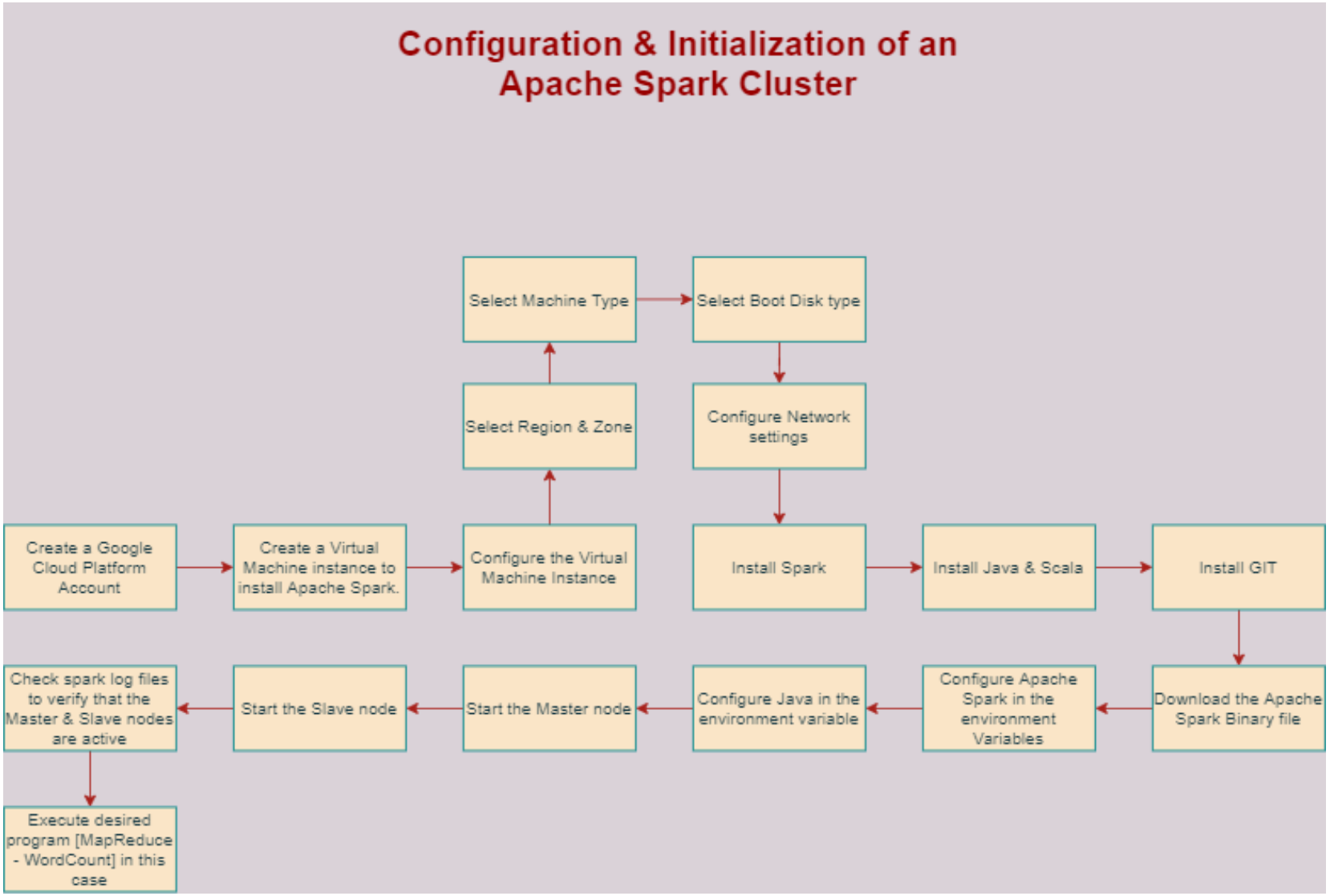
CSCI 5408

Assignment 4

Benny Daniel Tharigopala
Banner#: B00899629
Net Id: bn489600

URL to GitLab Repository: https://git.cs.dal.ca/benny/csci-5408-w2022-b00899629-benny_daniel

Problem 1 - TASK 1: Apache Spark Cluster Configuration



Problem 1 - TASK 2: Step 2 (Twitter search & stream APIs)

Search API:

The primary purpose of the Twitter search API is to retrieve tweets which contain words that match a specific set of keywords or "query". The API extracts tweets in JSON (JavaScript Object Notation) format and contains metadata in addition to the tweet. These metadata fields are in the form of key-value pairs. The API submits synchronous or asynchronous queries to the Twitter server to obtain the required tweets. The queries contain parameters out of which "query" is mandatory. This key has values which correspond to the keywords that tweets are matched against. Other parameters serve as attributes that describe a tweet. These parameters include, but are not limited to,

"geocode" – used to determine the latitude and longitude of tweet authors,

"lang" – used to determine the language of the text in a tweet,

"result_type" – used to specify the type of search results retrieved, and

"until" – used to filter tweets by restricting the tweet creation date within a specific date.

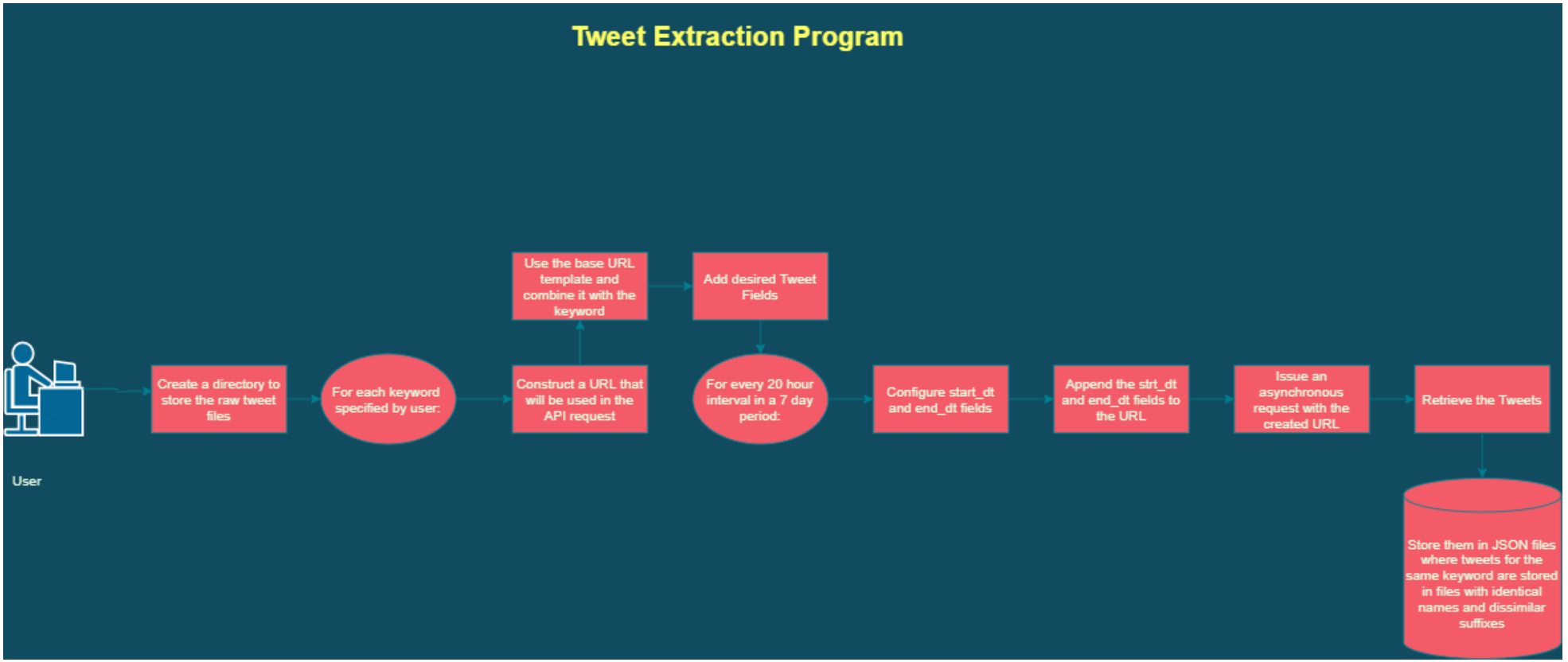
Tweet.fields:

The Tweet fields parameter provides a higher degree of customization by enabling developers to only request the fields that they require depending on their use cases [2]. Common Tweet fields include – "text", "author_id", "created_at", "geo", "lang" and "referenced_tweets".

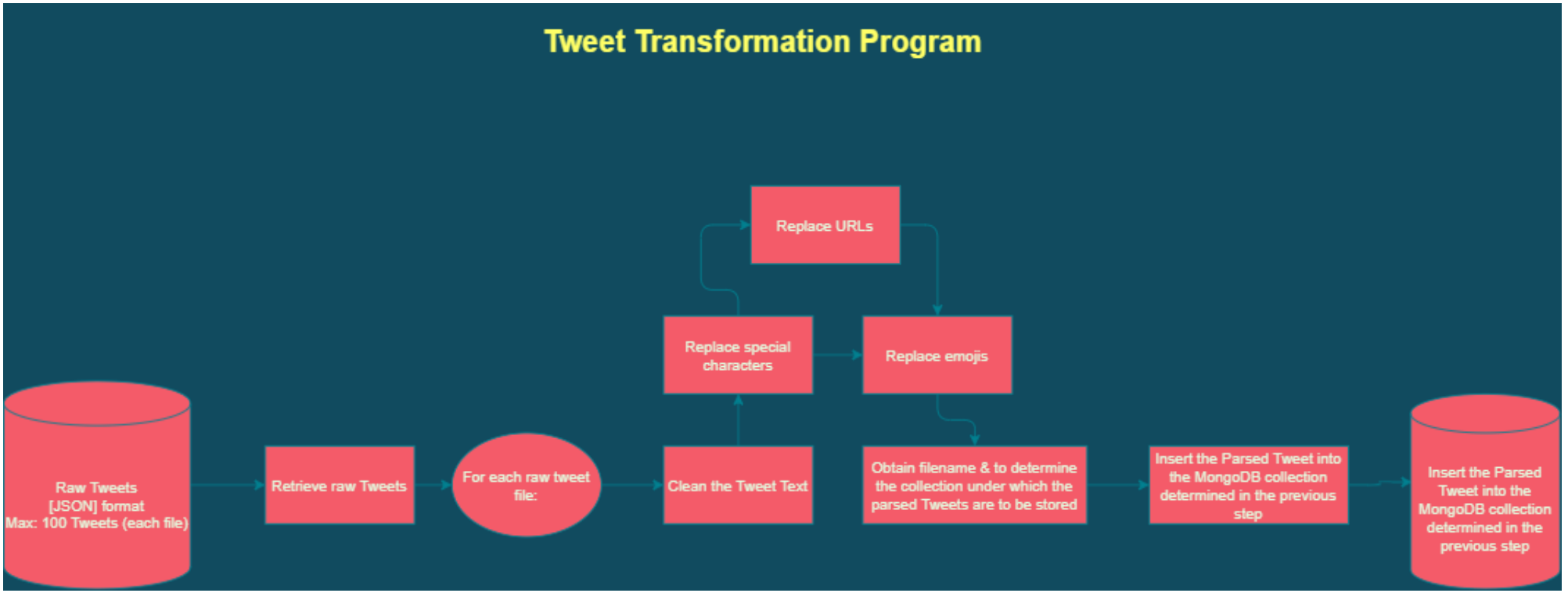
Stream API:

The stream API is similar to the search API but differs in terms of the tweet creation date. The stream API generates real time tweets from the server and is particularly useful when developers require information about current trends or if they want to be notified about a topic of interest. The API makes use of "filtered stream endpoints" to filter the tweets through rules by using operators to build the rule. Subsequently, a POST request is issued to the filtered stream rules endpoint with an added payload as an array of rules and operators.

Problem 1 - TASK 2: Tweet Extraction



Problem 1 - TASK 2: Tweet Transformation



Problem 1 - TASK 2 – Snips

Configuration.java

This file contains the Private variables – "bearerToken" and "mongoURL" for retrieving tweets via the Twitter API and storing the Tweets into MongoDB respectively.

ExtractionEngine.java

This file retrieves tweets and stores them in JSON files, such that each files contains a maximum of 100 Tweets.

1. Retrieve Tweets:

   I have retrieved tweets using a "Scheduler module" such that Tweets are retrieved in **20-hour intervals.**

```
private static void getTweets(String keyword, String bearerToken) throws IOException, URISyntaxException, InterruptedException
{

//    https://api.twitter.com/2/tweets/search/recent?query=&tweet.fields=author_id,created_at,geo,lang,source,referenced_tweets //URL generated through Postman
    String url  = "https://api.twitter.com/2/tweets/search/recent?query=" + keyword + "&tweet.fields=author_id,created_at,geo,lang,source,referenced_tweets" + "&max_results=100";
    String fileName = null;

    int fileNumber=0;
    DateFormat df = new SimpleDateFormat( pattern: "yyyy-MM-dd'T'HH:mm:ss'Z'");
    for(int scheduler=20;scheduler<144;scheduler+=20) //Iterates through 12 hour interval for 7 days
    {
        fileNumber++;
        //method - submit request
        Calendar start = Calendar.getInstance();
        start.add(Calendar.HOUR,  amount: -scheduler - 20);
        Date strtDt = start.getTime();
        String startDate = df.format(strtDt);

        Calendar end = Calendar.getInstance();
        end.add(Calendar.HOUR, -scheduler);
        Date endDt = end.getTime();
        String endDate = df.format(endDt);
```

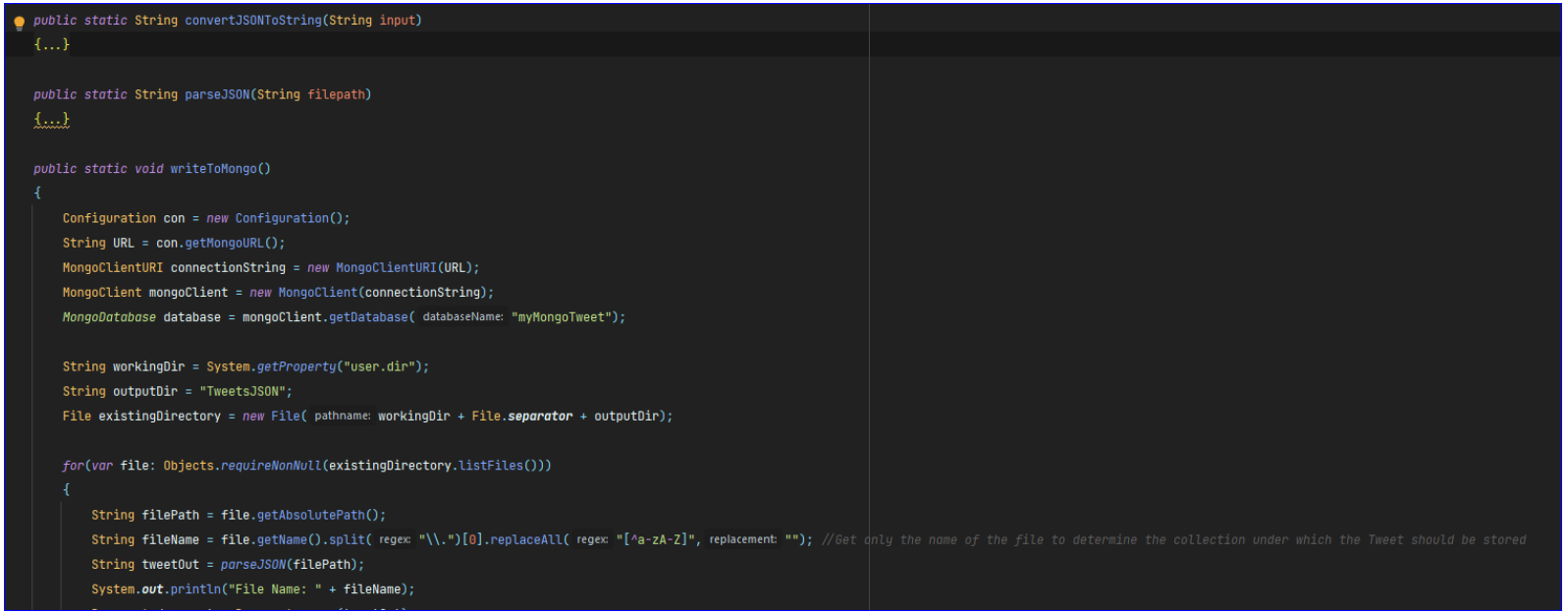
2. Create a dedicated directory inside the working directory to store the retrieved tweets.
3. Store the tweets in JSON files.

```java
public static void createDirectory()
{...}
public static void generateJSONFile(String tweet, String fileName)
{
    BufferedWriter writer = null;
    try {
        String workingDir = System.getProperty("user.dir");
        String outputDir = "TweetsJSON";
        File outputFile = new File( pathname: workingDir + File.separator + outputDir + File.separator + fileName + ".json"); //Store the output JSON files in a dedicated directory

        FileWriter fileWriter = new FileWriter(outputFile);
        writer = new BufferedWriter(fileWriter);
        writer.write(tweet);
        System.out.println("Tweets written to " + fileName + "!");
    }  catch (IOException e) {
        e.printStackTrace();
    }
    finally
```

FiltrationEngine.java

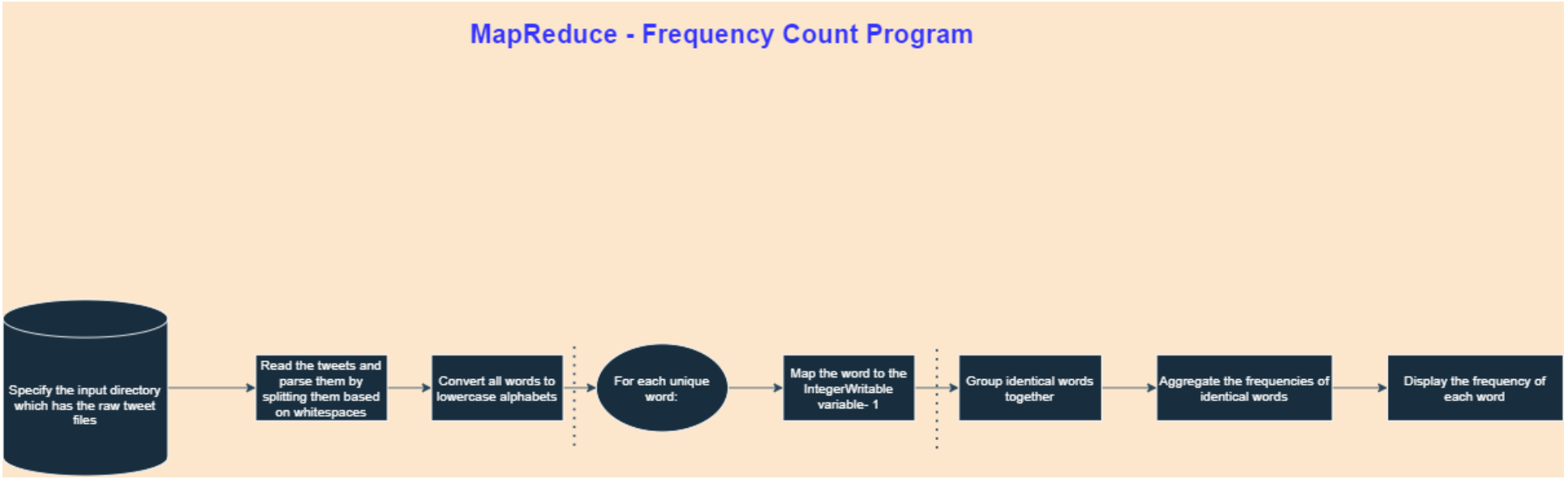
4. Read files by converting JSON files to Strings.
5. Parse the JSON files. Remove Unicode characters, emojis and URLs.
6. Write the files to MongoDB.

   While writing the parsed tweets to MongoDB, the names of the JSON files are used to determine the collection under which the tweets are to be stored.



```java
public static String convertJSONToString(String input)
{...}

public static String parseJSON(String filepath)
{...}

public static void writeToMongo()
{
    Configuration con = new Configuration();
    String URL = con.getMongoURL();
    MongoClientURI connectionString = new MongoClientURI(URL);
    MongoClient mongoClient = new MongoClient(connectionString);
    MongoDatabase database = mongoClient.getDatabase( databaseName: "myMongoTweet");

    String workingDir = System.getProperty("user.dir");
    String outputDir = "TweetsJSON";
    File existingDirectory = new File( pathname: workingDir + File.separator + outputDir);

    for(var file: Objects.requireNonNull(existingDirectory.listFiles()))
    {
        String filePath = file.getAbsolutePath();
        String fileName = file.getName().split( regex: "\\.")[0].replaceAll( regex: "[^a-zA-Z]", replacement: ""); //Get only the name of the file to determine the collection under which the Tweet should be stored
        String tweetOut = parseJSON(filePath);
        System.out.println("File Name: " + fileName);
```

Problem 2 - TASK 1: MapReduce Program for Frequency Count



```
benny28dany@data-assignment3:~$ start-master.sh
starting org.apache.spark.deploy.master.Master, logging to /opt/spark/logs/spark-benny28dany-org.apache.spark.deplo
y.master.Master-1-data-assignment3.out
benny28dany@data-assignment3:~$ start-slave.sh spark://data-assignment3.us-central1-a.c.a2bn489600.internal:7077
starting org.apache.spark.deploy.worker.Worker, logging to /opt/spark/logs/spark-benny28dany-org.apache.spark.deplo
y.worker.Worker-1-data-assignment3.out
benny28dany@data-assignment3:~$ tail /opt/spark/logs/spark-benny28dany-org.apache.spark.deploy.worker.Worker-1-data
-assignment3.out
22/03/15 01:53:42 INFO Worker: Spark home: /opt/spark
22/03/15 01:53:42 INFO ResourceUtils: ==============================================================
22/03/15 01:53:42 INFO ResourceUtils: Resources for spark.worker:

22/03/15 01:53:42 INFO ResourceUtils: ==============================================================
22/03/15 01:53:42 INFO Utils: Successfully started service 'WorkerUI' on port 8081.
22/03/15 01:53:42 INFO WorkerWebUI: Bound WorkerWebUI to 0.0.0.0, and started at http://data-assignment3.us-central
1-a.c.a2bn489600.internal:8081
22/03/15 01:53:42 INFO Worker: Connecting to master data-assignment3.us-central1-a.c.a2bn489600.internal:7077...
22/03/15 01:53:42 INFO TransportClientFactory: Successfully created connection to data-assignment3.us-central1-a.c.
a2bn489600.internal/10.128.0.3:7077 after 116 ms (0 ms spent in bootstraps)
22/03/15 01:53:43 INFO Worker: Successfully registered with master spark://data-assignment3.us-central1-a.c.a2bn489
600.internal:7077
```

Spark Master at spark://data-assignment3.us-central1-a.c.a2bn489600.internal:7077

URL: spark://data-assignment3.us-central1-a.c.a2bn489600.internal:7077
Alive Workers: 1
Cores in use: 2 Total, 0 Used
Memory in use: 2.8 GiB Total, 0.0 B Used
Resources in use:
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (1)

Worker Id	Address	State	Cores	Memory
worker-20220315015341-10.128.0.3-44719	10.128.0.3:44719	ALIVE	2 (0 Used)	2.8 GiB (0.0 B Used)

Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User

Completed Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User

Frequency Counts:

1. Flu: 281
2. Snow: 378
3. Cold: 242

Snow had the highest number of occurrences whereas Cold, relatively, had the lowest number of occurrences.

Problem 2 – Task 2

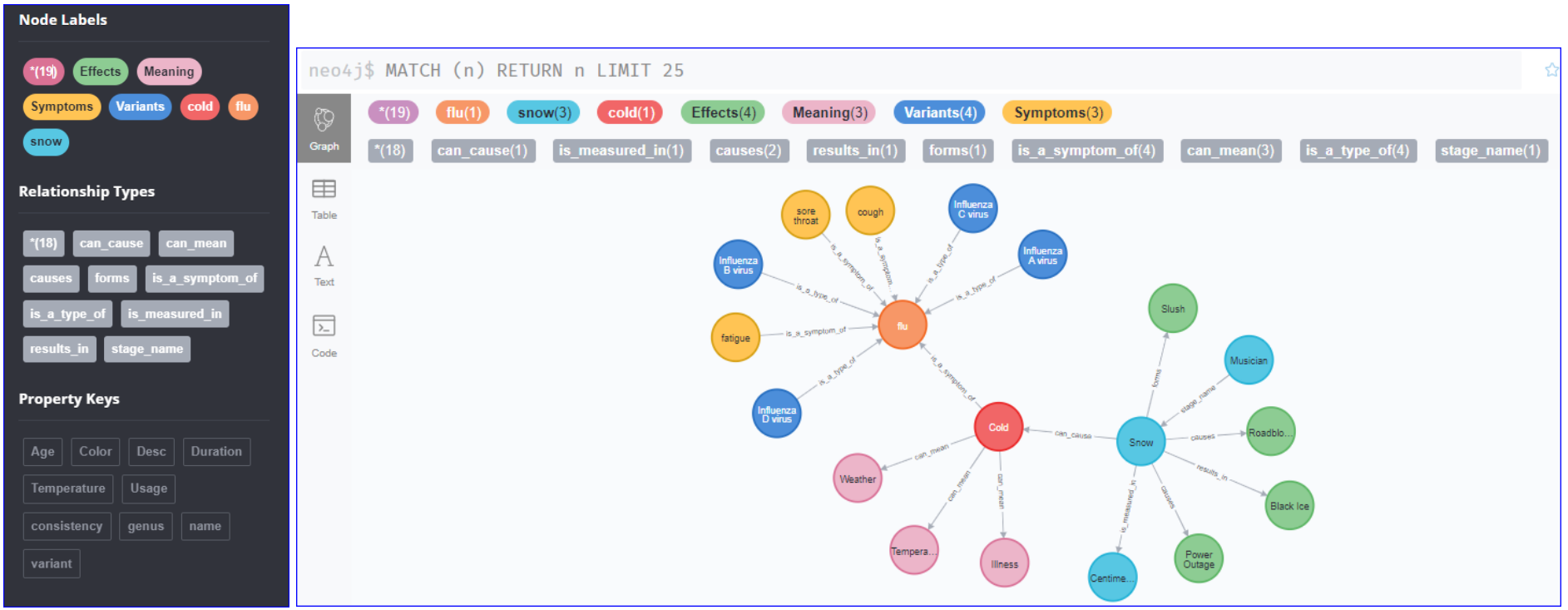
Cypher Queries:

1. CREATE (s:Effects {name: 'Slush', Color: 'Brown'})
2. CREATE (s:Effects {name: 'Black Ice', Color: 'Black'})
3. CREATE (s:Effects {name: 'Power Outage', Duration: '15 Hours'})
4. CREATE (s:Effects {name: 'Roadblock', Duration: '12 Hours'})
5. CREATE (s:Meaning {name: 'Weather', Usage: 'Used as an adjective to describe the weather'})
6. CREATE (s:Meaning {name: 'Temperature', Usage: 'Used as an adjective to describe the nature of an object'})
7. CREATE (s:Meaning {name: 'Illness', Usage: 'A common viral infection'})
8. CREATE (f:Variants {name: 'Influenza A virus', genus: 'Alphainfluenzavirus'})
9. CREATE (f:Variants {name: 'Influenza B virus', genus: 'Betainfluenzavirus'})r
10. CREATE (f:Variants {name: 'Influenza C virus', genus: 'Gammainfluenzavirus'})
11. CREATE (f:Variants {name: 'Influenza D virus', genus: 'Deltainfluenzavirus'})
12. CREATE (f:Symptoms {name: 'cough'})
13. CREATE (f:Symptoms {name: 'sore throat'})
14. CREATE (f:Symptoms {name: ' fatigue'})

15. CREATE (s:snow {name: 'Centimetres', Desc:''})
16. CREATE (s:snow {name: 'Musician', Desc:"Darrin Kenneth O'Brien"})
17. MATCH (s:Effects {name: 'Slush'})
    MATCH (sn:snow {name: 'Snow'})
    CREATE (sn)-[rel:forms]->(s)
18. MATCH (s:Effects {name: 'Black Ice'})
    MATCH (sn:snow {name: 'Snow'})
    CREATE (sn)-[rel:results_in]->(s)
19. MATCH (s:Effects {name: 'Power Outage'})
    MATCH (sn:snow {name: 'Snow'})
    CREATE (sn)-[rel:causes]->(s)
20. MATCH (s:Effects {name: 'Roadblock'})
    MATCH (sn:snow {name: 'Snow'})
    CREATE (sn)-[rel:causes]->(s)
21. MATCH (s:Meaning {name: 'Weather'}) MATCH (sn:cold {name: 'Cold'}) CREATE (sn)-[rel:can_mean]->(s)
22. MATCH (s:Meaning {name: 'Temperature'}) MATCH (sn:cold {name: 'Cold'}) CREATE (sn)-[rel:can_mean]->(s)
23. MATCH (s:Meaning {name: 'Illness'}) MATCH (sn:cold {name: 'Cold'}) CREATE (sn)-[rel:can_mean]->(s)
24. MATCH (s:Symptoms {name: 'fatigue'}) MATCH (sn:flu {name: 'flu'}) CREATE (s)-[rel:is_a_symptom_of]->(sn)
25. MATCH (s:Symptoms {name: sore throat}) MATCH (sn:flu {name: 'flu'}) CREATE (s)-[rel:is_a_symptom_of]->(sn)
26. MATCH (s:Symptoms {name: cough}) MATCH (sn:flu {name: 'flu'}) CREATE (s)-[rel:is_a_symptom_of]->(sn)
27. MATCH (s:Variants {name: 'Influenza A virus'}) MATCH (sn:flu {name: 'flu'}) CREATE (s)-[rel:is_a_type_of]->(sn)
28. MATCH (s:Variants {name: 'Influenza B virus'}) MATCH (sn:flu {name: 'flu'}) CREATE (s)-[rel:is_a_type_of]->(sn)
29. MATCH (s:Variants {name: 'Influenza C virus'}) MATCH (sn:flu {name: 'flu'}) CREATE (s)-[rel:is_a_type_of]->(sn)

30. MATCH (s:Variants {name: 'Influenza D virus'}) MATCH (sn:flu {name: 'flu'}) CREATE (s)-[rel:is_a_type_of]->(sn)
31. MATCH (s:snow {name: 'Musician'}) MATCH (sn:snow {name: 'Snow'}) CREATE (s)-[rel:stage_name]->(sn)
32. MATCH (s:snow {name: 'Centimetres'}) MATCH (sn:snow {name: 'Snow'}) CREATE (sn)-[rel:is_measured_in]->(s)
33. MATCH (c:cold {name: 'Cold'}) MATCH (sn:snow {name: 'Snow'}) CREATE (sn)-[rel:can_cause]->(c)
34. MATCH (c:cold {name: 'Cold'}) MATCH (f:flu {name: 'flu'}) CREATE (c)-[rel:is_a_symptom_of]->(f)

Neo4j Graph:



References

[1] "Mapreduce tutorial," *Apache Hadoop 3.3.2 – MapReduce Tutorial*. [Online]. Available: https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. [Accessed: 10-Mar-2022].

[2] "Fields," *developer.twitter.com*. [Online]. Available: https://developer.twitter.com/en/docs/twitter-api/fields. [Accessed: 09-Mar-2022]

[3] "Tweet object," *developer.twitter.com*. [Online]. Available: https://developer.twitter.com/en/docs/twitter-api/data-dictionary/object-model/tweet. [Accessed: 08-Mar-2022]

[4] "Tweets lookup introduction," *developer.twitter.com*. [Online]. Available: https://developer.twitter.com/en/docs/twitter-api/tweets/lookup/introduction. [Accessed: 08-Mar-2022]

[5] "Stream Tweets in real-time," *developer.twitter.com*. [Online]. Available: https://developer.twitter.com/en/docs/tutorials/stream-tweets-in-real-time. [Accessed: 07-Mar-2022]

[6] "Changing Java Date one hour back," *Stack Overflow*. [Online]. Available: https://stackoverflow.com/questions/8833399/changing-java-date-one-hour-back. [Accessed: 08-Mar-2022]

[7] "Word Count Program With MapReduce and Java - DZone Big Data," *dzone.com*. [Online]. Available: https://dzone.com/articles/word-count-hello-word-program-in-mapreduce. [Accessed: 09-Mar-2022]

[8] "Hadoop - MapReduce - Tutorialspoint," *www.tutorialspoint.com*. [Online]. Available: https://www.tutorialspoint.com/hadoop/hadoop_mapreduce.htm#:~:text=MapReduce%20is%20a%20processing%20technique. [Accessed: 07-Mar-2022]

[9] D. Taylor, "Hadoop & Mapreduce Examples: Create First Program in Java," *www.guru99.com*, 31-Jan-2020. [Online]. Available: https://www.guru99.com/create-your-first-hadoop-program.html. [Accessed: 08-Mar-2022]

[10] "MapReduce Tutorial - javatpoint," *www.javatpoint.com*. [Online]. Available: https://www.javatpoint.com/mapreduce. [Accessed: 11-Mar-2022]

[11] "Apache Hadoop 3.2.1 – MapReduce Tutorial," *hadoop.apache.org*. [Online]. Available: https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. [Accessed: 12-Mar-2022]

[12] “CREATE - Neo4j Cypher Manual,” *Neo4j Graph Database Platform*. [Online]. Available: https://neo4j.com/docs/cypher-manual/current/clauses/create/#create-create-multiple-nodes. [Accessed: 11-Mar-2022]

[13] “regex - Removing the url from text using java,” *Stack Overflow*. [Online]. Available: https://stackoverflow.com/questions/12366496/removing-the-url-from-text-using-java. [Accessed: 10-Mar-2022]

[14] “regex - Replace unicode matches in javascript,” *Stack Overflow*. [Online]. Available: https://stackoverflow.com/questions/14613080/replace-unicode-matches-in-javascript. [Accessed: 06-Mar-2022]

[15] baeldung, “Remove Emojis from a Java String | Baeldung,” *www.baeldung.com*, 05-Sep-2018. [Online]. Available: https://www.baeldung.com/java-string-remove-emojis. [Accessed: 05-Mar-2022]

[16] “Regex to match all emoji - Regex Tester/Debugger,” *www.regextester.com*. [Online]. Available: https://www.regextester.com/106421. [Accessed: 07-Mar-2022]

[17] baeldung, “Exploring the New HTTP Client in Java | Baeldung,” *www.baeldung.com*, 11-Feb-2018. [Online]. Available: https://www.baeldung.com/java-9-http-client. [Accessed: 10-Mar-2022]