single .java file called ReutSpark.java. I also kept two more files in the root folder of the project, one called “stopwords.txt” [8] which has all stop words, which will be used for cleaning and another file “reut2-009.sgm” which has all the main data, in which the word frequency counting needs to happen.

Overall, from code perspective, I first create a spark session and then load the whole .sgm file into a Spark dataset of lines. Then, for pre-processing, these lines are made upper case, cleaned (remove special characters and start & end tags), split into words and then finally the stop words are removed. After that, using groupBy function of Spark, the word frequency is calculated and displayed for each word. Finally, the highest frequency word and lowest frequency word is displayed with the help of descending and ascending sorting respectively.

Java Program

The java project structure looks something like shown in figure 21. There are 4 important files.

ReutSpark.java – This has all the core logic for word frequency. All parts of code are commented using Javadoc standards.

reut2-009.sqm is the input file for word frequency calculation.

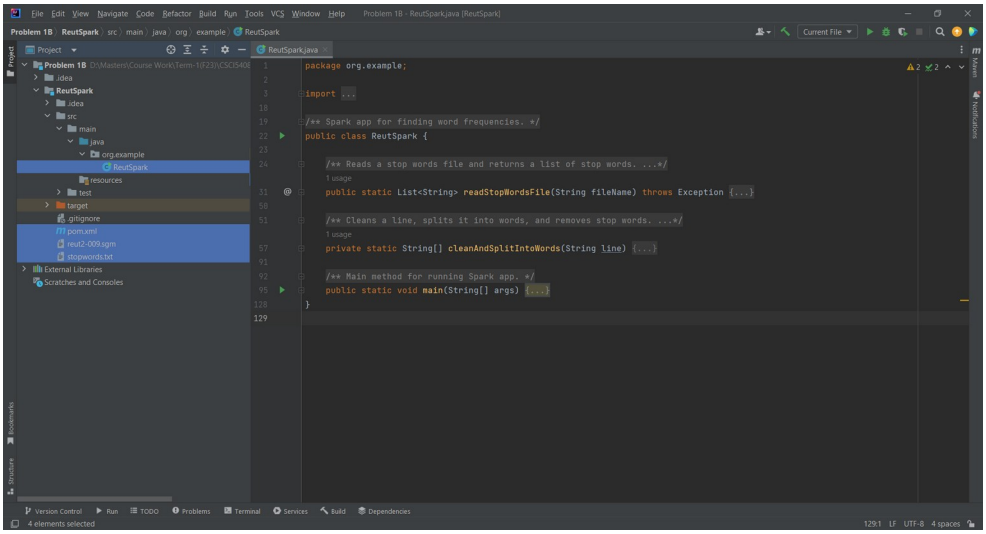


Figure 21: Overview of ReutSpark.java file and folder structure of project [2] [3] [4].

Pom.xml - 2 spark related dependencies are used in my maven project's pom.xml in order to use Spark related functionalities in java.

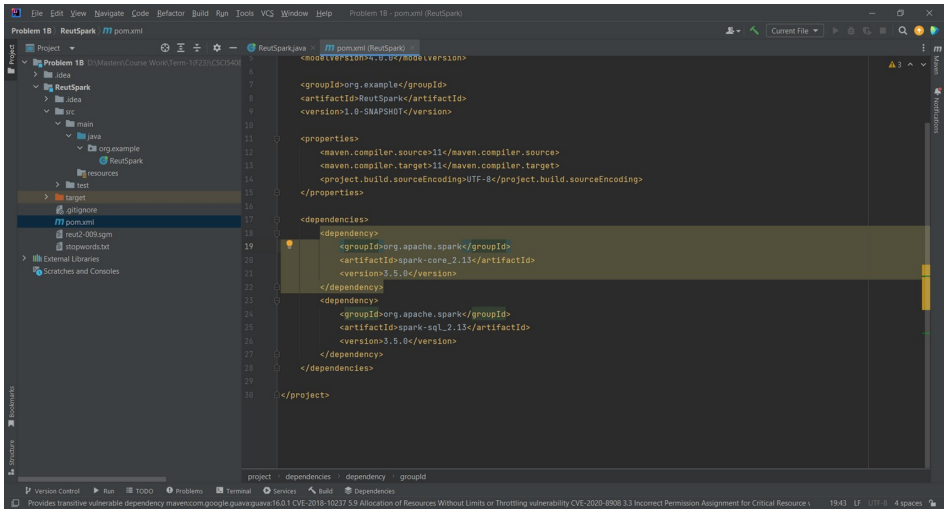


Figure 22: pom.xml file [2] [3] [4] [9] [10]

Stopwords.txt – This is a list of common stop words which are not much important and will be removed for our word frequency calculation. This file was downloaded from GitHub [8].

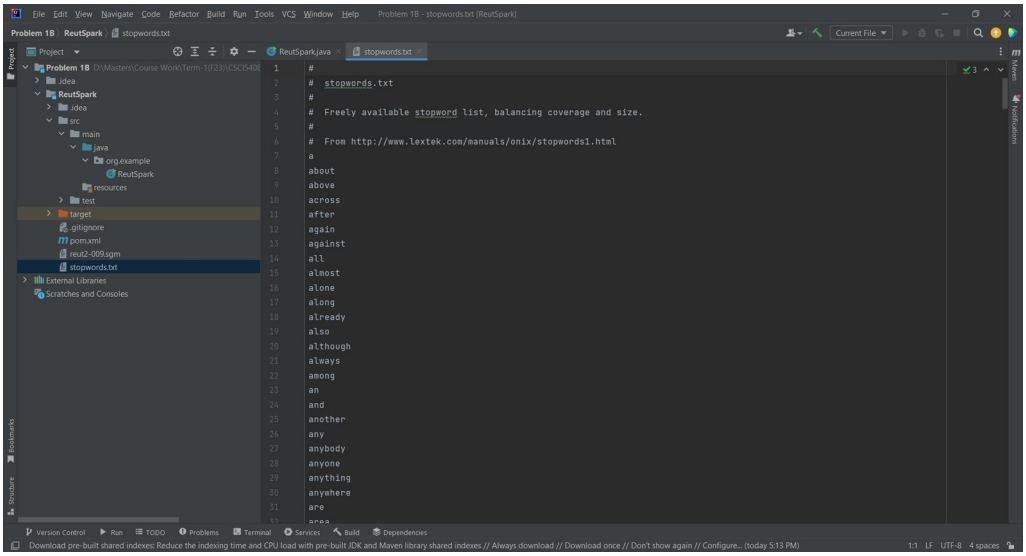


Figure 23: stopwords.txt file containing common stop words [8].

Explanation of ReutSpark.java file:

This files contains 3 methods in total, each is explained below:

1. **readStopWordsFile()** – This is just a utility method to read the stopwords.txt file contents and gives back a simple list of string stop-words.

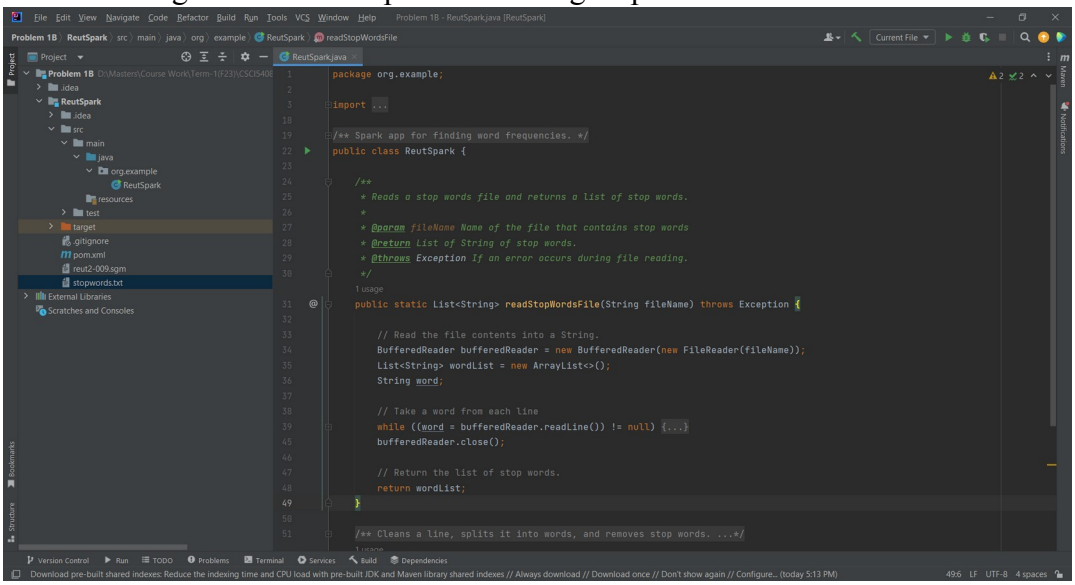


Figure 24: readStopWordsFile method logic [2] [3] [4].

2. **cleanAndSplitIntoWords()** – This method first converts all characters in uppercase, so that frequency calculation becomes case insensitive later on. After that, all the tags such as <TITLE>, </TITLE> etc are removed as they are not important for frequency calculation, otherwise such words would be most frequent, just similar to stop words. Then all the special characters are removed and then line is split into words on the basis of space. Then the list of words are filtered on the basis of stopwords.txt file content. The final list of words can now be used further in main method for frequency calculation using spark.

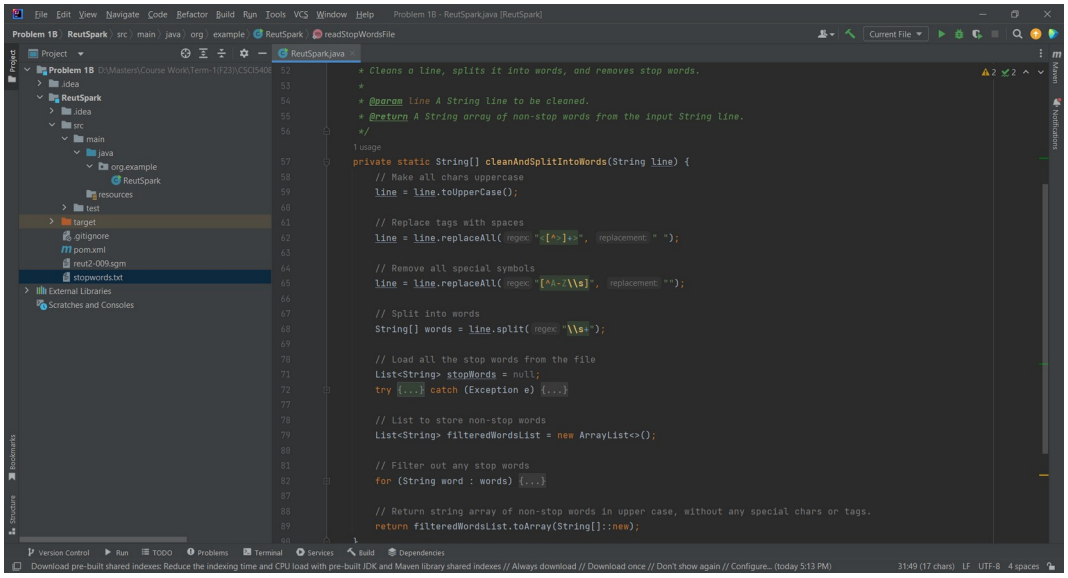
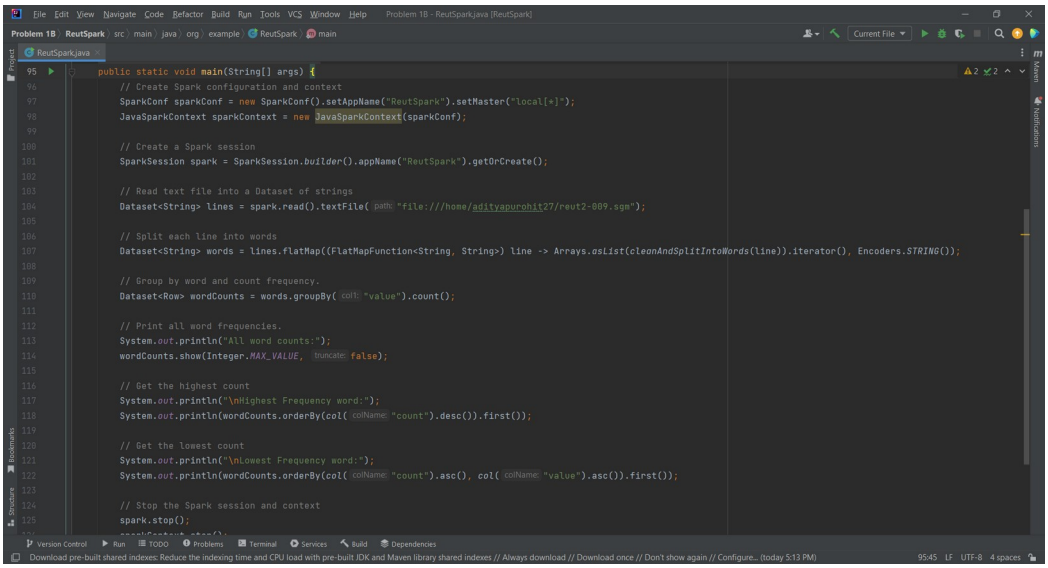


Figure 25: cleanAndSplitIntoWords method logic [2] [3] [4] [8].

3. **main()** – The spark session is created first and then we read the .sgm file in terms of spark Dataset of lines. Then we use our utility method cleanAndSplitIntoWords, to get the final set of clean words to perform word frequency calculation using spark. Then the groupBy method of spark is used to group the words by their value and get a count for each word. Then all the word frequencies are printed first, followed by highest frequency word and lowest frequency word, with the help of ascending and descending order operations on the count values of these words.

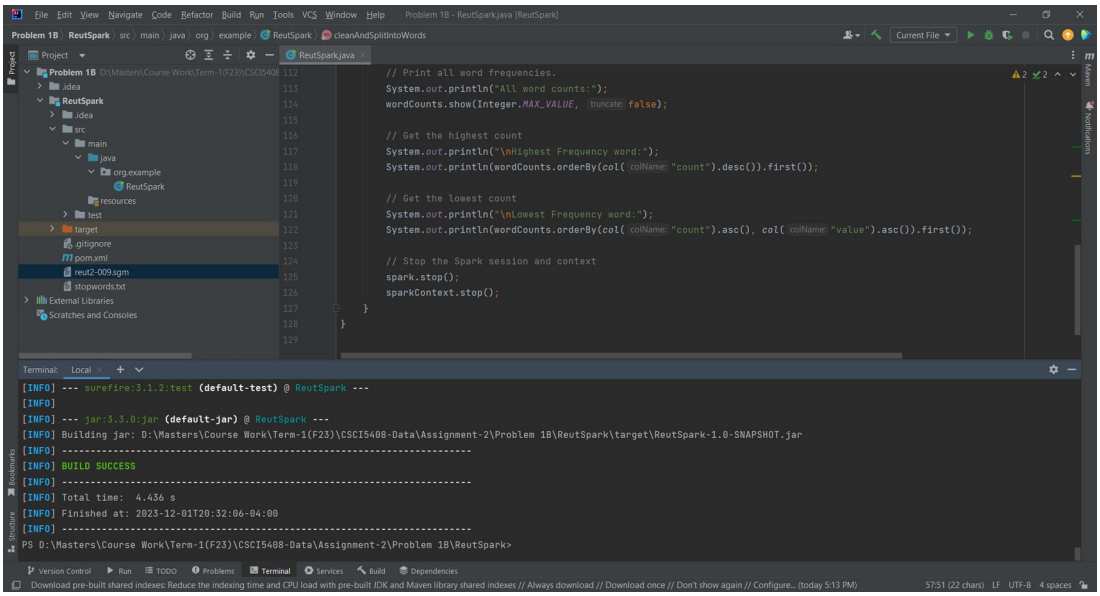


```
public static void main(String[] args) {
    // Create Spark configuration and context
    SparkConf sparkConf = new SparkConf().setAppName("ReutSpark").setMaster("local[*]");
    JavaSparkContext sparkContext = new JavaSparkContext(sparkConf);

    // Create a Spark session
    SparkSession spark = SparkSession.builder().appName("ReutSpark").getOrCreate();

    // Read text file into a Dataset of strings
    Dataset<String> lines = spark.read().textFile( path: "file:///home/adityapurohit27/reut2-009.sgm");

    // Split each line into words
    Dataset<String> words = lines.flatMap((FlatMapFunction<String, String>) line -> Arrays.asList(cleanAndSplitIntoWords(line)).iterator(), Encoders.STRING());

    // Group by word and count frequency.
    Dataset<Row> wordCounts = words.groupBy( col1: "value").count();

    // Print all word frequencies.
    System.out.println("All word counts:");
    wordCounts.show(Integer.MAX_VALUE, truncate: false);

    // Get the highest count
    System.out.println("\nHighest Frequency word:");
    System.out.println(wordCounts.orderBy(col( colName: "count").desc()).first());

    // Get the lowest count
    System.out.println("\nLowest Frequency word:");
    System.out.println(wordCounts.orderBy(col( colName: "count").asc(), col( colName: "value").asc()).first());

    // Stop the Spark session and context
    spark.stop();
```

Figure 26: main method logic [2] [3] [4] [9] [10].

Functional Testing

Step 1: Create a .jar file using “mvn clean package” command.



```
// Print all word frequencies.
System.out.println("All word counts:");
wordCounts.show(Integer.MAX_VALUE, truncate: false);

// Get the highest count
System.out.println("\nHighest Frequency word:");
System.out.println(wordCounts.orderBy(col( colName: "count").desc()).first());

// Get the lowest count
System.out.println("\nLowest Frequency word:");
System.out.println(wordCounts.orderBy(col( colName: "count").asc(), col( colName: "value").asc()).first());

// Stop the Spark session and context
spark.stop();
sparkContext.stop();
}
}
```

```
[INFO] --- surefire:3.1.2:test (default-test) @ ReutSpark ---
[INFO]
[INFO] --- jar:3.3.0:jar (default-jar) @ ReutSpark ---
[INFO] Building jar: D:\Masters\Course Work\Term-1(F23)\CSCI5408-Data\Assignment-2\Problem 1B\ReutSpark\target\ReutSpark-1.0-SNAPSHOT.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  4.436 s
[INFO] Finished at: 2023-12-01T20:32:06-04:00
[INFO] ------------------------------------------------------------------------
PS D:\Masters\Course Work\Term-1(F23)\CSCI5408-Data\Assignment-2\Problem 1B\ReutSpark>
```

Figure 27: Creating a .jar file

Step-2: Open the ssh of our master node of apache spark cluster and upload 3 files: ReutSpark-1.0-SNAPSHOT.jar, stopwords.txt and reut2-009.sgm

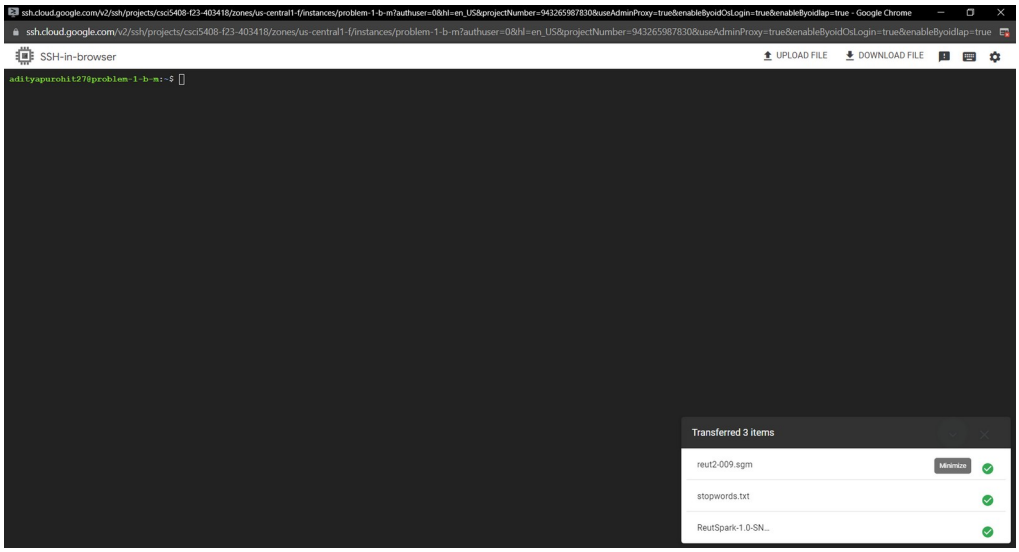


Figure 28: Upload 3 files to the master node [7].

Step-3: Execute the .jar on spark cluster using the command: "spark-submit --class org.example.ReutSpark ReutSpark-1.0-SNAPSHOT.jar"

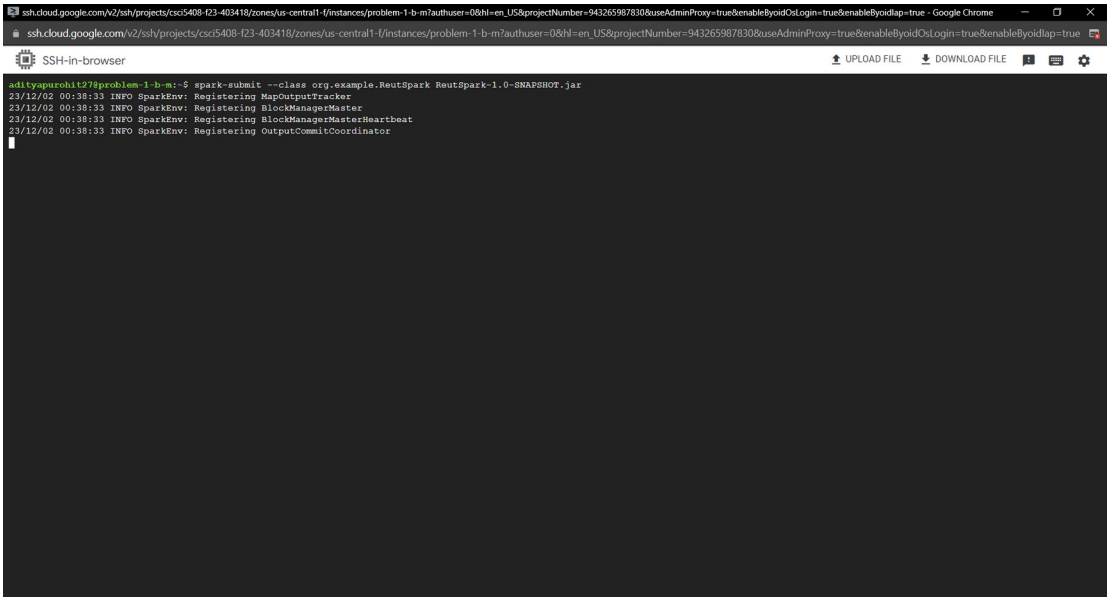


Figure 29: Execute the .jar file [7].

Figure 30: All word counts, with highest and lowest word counts [7].

Expected Output: all word counts with the highest frequency word and lowest frequency word.

Actual Output: As seen in figure 30, all the words and their counts are visible along with the highest frequency and lowest frequency word.

Highest Frequency word:

[**MLN**, 1284]

Lowest Frequency word:

[**AAA**, 1]

Flowchart: Spark framework frequency count operation

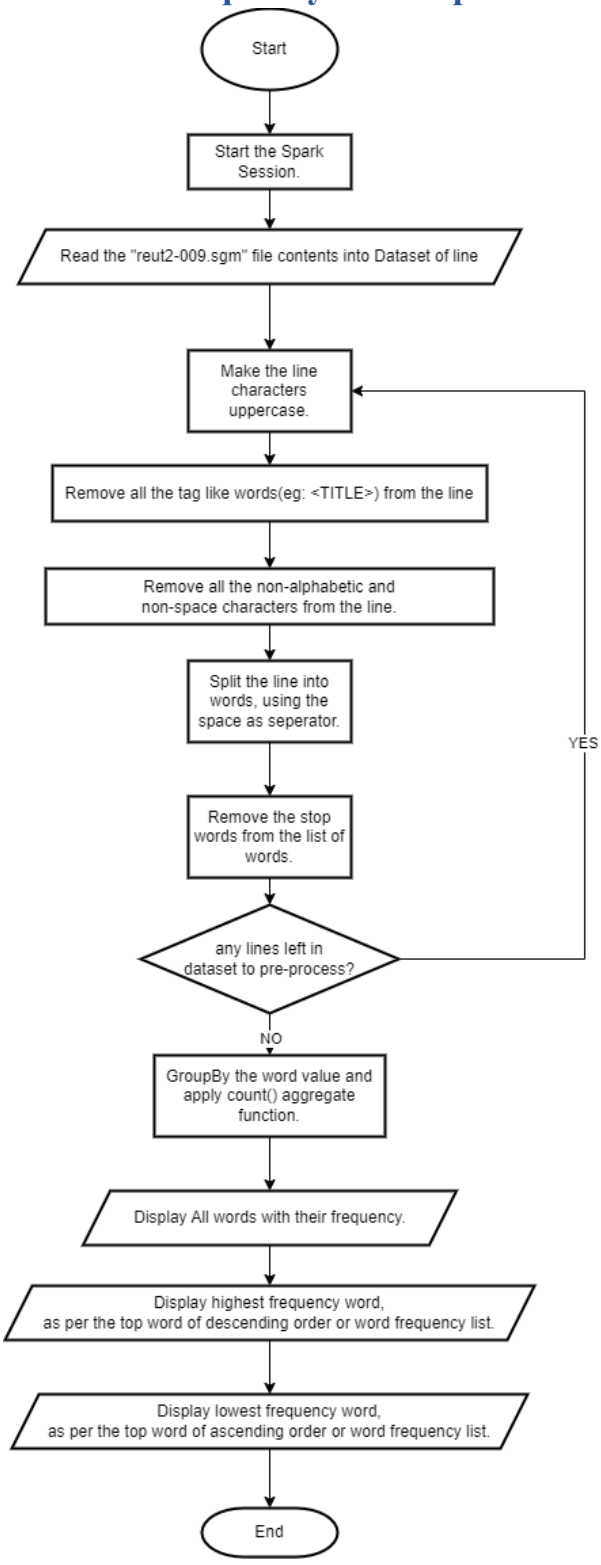


Figure 31: Flowchart of Problem-1B [6].

Problem 2: Sentiment Analysis.

Java Program

The java project structure looks something like show in figure 32. There are 6 important files.

SentimentAnalysis.java – This has all the core logic for sentiment analysis of news article titles. All parts of code are commented using Javadoc standards.

reut2-009.sqm and **reut2-014.sqm** are the input files for sentiment analysis.

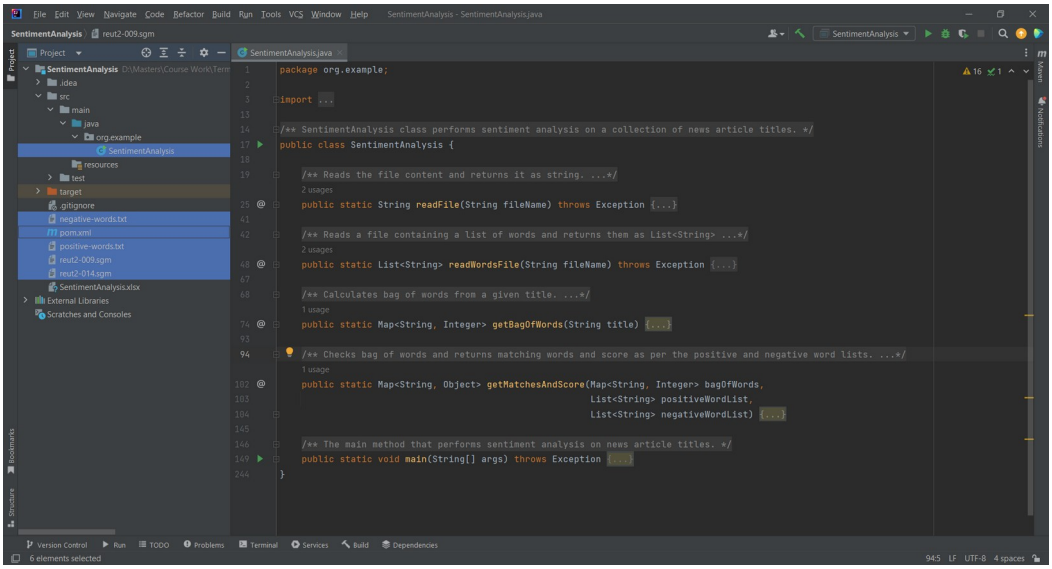


Figure 32: Overview of SentimentAnalysis.java file and folder structure of project [2] [3] [4]

Pom.xml - 1 apache dependency is used in my maven project's pom.xml in order to export the Sentiment Analysis in an Excel sheet.

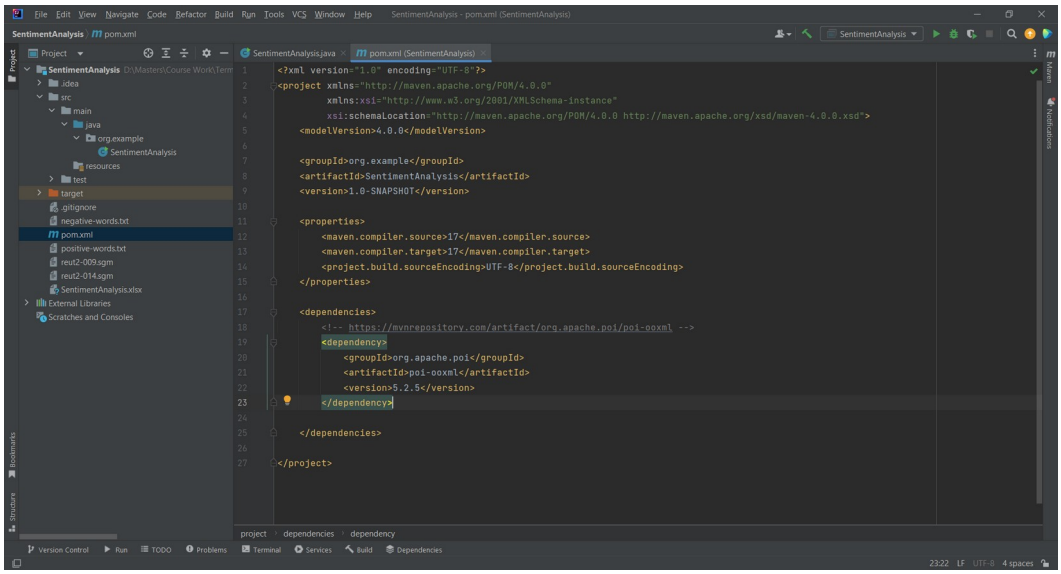


Figure 33: pom.xml file of Problem-2 [11].

positive-words.txt and **negative-words.txt**– These to files contain positive and negative word list. These are downloaded from GitHub [12] [13].

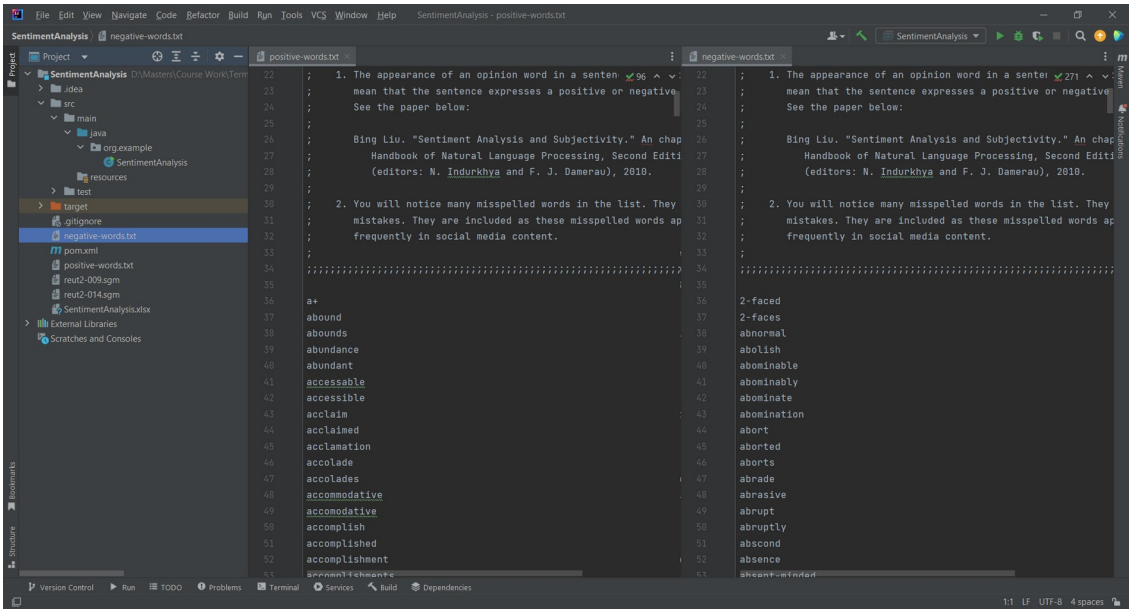


Figure 34: Positive and Negative word list [12] [13].

Explanation of SentimentAnalysis.java file:

This files contains 5 methods in total, each is explained below:

1. **readWordsFile()** – This is just a utility method to read the positive-words.txt and negative-words.txt file contents and gives back a simple list of words.

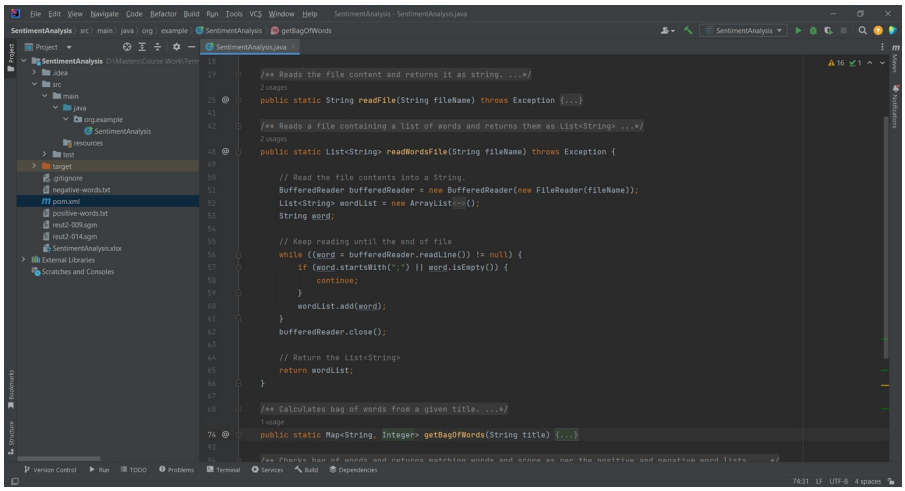


Figure 35: Logic of readWordsFile method [2] [3] [4].

2. **readFile()** – This is also a simple utility method to read the content of the .sgm files using the BufferedReader Class of java.

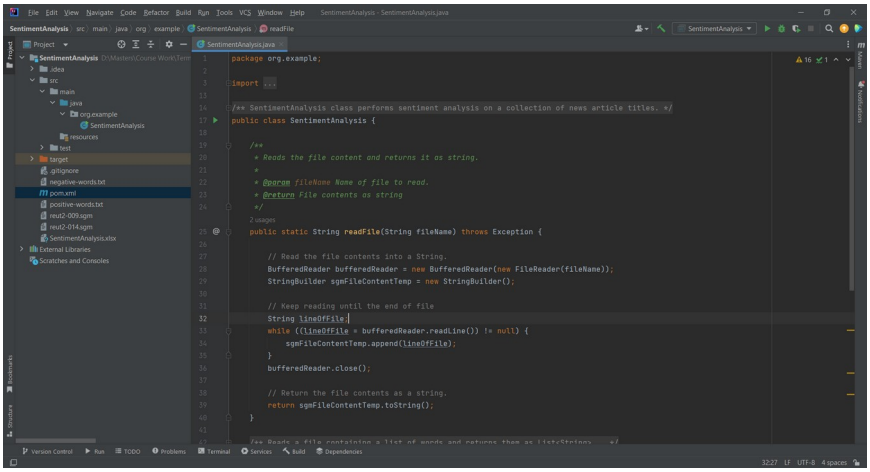


Figure 36: readFile method logic [2] [3] [4] [8].

3. **getBagOfWords()** – This method is used to create a HashMap of words and their respective counts in the title. First of all, an empty Map is created and then the title is plit

into an array of strings by the help of space separator. Then for each word, we get the current count of it in the HashMap if already exists and add a 1 to it, else we just set it 1 if it is being counted for the first time in the title. After the array is processed fully, the final HashMap is returned.

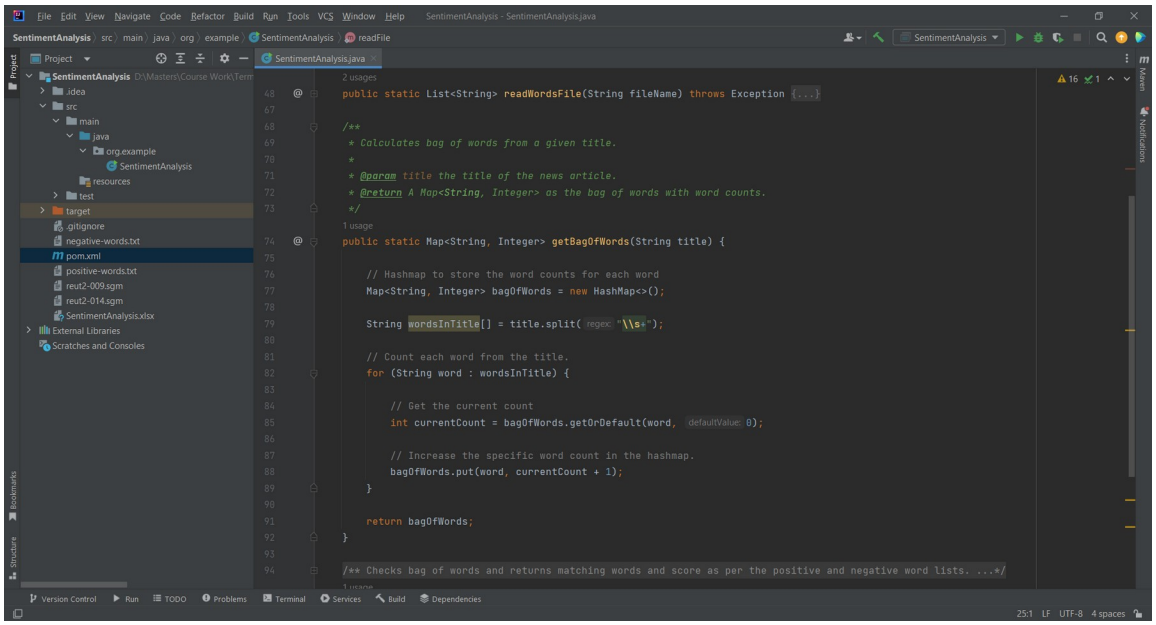


Figure 37: getBagOfWords method logic [2] [3] [4].

4. **getMatchesAndScore( )** – This method is used to calculate the matching words part of sentiment analysis and the score which can be –ve, 0 or +ve. I have first set the score to 0 and matching words as empty. Then for each word in the bagOfWords, I check it with the positive word list. If present, I add +1 to score and that word into matches list. If not, I check it with the negative word list; if a match occurs here, -1 is done in the score and word is added into matches list. But if the word is not found in any of the 2 lists, nothing is added to the score and nothing is added to the matches list for that word. In this way, whole bagOfWord is processed to get the sentiment match column and score column value.

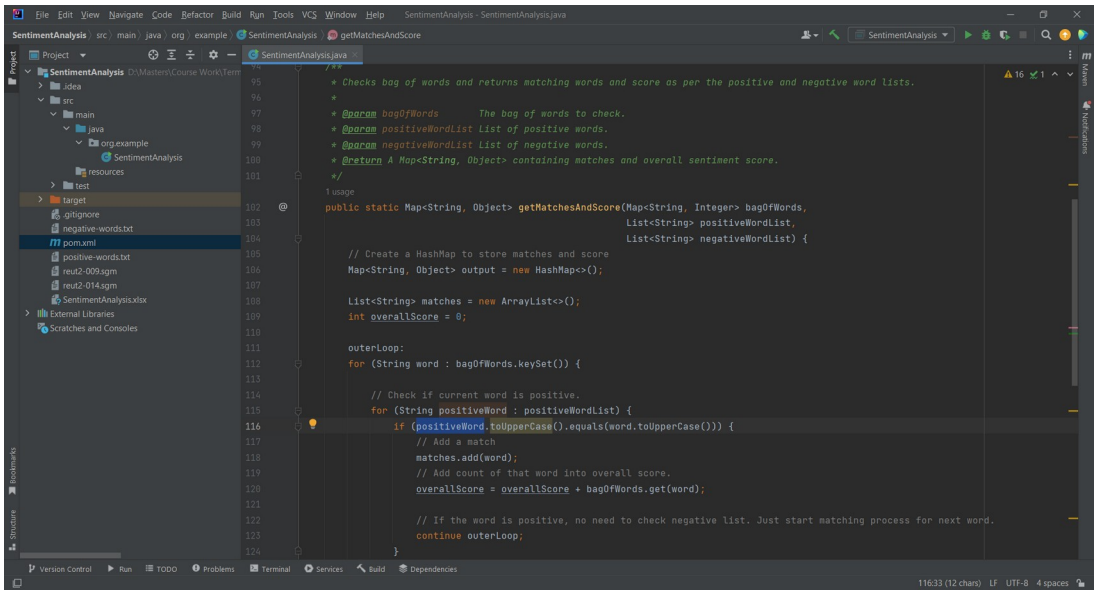


Figure 38: getMatchesAndScore method logic [2] [3] [4].

5. **main( )** – In this method, both the .sgm files are first read. The second file is appended with the first for easier processing later on. Then the lists of positive and negative words are populated from the .txt files. Then the regex is prepared for finding the content between titles. This regex is the subpart of the regex which was used in Problem 1A. Then an empty Excel sheet is created with just header row as of now. As per the regex matcher, all the titles are iterated from the top to bottom of the file contents and for each of the title, sentiment analysis is done and stored as a new row in excel sheet. Before finding bagOfWords, each title is made uppercase to make is case insensitive. Some special characters like comma, > and &LT are also removed for cleaning the title.

   The match and score values are generated with the help of getMatchesAndScore method as explained earlier. The polarity is determined as positive for +ve score, negative for –ve score value and else neutral. Along with adding analysis data to excel, it will be also printed on console. At last, the excel columns are resized and file is saved as SentimentAnalysis.xlsx

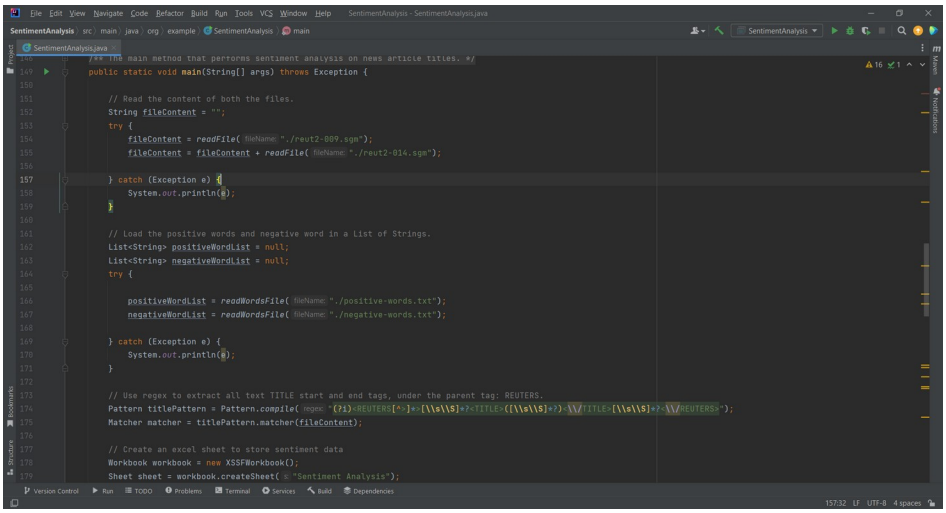


Figure 39: main() method logic [2] [3] [4].

Functional Testing

Just execute the main method of the java program. Make sure both .sgm files are in the root directory of the project.

Expected Output: Console should print the sentiment analysis for each news title and lastly create an excel sheet for the same.

Actual Output: Excel is generated along with console output of each title's sentiment analysis.

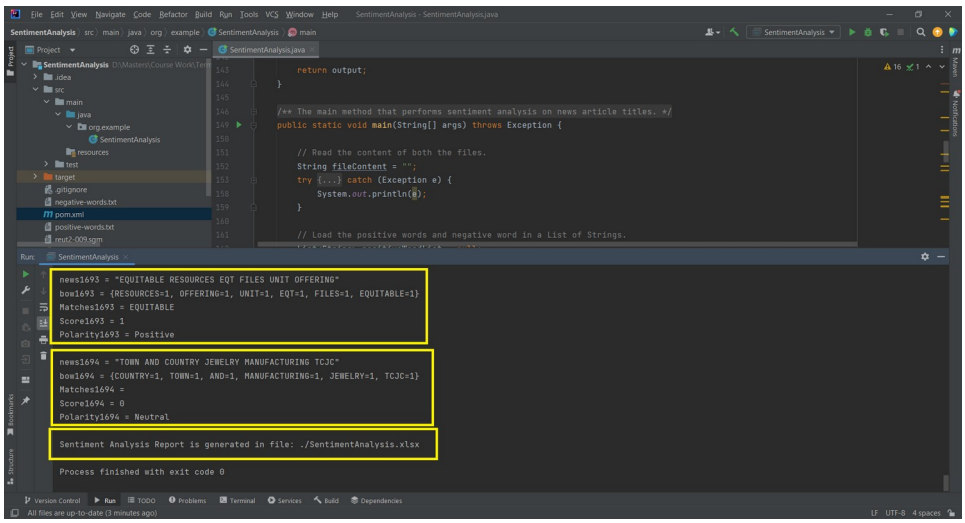
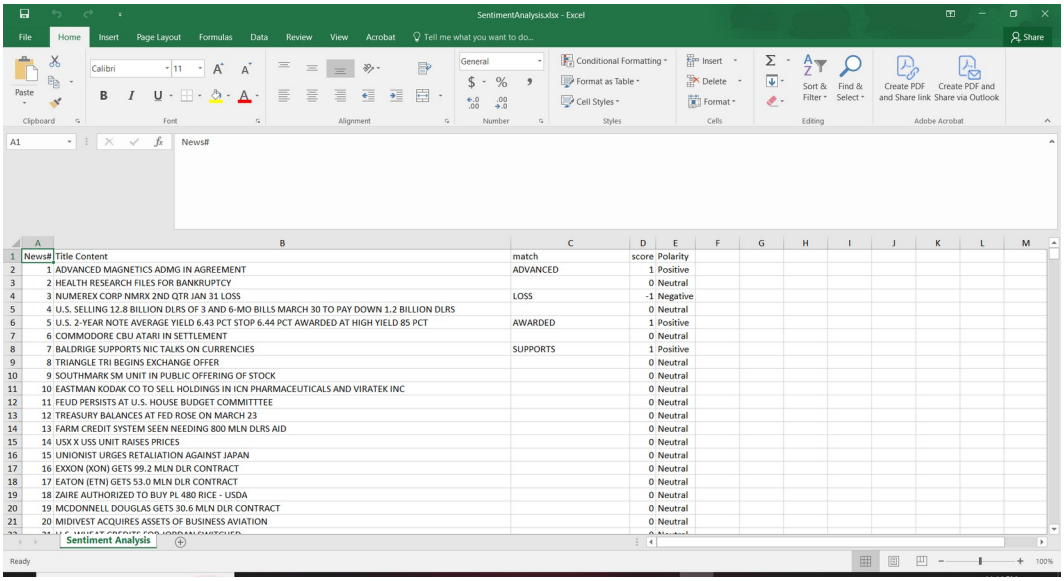


Figure 40: Execution of Sentiment Analysis program [2] [3] [4].



News#	Title Content	match	score	Polarity
1	ADVANCED MAGNETICS ADMG IN AGREEMENT	ADVANCED	1	Positive
2	HEALTH RESEARCH FILES FOR BANKRUPTCY		0	Neutral
3	NUMEREX CORP NMRX 2ND QTR JAN 31 LOSS	LOSS	-1	Negative
4	U.S. SELLING 12.8 BILLION DLRS OF 3 AND 6-MO BILLS MARCH 30 TO PAY DOWN 1.2 BILLION DLRS		0	Neutral
5	U.S. 2-YEAR NOTE AVERAGE YIELD 6.43 PCT STOP 6.44 PCT AWARDED AT HIGH YIELD 85 PCT	AWARDED	1	Positive
6	COMMODORE CBU ATARI IN SETTLEMENT		0	Neutral
7	BALDRIGE SUPPORTS NIC TALKS ON CURRENCIES	SUPPORTS	1	Positive
8	TRIANGLE TRI BEGINS EXCHANGE OFFER		0	Neutral
9	SOUTHMARK SM UNIT IN PUBLIC OFFERING OF STOCK		0	Neutral
10	EASTMAN KODAK CO TO SELL HOLDINGS IN ICN PHARMACEUTICALS AND VIRATEK INC		0	Neutral
11	FEUD PERSISTS AT U.S. HOUSE BUDGET COMMITTTEE		0	Neutral
12	TREASURY BALANCES AT FED ROSE ON MARCH 23		0	Neutral
13	FARM CREDIT SYSTEM SEEN NEEDING 800 MLN DLRS AID		0	Neutral
14	USX X USS UNIT RAISES PRICES		0	Neutral
15	UNIONIST URGES RETALIATION AGAINST JAPAN		0	Neutral
16	EXXON (XON) GETS 99.2 MLN DLR CONTRACT		0	Neutral
17	EATON (ETN) GETS 53.0 MLN DLR CONTRACT		0	Neutral
18	ZAIRE AUTHORIZED TO BUY PL 480 RICE - USDA		0	Neutral
19	MCDONNELL DOUGLAS GETS 30.6 MLN DLR CONTRACT		0	Neutral
20	MIDIVEST ACQUIRES ASSETS OF BUSINESS AVIATION		0	Neutral

Figure 41: Excel file of Sentiment analysis [11] [14].

References:

[1] "Download MongoDB Community Server," *MongoDB*. [Online]. Available: https://www.mongodb.com/try/download/community. [Accessed: 24-Nov-2023].

[2] "Java | Oracle," *Java.com*. [Online]. Available: https://www.java.com/en/. [Accessed: 25-Nov-2023].

[3] "IntelliJ IDEA – the leading Java and Kotlin IDE," *JetBrains*. [Online]. Available: https://www.jetbrains.com/idea/. [Accessed: 25-Nov-2023].

[4] B. Porter, J. van Zyl, and O. Lamy, "Welcome to Apache maven," *Apache.org*. [Online]. Available: https://maven.apache.org/. [Accessed: 25-Nov-2023].

[5] "Maven Repository: org.mongodb» mongo-java-driver» 3.12.14," *Mvnrepository.com*. [Online]. Available: https://mvnrepository.com/artifact/org.mongodb/mongo-java-driver/3.12.14. [Accessed: 25-Nov-2023].

[6] "Flowchart maker & online diagram software," *Diagrams.net*. [Online]. Available: https://app.diagrams.net/. [Accessed: 30-Nov-2023].

[7] "Dataproc," *Google Cloud*. [Online]. Available: https://cloud.google.com/dataproc?hl=en. [Accessed: 01-Dec-2023].

[8] larsyencken, "Stopwords.Txt," *GitHub*. [Online]. Available: https://gist.github.com/larsyencken/1440509. [Accessed: 01-Dec-2023].

[9] "spark-core," *Mvnrepository.com*. [Online]. Available: https://mvnrepository.com/artifact/org.apache.spark/spark-core. [Accessed: 01-Dec-2023].

[10] "spark-sql," *Mvnrepository.com*. [Online]. Available: https://mvnrepository.com/artifact/org.apache.spark/spark-sql. [Accessed: 01-Dec-2023].

[11] "Apache poi-ooxml," *Mvnrepository.com*. [Online]. Available: https://mvnrepository.com/artifact/org.apache.poi/poi-ooxml/5.2.5. [Accessed: 01-Dec-2023].

[12] Marcin, "positive-words.txt," *GitHub*. [Online]. Available: https://gist.github.com/mkulakowski2/4289437. [Accessed: 01-Dec-2023].

[13] Marcin, "negative-words.txt," *GitHub*. [Online]. Available: https://gist.github.com/mkulakowski2/4289441. [Accessed: 01-Dec-2023].

[14] “Microsoft Excel,” *Microsoft.com*. [Online]. Available: https://www.microsoft.com/en-ca/microsoft-365/excel. [Accessed: 01-Dec-2023].