**REAL TIME ANOMALY MONITORING SYSTEM**

**A PROJECT REPORT**

***Submitted by***

KEERTHINI M. (922115104019)

KOKILA K. (922115104020)

***in partial fulfilment for the award of the degree***

***of***

**BACHELOR OF ENGINEERING**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

**SSM INSTITUTE OF ENGINEERING AND TECHNOLOGY**

**DINDIGUL**

**ANNA UNIVERSITY::CHENNAI 600025**

**MARCH 2019**

**ANNA UNIVERSITY::CHENNAI 600025**

**BONAFIDE CERTIFICATE**

Certified that this project report **“REAL TIME ANOMALY MONITORING** **SYSTEM “**is the bonafide work of **“KEERTHINI M.(922115104019), KOKILA** **K.(922115104020)”** who carried out the project work under my supervision.

**SIGNATURE SIGNATURE**

**Dr. V. Shunmugavel M.E.,Ph.D., Ms. V. Nivedhitha M.E.,**

**HEAD OF THE DEPARTMENT SUPERVISOR**

**PROFESSOR ASSISTANT PROFESSOR**

DEPARTMENT OF CSE DEPARTMENT OF CSE

SSM INSTITUTE OF ENGINEERING SSM INSTITUTE OF ENGINEERING

AND TECHNLOGY AND TECHNOLOGY

DINDIGUL-624 002 DINDIGUL-624 002

Submitted for the University examination on \_\_\_\_\_\_\_\_\_\_\_\_\_\_

**INTERNAL EXAMINER EXTERNAL EXAMINER**

**TABLE OF CONTENTS**

**CHAPTER NO. TITLE PAGE NO.**

**ABSTRACT i**

**1. INTRODUCTION**  1

1.1 Existing System 2

1.1.1 Drawbacks 2

1.2 Proposed System 3

1.2.1 Advantages 3

**2. LITERATURE REVIEW** 4

**3. REQUIREMENT AND ANALYSIS**  9

3.1 Feasibility study 9

3.1.1 Technical feasibility 9 3.1.2 Economical feasibility 10

3.1.3 Social feasibility 10

3.1.4 Legal feasibility 10

3.1.5 Scheduling feasibility 10 3.2 Hardware Requirements 11

3.3 Software Requirements 11

3.4 Functional Requirements 11

3.5 Non-Functional Requirements 12

**4. DESIGN**  13

**5. CODING**  19

5.1 Login 19

5.2 Home page

5.2.1 Search

5.2.1.1 Hive client call

5.2.1.2 HDFS read

5.2.1.3 Data collection

5.2.1.4 JSON

5.2.1.5 Twitter call

5.2.1.6 Search results

5.2.2 Search storage

5.3 Reports

5.3.1 Location

5.3.2 Hash tag

5.3.3 Retweet count

5.3.4 Result

**6. IMPLEMENTATION**

6.1 Login module

6.2 Home page module

6.3 Report module

6.4 Screenshots

**7. TESTING**

7.1 System testing

7.1.1 System objectives

7.1.2 Types of testing

7.2 Unit testing

7.3 Validation testing

**8. CONCLUSION AND FUTURE WORK**

**9. REFERENCES**

**ABSTRACT**

Microblog platforms have been extremely popular in the big data era due to its real-time diffusion of information. It’s important to know what anomalous events are trending on the social network and be able to monitor their evolution and find related anomalies. In this project, it analyses the data of the social media “TWITTER”. It periodically monitors the data fed by the users in the twitter. This project analyses and gives the result of the trending event, list of users who post the data, location of the users, retweet count. It gives the view of the data like tree not the graphical representation of the result. As it gives these details, the project is useful for efficient detection of anomalous event and can track the event in a short span of time. It is also useful for the product manufacturer to find the feedback of the product that they post in the twitter.

**LIST OF FIGURES**

**CHAPTER NO. TITLE PAGE NO.**

**4.** 4.1 System architecture 17

4.2 Data flow diagram 18

4.3 UML diagram 19

4.3.1 Use case diagram 19

4.3.2 Activity diagram 20

4.3.3 Sequence diagram 21

4.3.4 Class diagram 22

**CHAPTER 1**

**INTRODUCTION**

Event monitoring of the social media ‘twitter’ became an essential part. Because usage of social media increased and enormous amount of data have been generated by the people every day. People regularly post the data in the twitter. So, analysing big data is a difficult task. “Big data” is a term used to describe a collection of data sets with the three characteristics – Volume, Velocity and Variety. Big data has become the new frontier of information management given the amount of data today’s systems are generating and consuming. The size and complexity of big data makes it difficult to use traditional database management and data processing tools. It has driven the need for technological infrastructure and tools that can capture, store, analyse and visualize vast technological infrastructure and tools that can capture, store, analyse and visualize vast amounts of disparate structured and unstructured data. There exists a situation where twitter generates huge data. For the analysing purpose, twitter has many members to analyse the data from the database. To overcome their difficulty, we approach this system. This system performs the regular monitoring of the tweets posted by the twitter users. It index the data and through that it find the same data entry in the future and past by correlating the data. Likewise, it finds the retweets and reposts count. By this finding, we can track the anomalous event and the list of users. From which we can find the trending events in the twitter. The system also tracks the location of the people who post the tweets. By tracking the location, we shall know the area from where the posts are posted in a high rate. As it gives the count of the tweets or reposts, it is easy to find when and where the information is getting popular. The system clearly presents its advantages over existing systems and methods from both the event monitoring perspective and the system perspective for the emerging event monitoring task.

* 1. **EXISTING SYSTEM**

Twitter Monitor provides online detection for general emerging anomalous events but could not reveal multiple aspects of the events nor track the evolution of them. CLEar provides real-time anomalous event detection and tracking but could not provide correlation analysis of them. System could detect potential anomalous events at small scale but is confined to only detection, and could not provide tracking and correlation analysis. The system has indexing system optimization combined with evolution tracking, but does not provide much of the detailed monitoring analysis. Existing systems, take a lot of time during update of graph structure. They also cannot rebalance workload when some nodes in the graph are more frequently updated than others, which is a common case when processing popular words in microblog texts. System is implemented through the Hadoop framework and uses Map reduce technique. And it also uses programming language java in this system.

**1.1.1 DRAWBACKS**

* It is slower than LDA, hence cannot be applied in a real-time scenario.
* The method would also tend to generate anomalous events that only contain a single keyword as description, which is hard to comprehend for users.
* None of the existing system methods provide the horizontal scalability with distributed implementations of their algorithms, nor do they investigate system optimizations for their applications.
  1. **PROPOSED SYSTEM**

The aim of the project is to develop a system which gives the detailed analysis of the data over text streams in twitter. The system provides the details like number of users tweeted the same event with timing, reposts or retweets count, location of the users who posted the tweet, location wise count. The system also helps the customers to collect the feedback of their product. System uses advanced tools for the process of analysing the data.

System analyses the data from the twitter by hacking. So, our system acts as a google server.

* + 1. **ADVANTAGES**
* This system represents a view of the original tweets, the retweets count as tree and location view.
* System analyses data from the twitter directly, it doesn’t have any duplicate data.
* This system enhances scalable monitoring of the fed data.
* It also useful for the efficient feedback collection to the customers about their product.

**CHAPTER 2**

**LITERATURE SURVEY**

**2.1 “Ring: Real-Time Emerging Anomaly Monitoring System over Text Streams”**

**AUTHORS**: Weiren Yu, Jianxin Li, Md Zakirul Alam Bhuiyan, Richong Zhang and Jinpeng Huai

In this paper, we present RING, a real-time emerging anomaly monitoring system over microblog text streams. Emerging anomaly monitoring has attracted much attention from the research domain. Here we aim to monitor emerging anomalous events on microblog platforms. Our emerging anomaly monitoring methods are based on graph mining techniques, which provides unique opportunities to integrate our emerging anomaly monitoring research and system optimizations. In the RING system, emerging anomaly monitoring includes early detection, correlation analysis and temporal evolution tracking of anomalous events. Early detection would capture emerging events before they go viral. Correlation analysis would automatically reveal multiple aspects of the anomalous event, or the causality of anomalous events, or categorical structure of related anomalies. For example, the history and current development of a political event reveal multiple facades of the event. The capture of a criminal and the crime he had committed form a causal relationship. Different genres of news from an agency detected at the same time would reveal categorical information. According to its popularity, anomalous events would emerge from different time granularities, e.g., a publicly concerned long trial or an overnight pop concert. The temporal evolution tracking of events could recover the evolution process of an anomalous event, to trace its origin and get the big picture. Such monitoring happens in real-time and provides valuable intelligence for government agencies, news groups and marketing agencies, etc.

**2.2 “Earthquake shakes twitter users: real-time event detection by social sensors”**

**AUTHORS:**  T. Sakaki, M. Okazaki, and Y. Matsuo

Twitter, a popular microblogging service, has received much attention recently. An important characteristic of Twitter is its real-time nature. For example, when an earthquake occurs, people make many Twitter posts (tweets) related to the earthquake, which enables detection of earthquake occurrence promptly, simply by observing the tweets. As described in this paper, we investigate the real-time interaction of events such as earthquakes in Twitter and propose an algorithm to monitor tweets and to detect a target event. To detect a target event, we devise a classifier of tweets based on features such as the keywords in a tweet, the number of words, and their context. Subsequently, we produce a probabilistic spatiotemporal model for the target event that can find the centre and the trajectory of the event location. We consider each Twitter user as a sensor and apply Kalman filtering and particle filtering, which are widely used for location estimation in ubiquitous/pervasive computing. The particle filter works better than other comparable methods for estimating the centres of earthquakes and the trajectories of typhoons. As an application, we construct an earthquake reporting system in Japan. Because of the numerous earthquakes and the large number of Twitter users throughout the country, we can detect an earthquake with high probability (96% of earthquakes of Japan Meteorological Agency (JMA) seismic intensity scale 3 or more are detected) merely by monitoring tweets. Our system detects earthquakes promptly and sends e-mails to registered users. Notification is delivered much faster than the announcements that are broadcast by the JMA.

**2.3 “Scalable distributed event detection for twitter”**

**AUTHORS:** R. McCreadie, C. Macdonald, I. Ounis, M. Osborne, and S. Petrovic

Social media streams, such as Twitter, have shown themselves to be useful sources of real-time information about what is happening in the world. Automatic detection and tracking of events identified in these streams have a variety of real-world applications, e.g. identifying and automatically reporting road accidents for emergency services. However, to be useful, events need to be identified within the stream with a very low latency. This is challenging due to the high volume of posts within these social streams. In this paper, we propose a novel event detection approach that can both effectively detect events within social streams like Twitter and can scale to thousands of posts every second. Through experimentation on a large Twitter dataset, we show that our approach can process the equivalent to the full Twitter Fire hose stream, while maintaining event detection accuracy and outperforming an alternative distributed event detection system.

**2.4 “Emerging topic detection for organizations from microblogs”**

**AUTHORS:** Y. Chen, H. Amiri, Z. Li, and T.-S. Chua

Microblog services have emerged as an essential way to strengthen the communications among individuals and organizations. These services promote timely and active discussions and comments towards products, markets as well as public events, and have attracted a lot of attentions from organizations. In particular, emerging topics are of immediate concerns to organizations since they signal current concerns of, and feedback by their users. Two challenges must be tackled for effective emerging topic detection. One is the problem of real-time relevant data collection and the other is the ability to model the emerging characteristics of detected topics and identify them before they become hot topics. To tackle these challenges, we first design a novel scheme to crawl the relevant messages related to the designated organization by monitoring multi-aspects of microblog content, including users, the evolving keywords and their temporal sequence. We then develop an incremental clustering framework to detect new topics, and employ a range of content and temporal features to help in promptly detecting hot emerging topics. Extensive evaluations on a representative real-world dataset based on Twitter data demonstrate that our scheme is able to characterize emerging topics well and detect them before they become hot topics.

**2.5 “Topicsketch: Realtime bursty topic detection from twitter”**

**AUTHORS:** W. Xie, F. Zhu, J. Jiang, E.-P. Lim, and K.Wang

Twitter has become one of the largest microblogging platforms for users around the world to share anything happening around them with friends and beyond. A bursty topic in Twitter is one that triggers a surge of relevant tweets within a short period of time, which often reflects important events of mass interest. How to leverage Twitter for early detection of bursty topics has therefore become an important research problem with immense practical value. Despite the wealth of research work on topic modelling and analysis in Twitter, it remains a challenge to detect bursty topics in real-time. As existing methods can hardly scale to handle the task with the tweet stream in real-time, we propose in this paper TopicSketch, a sketch-based topic model together with a set of techniques to achieve real-time detection. We evaluate our solution on a tweet stream with over 30 million tweets. Our experiment results show both efficiency and effectiveness of our approach. Especially it is also demonstrated that TopicSketch on a single machine can potentially handle hundreds of millions tweets per day, which is on the same scale of the total number of daily tweets in Twitter, and present bursty events in finer-granularity.

**2.6 “Signitrend: scalable detection of emerging topics in textual streams by hashed significance thresholds”**

**AUTHORS:** E. Schubert, M. Weiler, and H.-P. Kriegel

Social media such as Twitter or weblogs are a popular source for live textual data. Much of this popularity is due to the fast rate at which this data arrives, and there are a number of global events - such as the Arab Spring - where Twitter is reported to have had a major influence. However, existing methods for emerging topic detection are often only able to detect events of a global magnitude such as natural disasters or celebrity deaths, and can monitor user-selected keywords or operate on a curated set of hashtags only. Interesting emerging topics may, however, be of much smaller magnitude and may involve the combination of two or more words that themselves are not unusually hot at that time. Our contributions to the detection of emerging trends are three-fold first of all, we propose a significance measure that can be used to detect emerging topics early, long before they become "hot tags", by drawing upon experience from outlier detection. Secondly, by using hash tables in a heavy-hitters type algorithm for establishing a noise baseline, we show how to track even all keyword pairs using only a fixed amount of memory. Finally, we aggregate the detected co-trends into larger topics using clustering approaches, as often as a single event will cause multiple word combinations to trend at the same time.

**CHAPTER 3**

**REQUIREMENT AND ANALYSIS**

**3.1 FEASIBILITY STUDY**

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. A feasibility study allows project managers to investigate the possible negative and positive outcomes of a project investing too much time and money. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

* TECHNICAL FEASIBILITY
* ECONOMIC FEASIBILITY
* SOCIAL FEASIBILITY
* LEGAL FEASIBILITY
* SCHEDULING FEASIBILITY

### 3.1.1 TECHNICAL FEASIBILITY

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

**3.1.2 ECONOMICAL FEASIBILITY**

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

**3.1.3 SOCIAL FEASIBILITY**

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

**3.1.4 LEGAL FEASIBILITY**

This study investigates if the proposed system conflicts with legal requirements like data protection acts or social media laws.

**3.1.5 SCHEDULING FEASIBILITY**

In this feasibility study, we estimate how much time the system will take to complete, and with our technical skills we need to estimate the period to complete the project using various methods of estimation.

**3.2 HARDWARE REQUIREMENTS**

* Processor : Intel i3
* RAM : 4GB
* Internal memory : 500GB
* Input device : Keyboard, Mouse
* Output device : Monitor

**3.3 SOFTWARE REQUIREMENTS**

* Operating System : Windows 10
* Coding Language : Java 1.8.0
* IDE : Eclipse Luna
* Tool : Spark, Hadoop

**3.4 FUNCTIONAL REQUIREMENTS**

A functional requirements defines a function of a system or it components. A function is described as a set of inputs, the behaviour and outputs. Functional requirement deals with what the system should provide for users. They include description of the required functions, outlines of associated reports or online queries and details of data to be held in the system.

In our system the functional requirements are,

* Displaying the users who post the event
* Displaying the location of the post where it is posted in enormous amount
* Displaying the retweet count

Input : Event name

Behaviour : It can viewed anytime from anywhere by the approved customers of twitter and twitter analyst.

Output : Displaying the details of particular event.

**3.5 NON-FUNCTIONAL REQUIREMENTS**

The definition for a non-functional requirement is that it essentially specifies how the system should behave and that it is a constraint up on the systems behaviour.

Performance : Reliability, Accessibility

Cost : It includes labour cost, miscellaneous cost

Non-functional requirements include the following also:

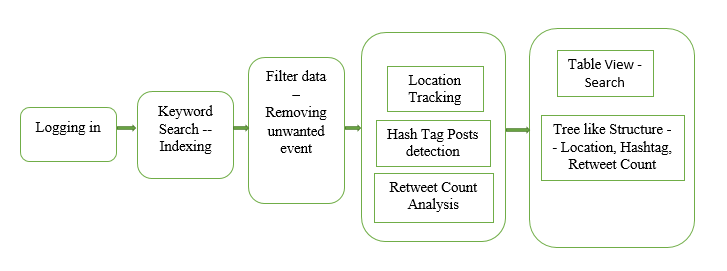
* The system shall be user friendly and consistent.
* The system shall allow developer access to installed environment.

**CHAPTER 4**

**DESIGN**

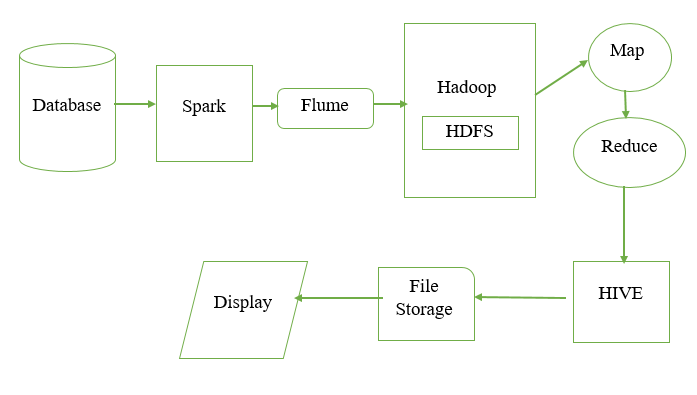
**4.1 SYSTEM ARCHITECTURE**

A system architecture or systems architecture is the conceptual model that defines the structure, behaviour and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structures and behaviours of the system. A system architecture can consist of system components and the sub-systems developed, that will work together to implement the overall system.



**4.2 DATA FLOW DIAGRAM**

A data-flow diagram (DFD) is a way of representing a flow of a data of a [process](https://en.wikipedia.org/wiki/Process) or a system The DFD also provides information about the outputs and inputs of each entity and the process itself. A data-flow diagram has no control flow, there are no rules and no loops. Specific operations based on the data can be represented by a flowchart. The data-flow diagram is part of the structured-analysis modelling tools. When using UML, the activity diagram typically takes over the role of the data-flow diagram. A special form of data-flow plan is a site-oriented data-flow plan. Data-flow diagrams (DFD) quickly became a popular way to visualize the major steps and data involved in software-system processes.

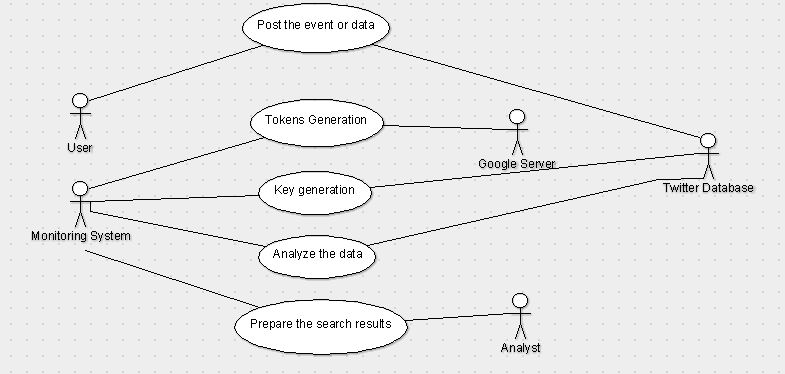


**4.3 UML DIAGRAM**

A UML diagram is a diagram based on the UML (Unified Modeling Language) with the purpose of **visually representing a system** along with its main actors, roles, actions, artifacts or classes, in order to better understand, alter, maintain, or document information about the system. UML is a modern approach to modeling and documenting software. In fact, it’s one of the most popular business [process modeling techniques](https://tallyfy.com/business-process-modeling-techniques). It is the diagrammatic representation of software components. There are several types of UML diagram.

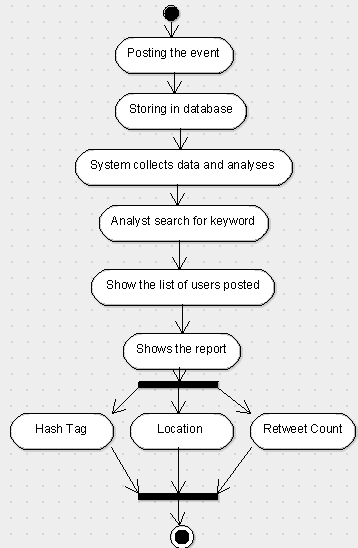
**4.3.1 USECASE DIAGRAM**

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different [use cases](https://en.wikipedia.org/wiki/Use_case) in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well. The use cases are represented by either circles or ellipses.

****

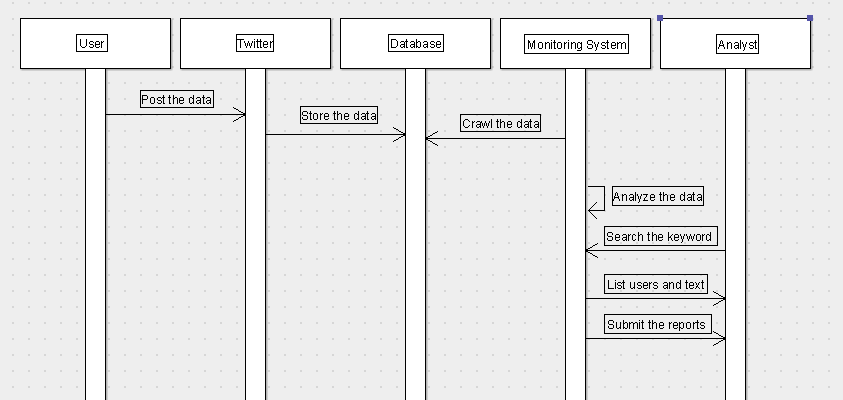
**4.3.2 ACTIVITY DIAGRAM**

Activity diagram is another important diagram in UML to describe the dynamic aspects of the system. Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent. Activity diagrams deal with all type of flow control by using different elements such as fork, join, etc.

****

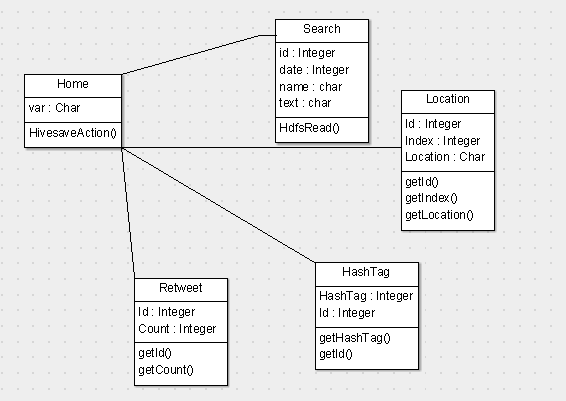
**4.3.3 SEQUENCE DIAGRAM**

A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems. Sequence diagrams use notations like actor, lifeline and messages. Messages are of different types as follows create message, delete message, self-message and reply message. Sequence diagrams visualise how messages and tasks move between objects or components in a system.



**4.3.4 CLASS DIAGRAM**

 A class diagram in the [Unified Modeling Language](https://en.wikipedia.org/wiki/Unified_Modeling_Language) (UML) is a type of static structure diagram that describes the structure of a system by showing the system's [classes](https://en.wikipedia.org/wiki/Class_(computer_science)), their attributes, operations (or methods), and the relationships among objects. The class diagram is the main building block of [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming) modeling. It is used for general [conceptual modeling](https://en.wikipedia.org/wiki/Conceptual_model) of the structure of the application, and for detailed modeling translating the models into [programming code](https://en.wikipedia.org/wiki/Programming_code). Class diagrams can also be used for data modeling.  The classes in a class diagram represent both the main elements, interactions in the application, and the classes to be programmed.



**\***

**CHAPTER 5**

**CODING**

**5.1 LOGIN**

**5.2 HOME PAGE**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="description" content="">

<meta name="author" content="">

<title>Online Twitter Sumarization</title>

<link href="css/bootstrap.min.css" rel="stylesheet">

<link href="css/plugins/metisMenu/metisMenu.min.css"

rel=”style sheet”>

<link href="css/plugins/timeline.css" rel="stylesheet">

<link href="css/sb-admin-2.css" rel="stylesheet">

<link href="css/plugins/morris.css" rel="stylesheet">

<link href="css/home.css" rel="stylesheet">

<link href="font-awesome-4.1.0/css/font-awesome.min.css"

rel="stylesheet" type="text/css">

<script>

src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js">

</script>

<script>

$(function(){

$( "#submit" ).click(function() {

var formInput=$(this).serialize();

var search=$('#search').val();

console.log($('#search').val());

$.getJSON('SearchAction.action?search='+search, formInput, function(data)

{

if(data.searchDetails !=null)

{

$.each(data.searchDetails, function(key, val) {

//Whatever you what to do, eg.

console.log(key);

console.log(val.id);

console.log(val.screenName);

console.log(val.name);

console.log(val.createdDate);

console.log(val.text);

var row=' <tr> <td style="word-break:break-all;"> '+ val.id +'</td> <td style="word-break:break-all;">'+val.screenName +'</td><td style="word-break:break-all;">'+val.name +'</td> <td style="word-break:break-all;">'+val.createdDate +'</td> <td style="word-break:break-all;">'+val.text +'</td></tr> ';

$('#result').append(row);

});

}

});

});

})

</script>

</head>

<body>

<form action="HiveSaveAction.action" id="searchForm">

<div id="wrapper">

<nav class="navbar navbar-default navbar-static-top" role="navigation" style="margin-bottom: 0">

<div class="navbar-header">

<button type="button" class="navbar-toggle" data toggle = "collapse" data-target=".navbar-collapse">

<span class="sr-only">Toggle navigation</span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

</div>

<ul class="nav navbar-top-links navbar-right">

<li class="dropdown">

<a class="dropdown-toggle" data-toggle="dropdown"

href="#">

<i class="fa fa-user fa-fw"></i> <i class="fa fa-caret- down"></i>

</a>

</li>

</ul>

<div class="navbar-default sidebar" role="navigation">

<div class="sidebar-nav navbar-collapse">

<ul class="nav" id="side-menu">

<li>

<a class="active" href="home.jsp"><i class="">

</i>Home</a>

</li>

<!-- <li>

<a class="active" href="flume.jsp"><i class=""> </i>

Flume</a>

</li>

<li>

<a class="active" href="hdfs.jsp"><i class="">

</i>Hive</a>

</li> -->

<li>

<a class="active" href="clustering.jsp"><i class="">

</i>Reports</a>

</li>

</ul>

</div>

</div>

</nav>

<div id="page-wrapper">

<div class="row">

<div class="col-lg-12">

<h1 class="page-header">Search</h1>

<table>

<tr>

<td colspan="1">

<label><b>Search Word</b></label>

</td>

<td colspan="1">

<input id="search" name="search"

type="text"/>

</td>

</tr>

<tr align="center">

<td colspan="2">

<input id="submit" type="button"

value="Submit"/>

</td>

</tr>

</table>

</div>

</div>

<div class="row">

<br>

</div>

<div class="row">

<div class="col-lg-8s">

<div class="panel panel-default">

<div class="panel-heading">

<table id="result" cellspacing="20" width="100%" style="

border:1px solid rgb(177, 210, 228);">

<tbody>

<tr style="">

<th width="20%" bgcolor="#63E14F" align="left">Id</th>

<th width="20%" bgcolor="#63E14F" align="left">Screen Name</th>

<th width="20%" bgcolor="#63E14F" align="left">Name</th>

<th width="20%" bgcolor="#63E14F" align="left">Created Date</th>

<th width="20%" bgcolor="#63E14F" align="left">Text</th>

</tr>

</tbody>

</table>

</div>

</div>

</div>

<div class="col-lg-4">

</div>

</div>

</div>

</div>

</form>

</body>

</html>

**5.2.1 SEARCH**

package com.mycompany.action;

import java.io.File;

import java.io.FileInputStream;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.Statement;

import java.util.Properties;

import javax.servlet.ServletContext;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FileStatus;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.struts2.ServletActionContext;

import com.opensymphony.xwork2.ActionSupport;

public class HiveSaveAction {

private static String driverName = "org.apache.hadoop.hive.jdbc.HiveDriver";

public String execute() {

try

{

ServletContext context = ServletActionContext.getServletContext();

String fpath = context.getRealPath("") + "/" + "config.properties";

FileInputStream fileInputStream=new FileInputStream(new

File(fpath));

Properties properties=new Properties();

properties.load(fileInputStream);

Class.forName(driverName);

Connection con = DriverManager.getConnection(

"jdbc:hive://localhost:10000/dataminingbigdata", "", "");

Statement stmt = con.createStatement();

Configuration conf = new Configuration();

conf.addResource(new Path(properties.getProperty("hadoopLoc")));

FileSystem fs = FileSystem.get(conf);

Path pt = new Path(properties.getProperty("hdfsLoc"));

FileStatus[] status = fs.listStatus(pt);

// fs.delete(pt, true);

for (int i = 0; i < status.length; i++) {

String filePath = status[i].getPath().toString();

Path path=new Path(filePath);

System.out.println("calling insert query !!");

String sql = "load data inpath '" + "/result/"+path.getName()

+ "' OVERWRITE into table " + "twitter\_table";

stmt.executeQuery(sql);

}

fileInputStream.close();

stmt.close();

con.close();

fs.close();

System.out.println("Inserted Sucessfully !!");

}

catch(Exception e){

e.printStackTrace();

}

return ActionSupport.SUCCESS;

}

}

**5.2.1.1 HIVE CLIENT CALL**

package com.mycompany.logic;

import java.io.File;

import java.io.FileOutputStream;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

public class HiveClientCall {

private static String driverName = "org.apache.hadoop.hive.jdbc.HiveDriver";

private JsonConverter jsonConverter=new JsonConverter();

public void clientCall(String FilePath) {

try {

Class.forName(driverName);

Connection con = DriverManager.getConnection(

"jdbc:hive://localhost:10000/dataminingbigdata", "", "");

Statement stmt = con.createStatement();

String sql = "select \* from twitter\_table";

ResultSet res=stmt.executeQuery(sql);

String content="";

while(res.next())

{

String row=res.getString(1);

if(row.contains("since\_id\_str"))

{

//FileOutputStream fileOutputStream=new

FileOutputStream(new File("result.json"));;

content+=row;

System.out.println(content);

FileOutputStream fileOutputStream=new

FileOutputStream(new File(FilePath+"/result.json"));;

fileOutputStream.write(content.getBytes());

fileOutputStream.close();

content="";

}

else

{

content+=row;

}

}

}

catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

public static void main(String[] args) {

HiveClientCall hiveClientCall = new HiveClientCall();

hiveClientCall.clientCall("");

}

}

**5.2.1.2 HDFS READ**

package com.mycompany.workout;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FileStatus;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

public class HdfsRead {

public HdfsRead() throws IOException {

Configuration conf = new Configuration();

conf.addResource(new Path(

"/project/ProjectM/Proj/Projects/JP/Hadoop/hadoop-

1.2.1/conf/core-site.xml"));

FileSystem fs = FileSystem.get(conf);

Path pt = new Path("hdfs://localhost:8020/new");

fs.mkdirs(pt);

}

public static void main(String[] args) throws IOException {

new HdfsRead();

}

}

**5.2.1.3 DATA COLLECTION**

package com.mycompany.action;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import javax.servlet.ServletContext;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FileStatus;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.struts2.ServletActionContext;

import com.opensymphony.xwork2.ActionSupport;

public class FlumeSaveAction {

public String execute() {

try {

ServletContext context = ServletActionContext.getServletContext();

String filePath = context.getRealPath("") + "/result";

File file = new File(filePath);

String[] file\_list = file.list();

for (String fileAbsolutePath : file\_list) {

String fileAbsPath = filePath + "/" + fileAbsolutePath;

File file\_abs = new File(fileAbsPath);

FileInputStream fileInputStream = new

FileInputStream(file\_abs);

byte[] byt = new byte[fileInputStream.available()];

fileInputStream.read(byt);

fileInputStream.close();

String outPutDir = new File(new File

(file\_abs.getParent()).getParent()).getParent()+ "/result";

FileOutputStream fileOutputStream2 = new

FileOutputStream(new File(outPutDir + "/"+ new File(

fileAbsPath ).getName()));

fileOutputStream2.write(byt);

fileOutputStream2.close();

file\_abs.delete();

}

} catch (Exception e) {

e.printStackTrace();

}

return ActionSupport.SUCCESS;

}

public static void main(String[] args) {

FlumeSaveAction filAction = new FlumeSaveAction();

filAction.execute();

}

}

**5.2.1.4 JSON**

package com.mycompany.logic;

import java.io.File;

import java.io.FileInputStream;

import java.util.ArrayList;

import java.util.List;

import java.util.Vector;

import org.json.simple.JSONArray;

import org.json.simple.JSONObject;

import org.json.simple.parser.JSONParser;

import com.mycompany.vo.ResponseVo;

import com.mycompany.vo.SearchVo;

public class JsonConverter {

public void convertTrendsValue(String fileName) {

try {

JSONParser parser = new JSONParser();

FileInputStream fileInputStream = new

FileInputStream(new File(fileName));

byte[] byt = new byte[fileInputStream.available()];

fileInputStream.read(byt);

String s = new String(byt);

try {

Object obj = parser.parse(s);

JSONArray array = (JSONArray) obj;

JSONObject obj2 = (JSONObject) array.get(0);

JSONArray trends = (JSONArray) obj2.get("trends");

for (int i = 0; i < trends.size(); i++) {

JSONObject jsonObject = (JSONObject)

trends.get(i);

Vector<String> rowData = new

Vector<String>();

rowData.add(jsonObject.get("name").toString());

rowData.add(jsonObject.get("url").toString());

/\*

\* MainForm.defaultTableModel.addRow(rowData);

\*MainForm.jComboBox.addItem(jsonObject.get("name");

\* .toString());

\*/

}

} catch (Exception pe) {

pe.printStackTrace();

} finally {

fileInputStream.close();

}

} catch (Exception e) {

e.printStackTrace();

}

}

public List<SearchVo> convertSearchValue(String fileName) {

List<SearchVo> searchDetails = new ArrayList<SearchVo>();

try {

JSONParser parser = new JSONParser();

FileInputStream fileInputStream = new FileInputStream(new File(

fileName));

byte[] byt = new byte[fileInputStream.available()];

fileInputStream.read(byt);

String s = new String(byt);

try {

Object obj = parser.parse(s);

JSONObject jsonObject = (JSONObject) obj;

JSONArray trends = (JSONArray) jsonObject.get("statuses");

System.out.println(trends.size());

for (int i = 0; i < trends.size(); i++) {

JSONObject jsObject = (JSONObject) trends.get(i);

JSONObject jObject = (JSONObject)

jsObject.get("user");

SearchVo searchVo = new SearchVo();

searchVo.setId(jObject.get("id").toString());

searchVo.setName(jObject.get("name").toString());

searchVo.setScreenName(jObject.get("screen\_name")

.toString());

searchVo.setScreenName(jObject.get("screen\_name")

.toString());

searchVo.setText(jsObject.get("text").toString());

searchVo.setCreatedDate(jsObject.get("created\_at")

.toString());

searchDetails.add(searchVo);

}

} catch (Exception pe) {

pe.printStackTrace();

} finally {

fileInputStream.close();

}

}

catch (Exception e) {

e.printStackTrace();

}

return searchDetails;

}

public void convertAggregationValue(String fileName) {

try {

JSONParser parser = new JSONParser();

FileInputStream fileInputStream = new FileInputStream(new File(

fileName));

byte[] byt = new byte[fileInputStream.available()];

fileInputStream.read(byt);

String s = new String(byt);

try {

Object obj = parser.parse(s);

JSONObject jsonObject = (JSONObject) obj;

JSONArray trends = (JSONArray) jsonObject.get("statuses");

System.out.println(trends.size());

/\*

\* JSONObject jsObject = (JSONObject) trends

.get(trends.size()

\* - 1); String inTime = jsObject.get("created\_at").toString();

\* jsObject = (JSONObject) trends.get(0); String outTime =

\* jsObject.get("created\_at").toString(); int retweet\_count = 0;

\*/

} catch (Exception pe) {

pe.printStackTrace();

} finally {

fileInputStream.close();

}

} catch (Exception e) {

e.printStackTrace();

}

}

// public List<>

public List<ResponseVo> getResponseVos(String filePath, String ids) {

List<ResponseVo> responseList=new ArrayList<ResponseVo>();

try {

JSONParser parser = new JSONParser();

FileInputStream fileInputStream = new FileInputStream(new File(

filePath));

byte[] byt = new byte[fileInputStream.available()];

fileInputStream.read(byt);

String s = new String(byt);

try {

Object obj = parser.parse(s);

JSONObject jsonObject = (JSONObject) obj;

JSONArray trends = (JSONArray) jsonObject.get("statuses");

for (int i = 0; i < trends.size(); i++) {

JSONObject jsObject = (JSONObject) trends.get(i);

JSONObject jObject = (JSONObject)

jsObject.get("user");

JSONObject hashEntity = (JSONObject) jsObject

.get("entities");

JSONArray hash\_trends = (JSONArray) hashEntity

.get("hashtags");

if (ids.contains(jsObject.get("id").toString())) {

ResponseVo responseVo = new ResponseVo();

responseVo.setId(jsObject.get("id").toString());

responseVo.setText(jsObject.get("text").toString());

responseVo.setName(jObject.get("name").toString());

responseVo.setCreatedAt(jsObject.get("created\_at")

.toString());

responseVo.setLocation(jObject.get("location")

.toString());

responseVo.setRetweetCount(jsObject

.get("retweet\_count").toString());

String hashTag = "";

for (int hash\_index = 0; hash\_index <

hash\_trends.size(); hash\_index++) {

JSONObject hashObject = (JSONObject)

hash\_trends.get(hash\_index);

if (hashTag.equals(""))

hashTag =

hashObject.get("text").toString();

else {

hashTag += ","

+ hashObject.get("text").toString();

}

}

responseVo.setHashTag(hashTag);

responseList.add(responseVo);

}

}

} catch (Exception pe) {

pe.printStackTrace();

} finally {

fileInputStream.close();

}

} catch (Exception e) {

e.printStackTrace();

}

return responseList;

}

public static void main(String[] args) {

JsonConverter jsonConverter = new JsonConverter();

jsonConverter.convertAggregationValue("result.json");

}

}

**5.2.1.5 TWITTER CALL**

package com.mycompany.logic;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.util.List;

import java.util.Properties;

import oauth.signpost.OAuthConsumer;

import oauth.signpost.commonshttp.CommonsHttpOAuthConsumer;

import org.apache.commons.fileupload.FileUpload;

import org.apache.commons.io.IOUtils;

import org.apache.http.HttpHost;

import org.apache.http.HttpResponse;

import org.apache.http.client.HttpClient;

import org.apache.http.client.methods.HttpGet;

import org.apache.http.conn.params.ConnRoutePNames;

import org.apache.http.impl.client.DefaultHttpClient;

import com.mycompany.vo.SearchVo;

public class TwitterRestCall {

static String consumerKeyStr = "";

static String consumerSecretStr = "";

static String accessTokenStr = "";

static String accessTokenSecretStr = "";

OAuthConsumer consumer;

private JsonConverter jsonConverter;

public TwitterRestCall(String filePath) {

jsonConverter = new JsonConverter();

consumerKeyStr = readProperties(filePath, "consumerKey");

consumerSecretStr = readProperties(filePath, "consumerSecret");

accessTokenStr = readProperties(filePath, "accessToken");

accessTokenSecretStr = readProperties(filePath, "accessTokenSecret");

consumer = new CommonsHttpOAuthConsumer(consumerKeyStr,

consumerSecretStr);

consumer.setTokenWithSecret(accessTokenStr, accessTokenSecretStr);

}

private String readProperties(String filePath, String key) {

String url = "";

try {

Properties properties = new Properties();

FileInputStream fileInputStream = new FileInputStream(new File(

filePath));

properties.load(fileInputStream);

url = properties.getProperty(key);

} catch (Exception e) {

e.printStackTrace();

}

return url;

}

public List<SearchVo> getTweetSearch(String filePath, String queryName) {

try {

String url = readProperties(filePath, "search");

queryName = queryName.replace("#", "");

queryName = queryName.replace(" ", "+");

url += "=" + queryName + "&count=1000";

System.out.println(url);

HttpGet request = new HttpGet(url);

consumer.sign(request);

HttpClient client = new DefaultHttpClient();

HttpResponse response = client.execute(request);

int statusCode = response.getStatusLine().getStatusCode();

System.out.println(statusCode + ":"

+ response.getStatusLine().getReasonPhrase());

filePath = filePath.replaceAll("config.properties",

"result/result.json");

File file=new File(filePath);

FileOutputStream fileOutputStream = new FileOutputStream(file);

fileOutputStream.write(IOUtils.toString(

response.getEntity().getContent()).getBytes());

List<SearchVo> searchDetails = jsonConverter

.convertSearchValue(filePath);

fileOutputStream.flush();

fileOutputStream.close();

/\*FileInputStream fileInputStream=new FileInputStream(new

File(filePath);

byte[] byt=new byte[fileInputStream.available()];

fileInputStream.read(byt);

fileInputStream.close();

String outPutDir=new File(new

File(file.getParent()).getParent()).getParent()+"/result";

FileOutputStream fileOutputStream2=new FileOutputStream(new

File(outPutDir+"/"+new File(filePath).getName()));

fileOutputStream2.write(byt);

fileOutputStream2.close();\*/

return searchDetails;

} catch (Exception e) {

e.printStackTrace();

}

return null;

}

}

**5.2.1.6 SEARCH RESULTS**

package com.mycompany.vo;

public class SearchVo {

private String id;

private String screenName;

private String name;

private String createdDate;

private String text;

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getScreenName() {

return screenName;

}

public void setScreenName(String screenName) {

this.screenName = screenName;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getCreatedDate() {

return createdDate;

}

public void setCreatedDate(String createdDate) {

this.createdDate = createdDate;

}

public String getText() {

return text;

}

public void setText(String text) {

this.text = text;

}

}

**5.2.2 SEARCH STORAGE**

package com.mycompany.logic;

import java.io.File;

import java.io.FileInputStream;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import org.json.simple.JSONArray;

import org.json.simple.JSONObject;

import org.json.simple.parser.JSONParser;

import com.mycompany.vo.HashTagVo;

import com.mycompany.vo.LocationVo;

import com.mycompany.vo.RetweetCountVo;

public class ClusteringLogic {

public List<LocationVo> LocationServices(String filePath) {

List<LocationVo> locationList = new ArrayList<LocationVo>();

try

{

System.out.println(filePath);

HashMap<String, LocationVo> duplicateLocationMap = new

HashMap<String, LocationVo>();

// List<String> duplicateLocationList = new

ArrayList<String>();

JSONParser parser = new JSONParser();

FileInputStream fileInputStream = new FileInputStream(new

File(filePath + "/" + "result.json"));

/\*

\* FileInputStream fileInputStream = new FileInputStream(new

\*File( "result.json"));

\*/

byte[] byt = new byte[fileInputStream.available()];

fileInputStream.read(byt);

String s = new String(byt);

Object obj = parser.parse(s);

JSONObject jsonObject = (JSONObject) obj;

JSONArray trends = (JSONArray) jsonObject.get("statuses");

for (int i = 0; i < trends.size(); i++) {

JSONObject jsObject = (JSONObject) trends.get(i);

JSONObject jObject = (JSONObject) jsObject.get("user");

String location = jObject.get("location").toString().trim()

.toUpperCase();

if (!location.equals("") &&

!duplicateLocationMap.containsKey(location)) {

LocationVo locationVo = new LocationVo();

locationVo.setIndex(String.valueOf(i));

locationVo.setLocation(location);

locationVo.setId(jsObject.get("id").toString());

duplicateLocationMap.put(location, locationVo);

} else {

if (!location.equals("")) {

LocationVo locationVo =

duplicateLocationMap.get(location);

String id = locationVo.getId() + ","

+ jsObject.get("id").toString();

locationVo.setId(id);

duplicateLocationMap.remove(location);

duplicateLocationMap.put(location,

locationVo);

}

}

}

Iterator iter = duplicateLocationMap.entrySet().iterator();

while (iter.hasNext()) {

Map.Entry pairs = (Map.Entry) iter.next();

locationList.add((LocationVo) pairs.getValue());

}

for (LocationVo locationVo : locationList) {

System.out.println(locationVo.getId());

}

fileInputStream.close();

} catch (Exception e) {

e.printStackTrace();

}

return locationList;

}

public List<HashTagVo> HashTagServices(String filePath) {

List<HashTagVo> hashTagList = new ArrayList<HashTagVo>();

HashMap<String, HashTagVo> duplicateHashTagMap = new

HashMap<String, HashTagVo>();

try {

JSONParser parser = new JSONParser();

FileInputStream fileInputStream = new FileInputStream(new

File(filePath + "/" + "result.json"));

byte[] byt = new byte[fileInputStream.available()];

fileInputStream.read(byt);

String s = new String(byt);

Object obj = parser.parse(s);

JSONObject jsonObject = (JSONObject) obj;

JSONArray trends = (JSONArray) jsonObject.get("statuses");

for (int i = 0; i < trends.size(); i++)

{

JSONObject jsObject = (JSONObject) trends.get(i);

JSONObject jObject = (JSONObject)

jsObject.get("entities");

JSONArray hash\_trends = (JSONArray)

jObject.get("hashtags");

for (int hash\_index = 0; hash\_index < hash\_trends.size();

hash\_index++)

{

JSONObject hashObject = (JSONObject)

hash\_trends.get(hash\_index);

if (!hashObject.get("text").toString().equals("") && !duplicateHashTagMap.containsKey(hashObject.get("text").toString()))

{

HashTagVo hashTagVo = new HashTagVo();

hashTagVo.setHashTag(hashObject.get("text")

.toString());

hashTagVo.setId(jsObject.get("id").toString());

// hashTagList.add(hashTagVo);

duplicateHashTagMap.put(hashObject.get("text")

.toString(), hashTagVo);

} else

{

if (!hashObject.get("text").toString().equals(""))

{

HashTagVo hashTagVo =

duplicateHashTagMap.get(hashObject.

get("text").toString());

String id = hashTagVo.getId() + ","

+jsObject.get("id").toString();

hashTagVo.setId(id);

duplicateHashTagMap.remove(hashObject.

get("text").toString());

duplicateHashTagMap.put(hashObject.

get("text").toString(), hashTagVo);

}

}

}

}

fileInputStream.close();

} catch (Exception e) {

e.printStackTrace();

}

Iterator iter = duplicateHashTagMap.entrySet().iterator();

while (iter.hasNext()) {

Map.Entry pairs = (Map.Entry) iter.next();

hashTagList.add((HashTagVo) pairs.getValue());

}

return hashTagList;

}

public List<RetweetCountVo> retweetCountServices(String filePath) {

List<RetweetCountVo> retweetList = new ArrayList<RetweetCountVo>();

HashMap<String, RetweetCountVo> duplicateRetweetMap = new

HashMap<String, RetweetCountVo>();

try {

JSONParser parser = new JSONParser();

FileInputStream fileInputStream = new FileInputStream(new File(

filePath + "/" + "result.json"));

byte[] byt = new byte[fileInputStream.available()];

fileInputStream.read(byt);

String s = new String(byt);

Object obj = parser.parse(s);

JSONObject jsonObject = (JSONObject) obj;

JSONArray trends = (JSONArray) jsonObject.get("statuses");

for (int i = 0; i < trends.size(); i++) {

JSONObject jsObject = (JSONObject) trends.get(i);

String retweetCount =

jsObject.get("retweet\_count").toString();

String id = jsObject.get("id").toString();

if (!duplicateRetweetMap.containsKey(retweetCount)) {

RetweetCountVo retweetCountVo=new

RetweetCountVo();

retweetCountVo.setId(id);

retweetCountVo.setRetweetCount(retweetCount);

duplicateRetweetMap.put(retweetCount,

retweetCountVo);

} else {

RetweetCountVo retweetCountVo =

duplicateRetweetMap.get(jsObject.get("retweet\_count"). toString());

id = retweetCountVo.getId() + ","

+ jsObject.get("id").toString();

retweetCountVo.setId(id);

retweetCountVo.setRetweetCount(retweetCount);

duplicateRetweetMap.remove(jsObject.

get("retweet\_count").toString());

duplicateRetweetMap.put(jsObject.

get("retweet\_count").toString(),retweetCountVo);

}

}

fileInputStream.close();

} catch (Exception e) {

e.printStackTrace();

}

Iterator iter = duplicateRetweetMap.entrySet().iterator();

while (iter.hasNext()) {

Map.Entry pairs = (Map.Entry) iter.next();

retweetList.add((RetweetCountVo) pairs.getValue());

}

return retweetList;

}

public static void main(String[] args) {

ClusteringLogic clusteringLogic = new ClusteringLogic();

clusteringLogic.HashTagServices("");

}

}

**5.3 REPORTS**

package com.mycompany.logic;

import java.io.File;

import java.io.FileInputStream;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import org.json.simple.JSONArray;

import org.json.simple.JSONObject;

import org.json.simple.parser.JSONParser;

import com.mycompany.vo.HashTagVo;

import com.mycompany.vo.LocationVo;

import com.mycompany.vo.RetweetCountVo;

public class ClusteringLogic {

public List<LocationVo> LocationServices(String filePath) {

List<LocationVo> locationList = new ArrayList<LocationVo>();

try {

System.out.println(filePath);

HashMap<String, LocationVo> duplicateLocationMap = new

HashMap<String, LocationVo>();

// List<String> duplicateLocationList = new ArrayList<String>();

JSONParser parser = new JSONParser();

FileInputStream fileInputStream = new FileInputStream(new File(

filePath + "/" + "result.json"));

/\*

\* FileInputStream fileInputStream = new FileInputStream(new File(

\* "result.json"));

\*/

byte[] byt = new byte[fileInputStream.available()];

fileInputStream.read(byt);

String s = new String(byt);

Object obj = parser.parse(s);

JSONObject jsonObject = (JSONObject) obj;

JSONArray trends = (JSONArray) jsonObject.get("statuses");

for (int i = 0; i < trends.size(); i++) {

JSONObject jsObject = (JSONObject) trends.get(i);

JSONObject jObject = (JSONObject) jsObject.get("user");

String location = jObject.get("location").toString().trim()

.toUpperCase();

if (!location.equals("")

&& !duplicateLocationMap.containsKey(location))

{

LocationVo locationVo = new LocationVo();

locationVo.setIndex(String.valueOf(i));

locationVo.setLocation(location);

locationVo.setId(jsObject.get("id").toString());

duplicateLocationMap.put(location, locationVo);

}

else {

if (!location.equals("")) {

LocationVo locationVo = duplicateLocationMap

.get(location);

String id = locationVo.getId() + ","

+ jsObject.get("id").toString();

locationVo.setId(id);

duplicateLocationMap.remove(location);

duplicateLocationMap.put(location, locationVo);

}

}

}

Iterator iter = duplicateLocationMap.entrySet().iterator();

while (iter.hasNext()) {

Map.Entry pairs = (Map.Entry) iter.next();

locationList.add((LocationVo) pairs.getValue());

}

for (LocationVo locationVo : locationList) {

System.out.println(locationVo.getId());

}

fileInputStream.close();

}

catch (Exception e)

{

e.printStackTrace();

}

return locationList;

}

public List<HashTagVo> HashTagServices(String filePath) {

List<HashTagVo> hashTagList = new ArrayList<HashTagVo>();

HashMap<String, HashTagVo> duplicateHashTagMap = new

HashMap<String, HashTagVo>();

try {

JSONParser parser = new JSONParser();

FileInputStream fileInputStream = new FileInputStream(new File(

filePath + "/" + "result.json"));

byte[] byt = new byte[fileInputStream.available()];

fileInputStream.read(byt);

String s = new String(byt);

Object obj = parser.parse(s);

JSONObject jsonObject = (JSONObject) obj;

JSONArray trends = (JSONArray) jsonObject.get("statuses");

for (int i = 0; i < trends.size(); i++) {

JSONObject jsObject = (JSONObject) trends.get(i);

JSONObject jObject = (JSONObject) jsObject.get("entities");

JSONArray hash\_trends = (JSONArray) jObject .get

("hashtags");

for (int hash\_index = 0; hash\_index < hash\_trends.size();

hash\_index++) {

JSONObject hashObject = (JSONObject) hash\_trends

.get(hash\_index);

if (!hashObject.get("text").toString().equals("")&& !duplicateHashTagMap.containsKey(hashObject.get("text").toString())) {

HashTagVo hashTagVo = new HashTagVo();

hashTagVo.setHashTag(hashObject.get("text")

.toString());

hashTagVo.setId(jsObject.get("id").toString());

// hashTagList.add(hashTagVo);

duplicateHashTagMap.put(hashObject.get("text")

.toString(), hashTagVo);

}

else {

if (!hashObject.get("text").toString().equals("")) {

HashTagVo hashTagVo = duplicateHashTagMap.get(hashObject.

get("text").toString());

String id = hashTagVo.getId() + ","

+ jsObject.get("id").toString();

hashTagVo.setId(id);

duplicateHashTagMap.remove(hashObject.

get("text").toString());

duplicateHashTagMap.put(hashObject.

get("text").toString(), hashTagVo);

}

}

}

}

fileInputStream.close();

} catch (Exception e) {

e.printStackTrace();

}

Iterator iter = duplicateHashTagMap.entrySet().iterator();

while (iter.hasNext()) {

Map.Entry pairs = (Map.Entry) iter.next();

hashTagList.add((HashTagVo) pairs.getValue());

}

return hashTagList;

}

public List<RetweetCountVo> retweetCountServices(String filePath) {

List<RetweetCountVo> retweetList = new ArrayList<RetweetCountVo>();

HashMap<String, RetweetCountVo> duplicateRetweetMap = new

HashMap<String, RetweetCountVo>();

try {

JSONParser parser = new JSONParser();

FileInputStream fileInputStream = new FileInputStream(new File(

filePath + "/" + "result.json"));

byte[] byt = new byte[fileInputStream.available()];

fileInputStream.read(byt);

String s = new String(byt);

Object obj = parser.parse(s);

JSONObject jsonObject = (JSONObject) obj;

JSONArray trends = (JSONArray) jsonObject.get("statuses");

for (int i = 0; i < trends.size(); i++) {

JSONObject jsObject = (JSONObject) trends.get(i);

String retweetCount = jsObject.get("retweet\_count").toString();

String id = jsObject.get("id").toString();

if (!duplicateRetweetMap.containsKey(retweetCount))

{

RetweetCountVo retweetCountVo=new

RetweetCountVo();

retweetCountVo.setId(id);

retweetCountVo.setRetweetCount(retweetCount);

duplicateRetweetMap.put(retweetCount,

retweetCountVo);

} else {

RetweetCountVo retweetCountVo = duplicateRetweetMap.get(jsObject.

get("retweet\_count").toString());

id = retweetCountVo.getId() + ","

+ jsObject.get("id").toString();

retweetCountVo.setId(id);

retweetCountVo.setRetweetCount(retweetCount);

duplicateRetweetMap.remove(jsObject.

get("retweet\_count").toString());

duplicateRetweetMap.put(jsObject.

get("retweet\_count").toString(), retweetCountVo);

}

}

fileInputStream.close();

}

catch (Exception e) {

e.printStackTrace();

}

Iterator iter = duplicateRetweetMap.entrySet().iterator();

while (iter.hasNext()) {

Map.Entry pairs = (Map.Entry) iter.next();

retweetList.add((RetweetCountVo) pairs.getValue());

}

return retweetList;

}

public static void main(String[] args) {

ClusteringLogic clusteringLogic = new ClusteringLogic();

clusteringLogic.HashTagServices("");

}

}

**5.3.1 LOCATION**

package com.mycompany.vo;

public class LocationVo {

private String location;

private String id;

private String index;

public String getIndex() {

return index;

}

public void setIndex(String index) {

this.index = index;

}

public String getLocation() {

return location;

}

public void setLocation(String location) {

this.location = location;

}

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

}

**5.3.2 HASH TAG**

package com.mycompany.vo;

public class HashTagVo {

private String hashTag;

private String id;

public String getHashTag() {

return hashTag;

}

public void setHashTag(String hashTag) {

this.hashTag = hashTag;

}

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

}

**5.3.3 RETWEET COUNT**

package com.mycompany.vo;

public class RetweetCountVo {

private String id;

private String retweetCount;

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getRetweetCount() {

return retweetCount;

}

public void setRetweetCount(String retweetCount) {

this.retweetCount = retweetCount;

}

}

**5.3.4 RESULT**

package com.mycompany.action;

import java.util.List;

import javax.servlet.ServletContext;

import org.apache.struts2.ServletActionContext;

import com.mycompany.logic.JsonConverter;

import com.mycompany.vo.ResponseVo;

import com.opensymphony.xwork2.ActionSupport;

public class ResponseAction {

private JsonConverter jsonConverter;

private String ids;

private List<ResponseVo> responseList;

public List<ResponseVo> getResponseList() {

return responseList;

}

public void setResponseList(List<ResponseVo> responseList) {

this.responseList = responseList;

}

public String getIds() {

return ids;

}

public void setIds(String ids) {

this.ids = ids;

}

public String execute() {

jsonConverter=new JsonConverter();

ServletContext context = ServletActionContext.getServletContext();

String filePath = context.getRealPath("")+"/result/result.json";

System.out.println("ids ::" + ids);

responseList=jsonConverter.getResponseVos(filePath, ids);

return ActionSupport.SUCCESS;

}

}

**CHAPTER 6**

**IMPLEMENTATION**

Project implementation is the phase where visions become reality. This is the logical conclusion, after evaluating, deciding, visioning, planning, applying for funds and finding the financial resources of a project. Implementation plans require a great deal of preparation and research before they are executed. Implementation is the realization of an application, or execution of a plan, idea, model, design, specification, standard, algorithm, or policy. Here we are going to implement a real time anomaly monitoring system. This monitoring system monitors the social media “Twitter”. In this system, we have three modules for implementing. They are login module, home page module and reports module.

**6.1 LOGIN MODULE**

* This page is used for verification of the analyzer.
* It verifies for the security purpose.
* It has a username and password option.
* It validates the username and password.
* After verifying it, it goes to the next page.
* It is created using html.
* It shows the logout option in the top right corner of the page.

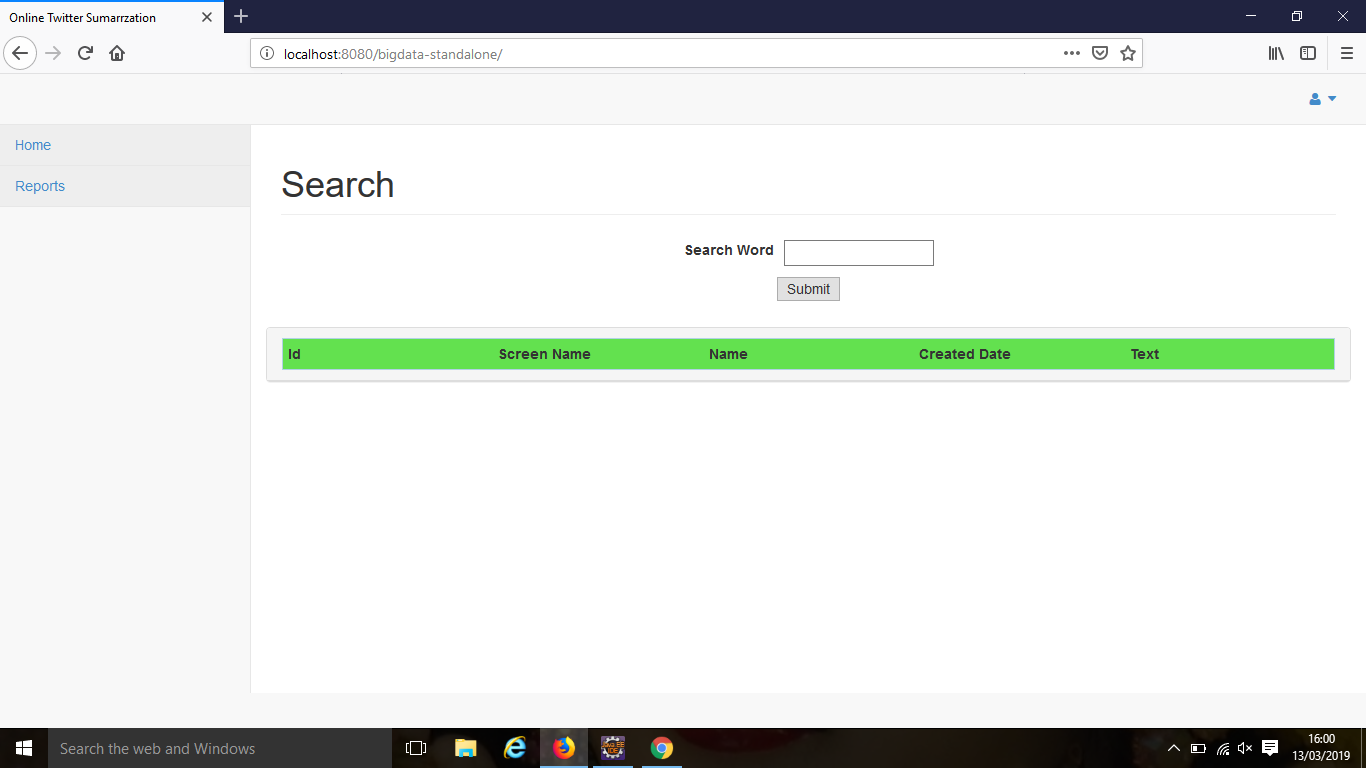
**6.2 HOMEPAGE MODULE**

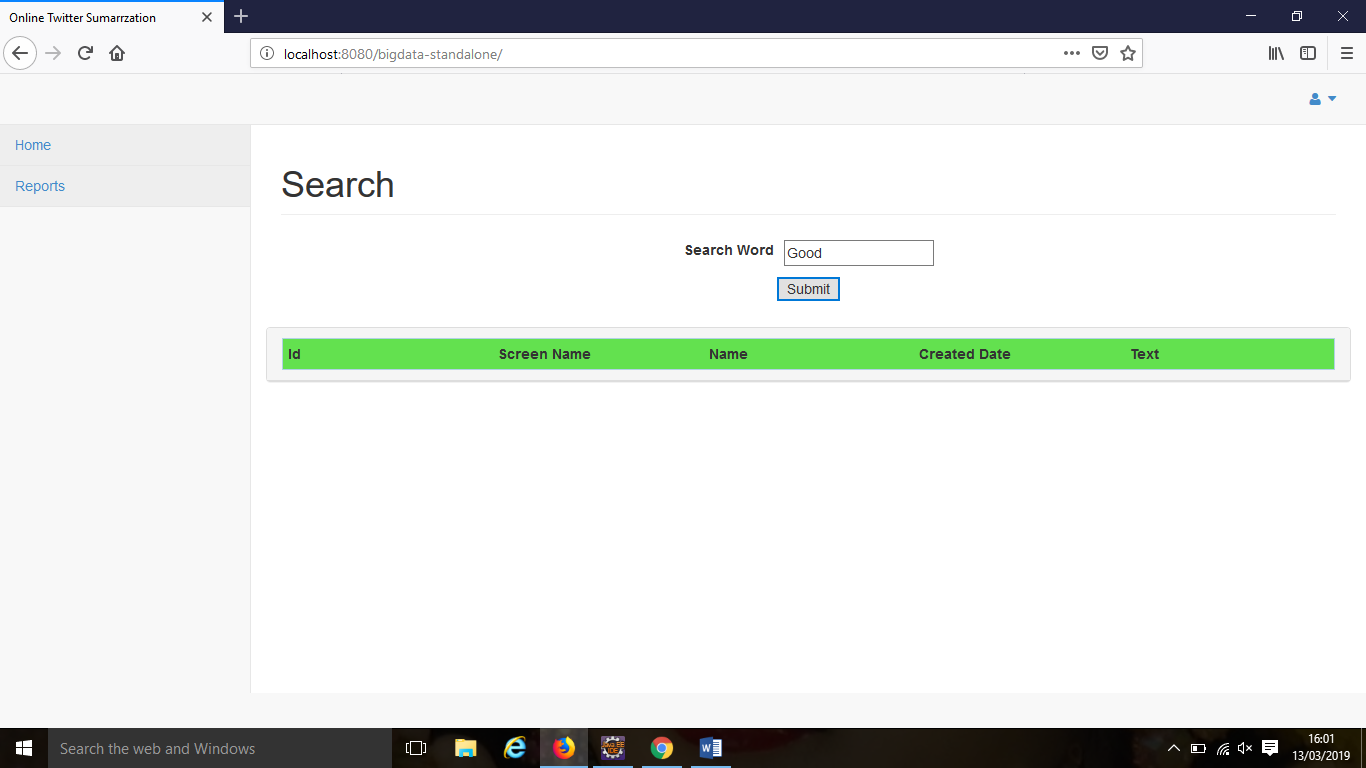
* In this module, searching for the keyword is done.
* Previously, it gets permission for accessing the data from the twitter database.
* Spark collects the data from the database.
* Flume transfers the data to hadoop for further purpose.
* Hive gives the query for search in the HDFS (Hadoop Distributed FileSystem) of the Hadoop.
* Hadoop performs Map Reduce and gives the reduced data.

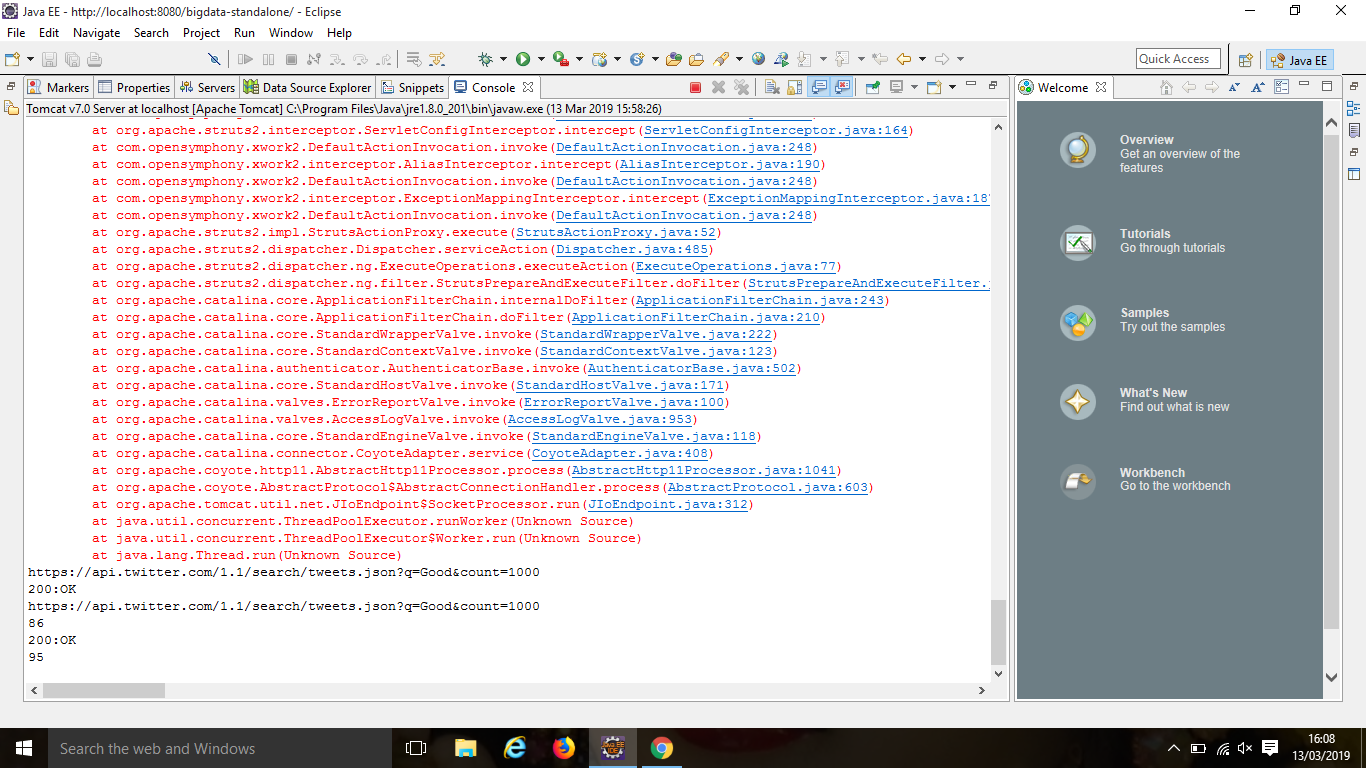
**6.3 REPORT MODULE**

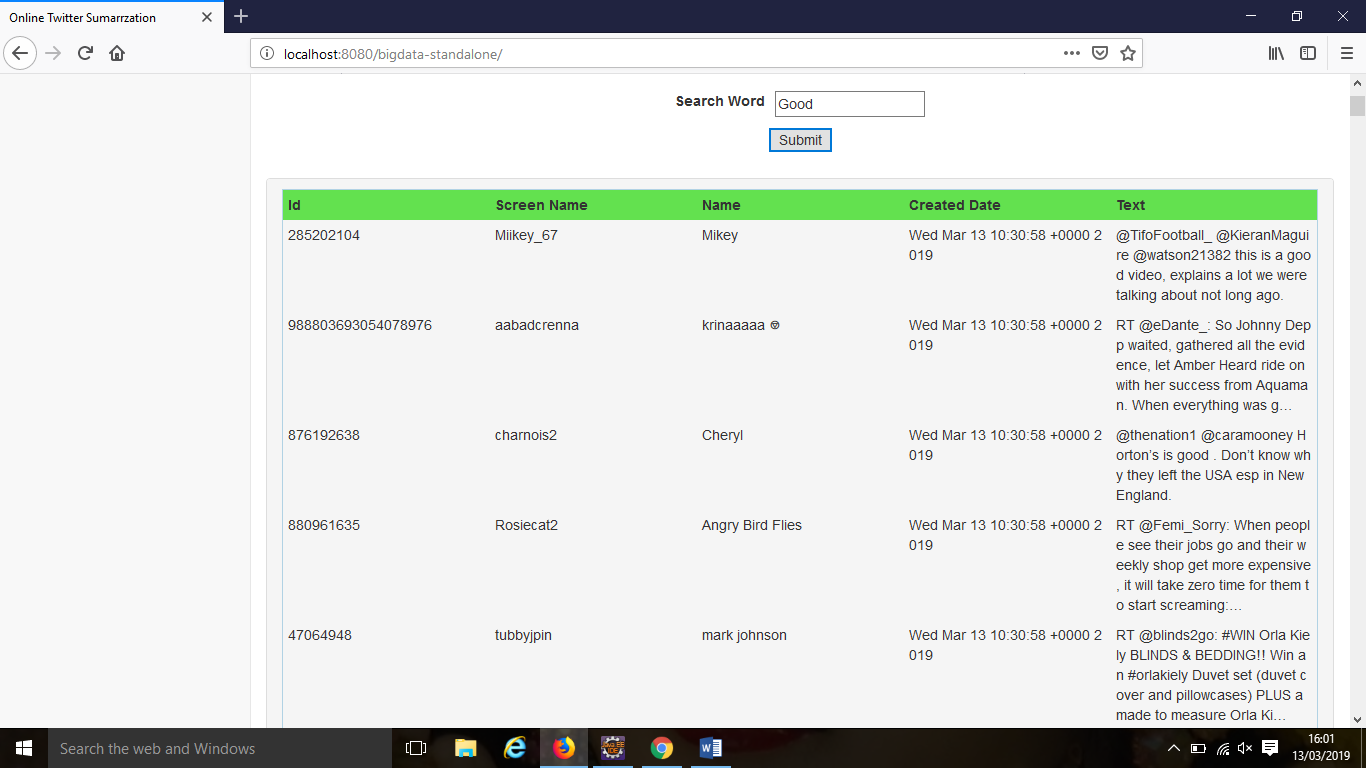
* This module shows the detail about the analyzed keyword.
* In this module, there are three types of reports.
* Reports are
  + - 1. Location
      2. Hash Tag
      3. Retweet Count
* Using the list concept of java, we perform this report module actions.
* The file which has the content, do the filtration for location, hashtag and retweet count.
* It displays the text and details as a result page.

**6.4 SCREENSHOTS**

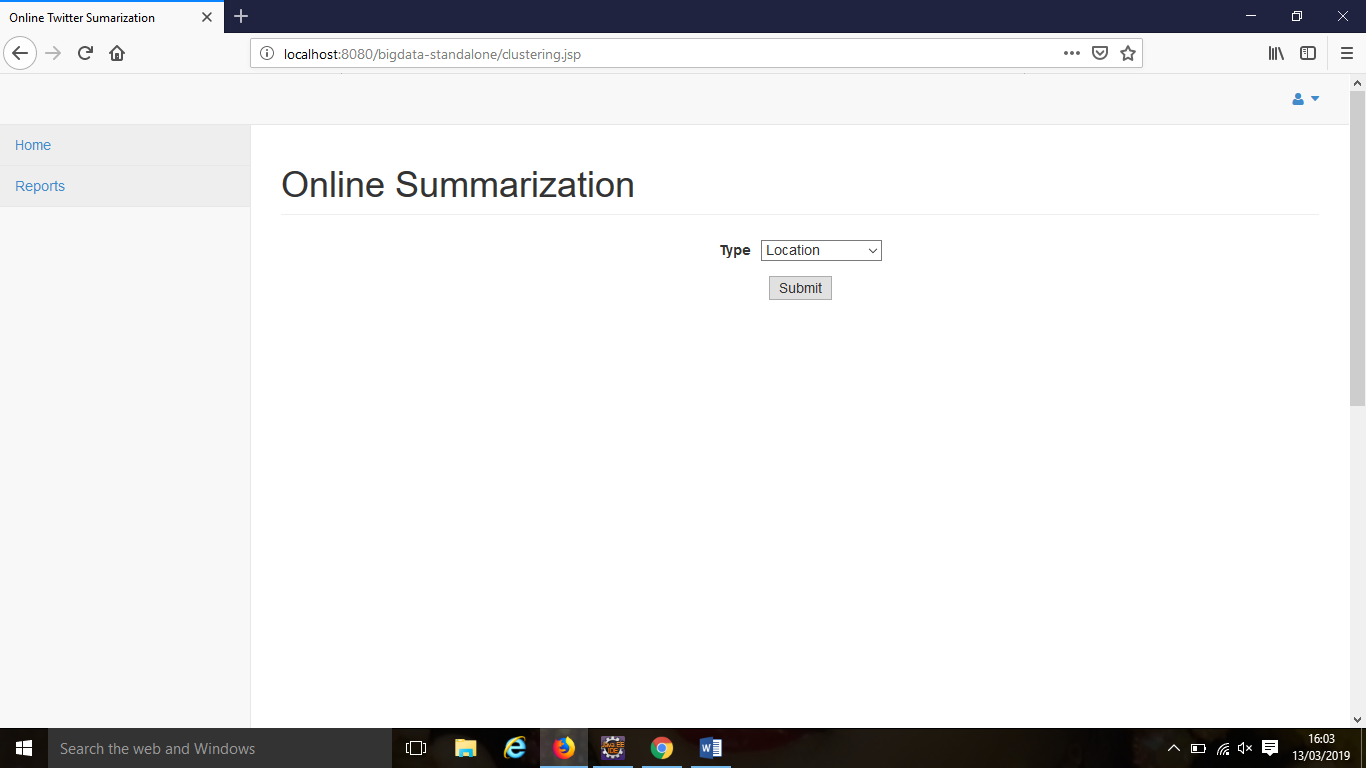


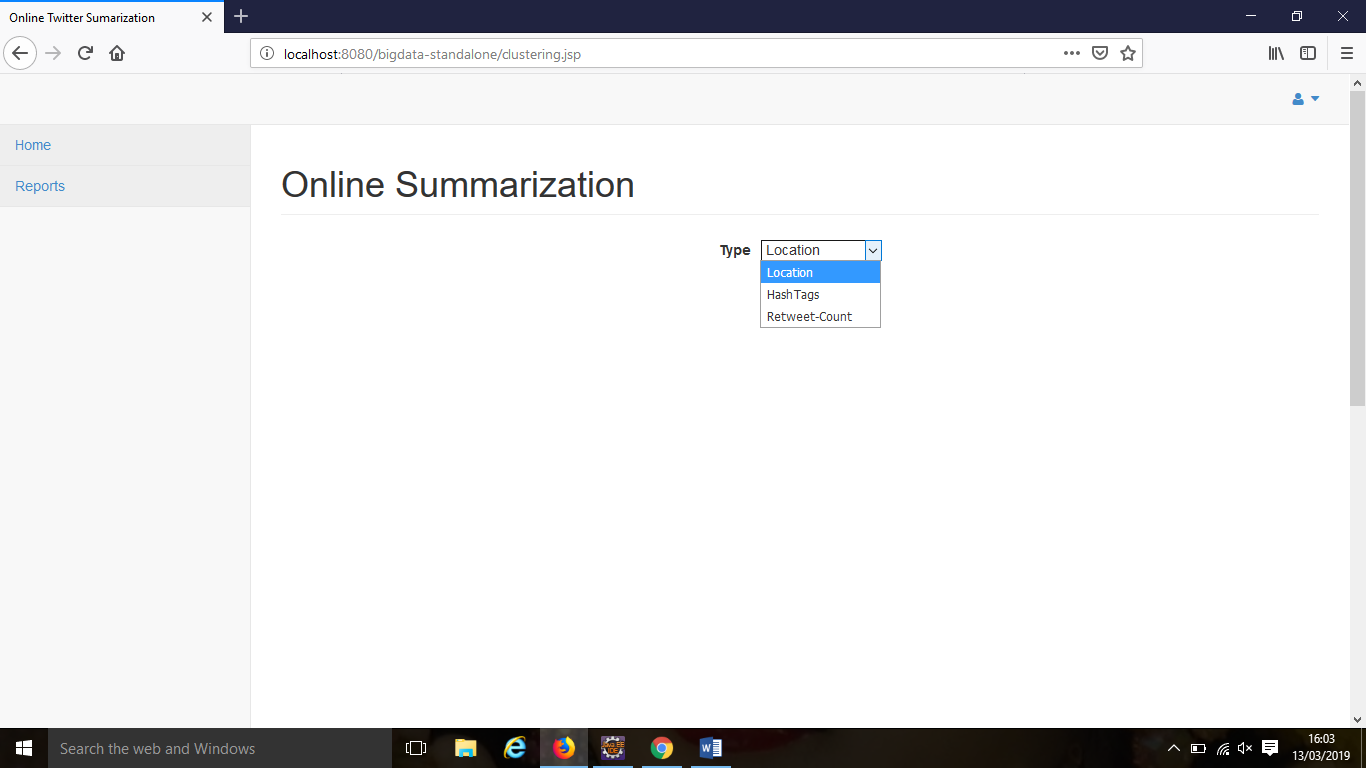


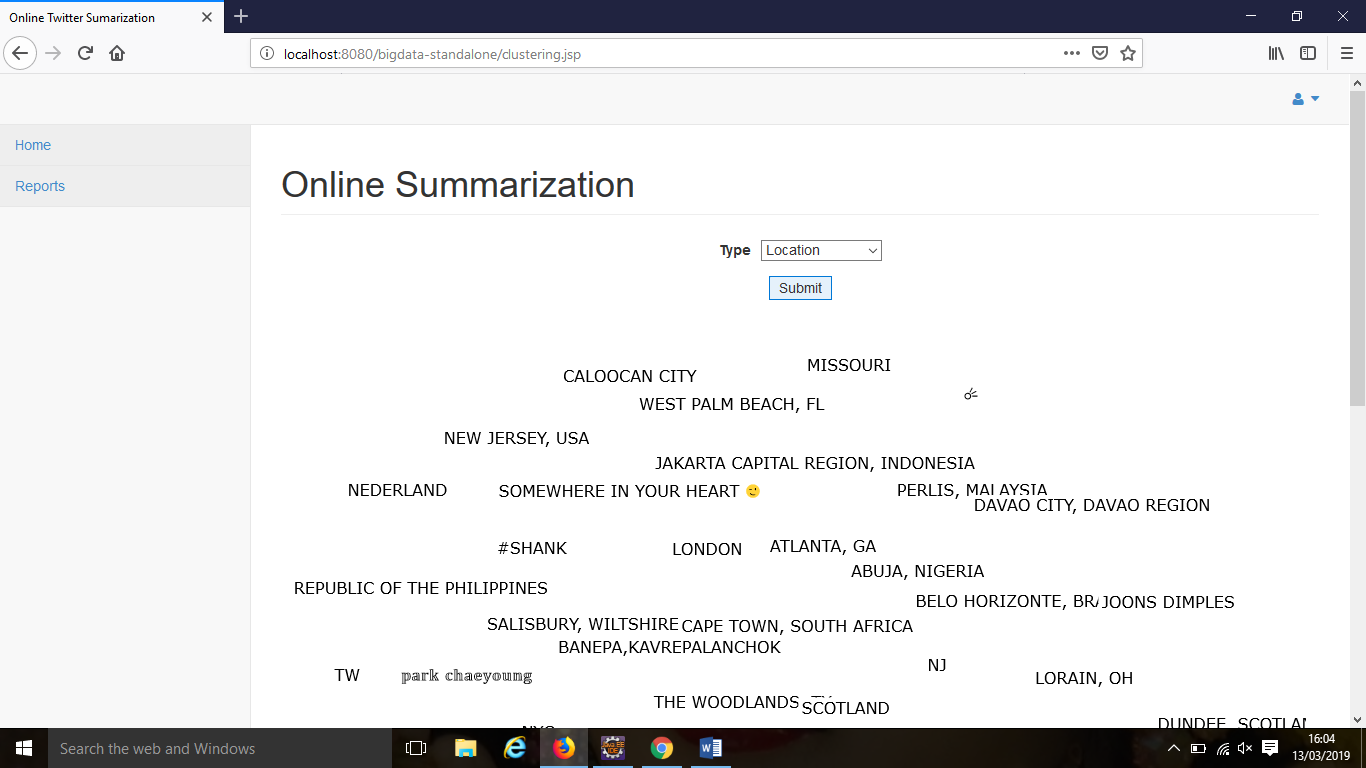


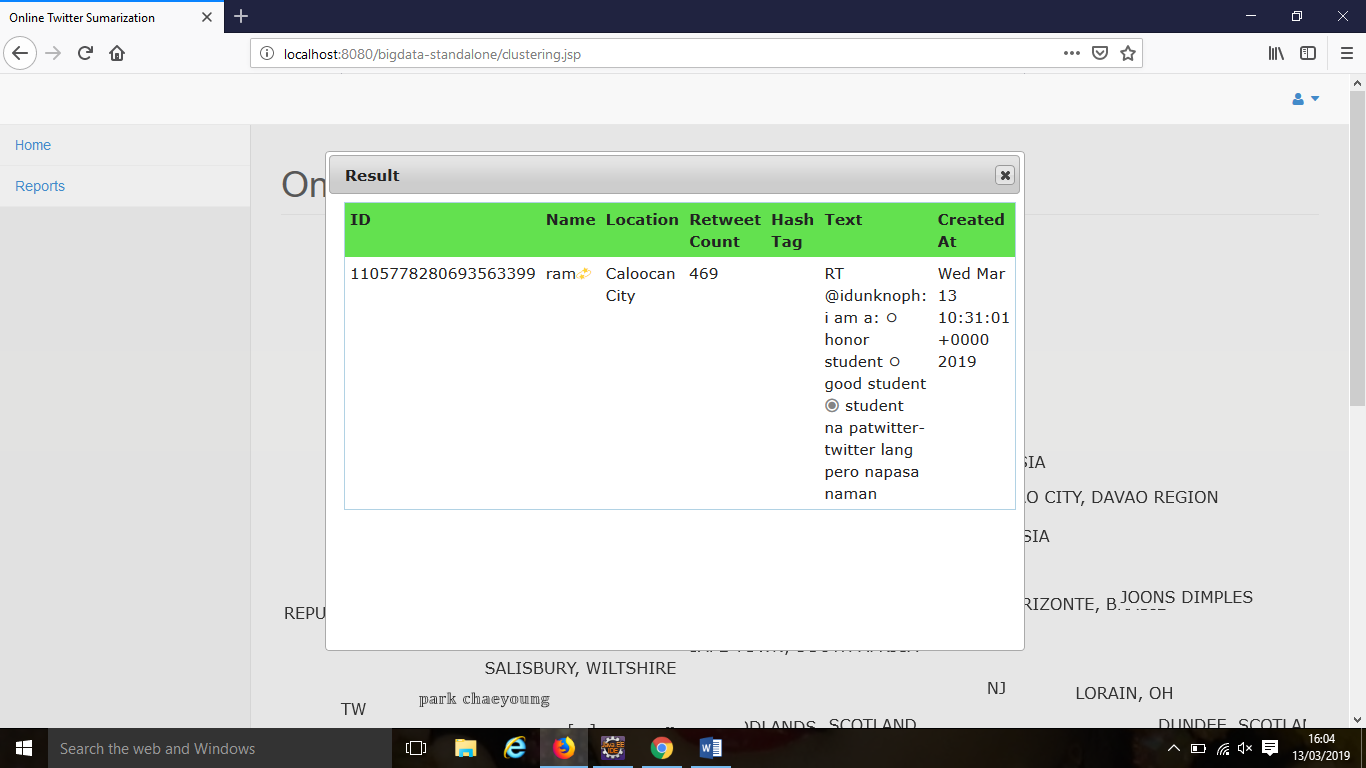


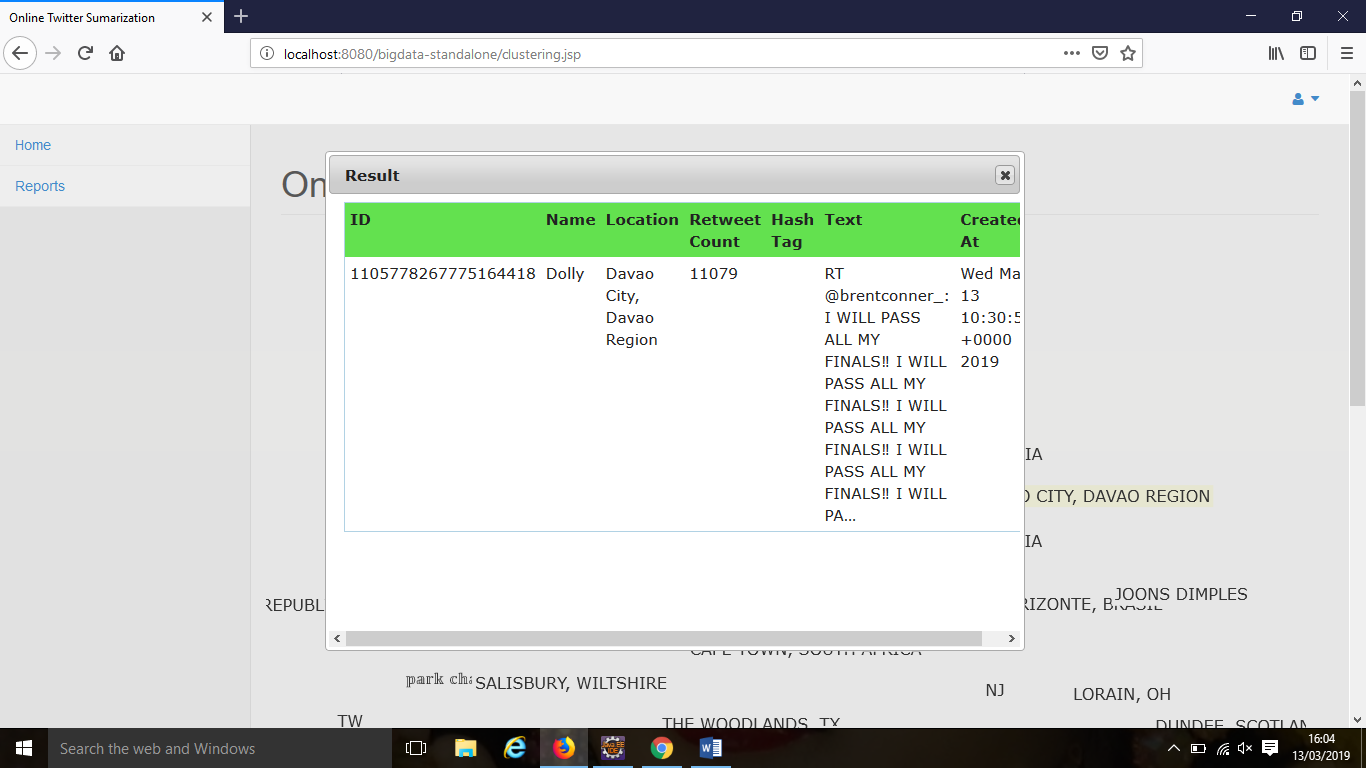


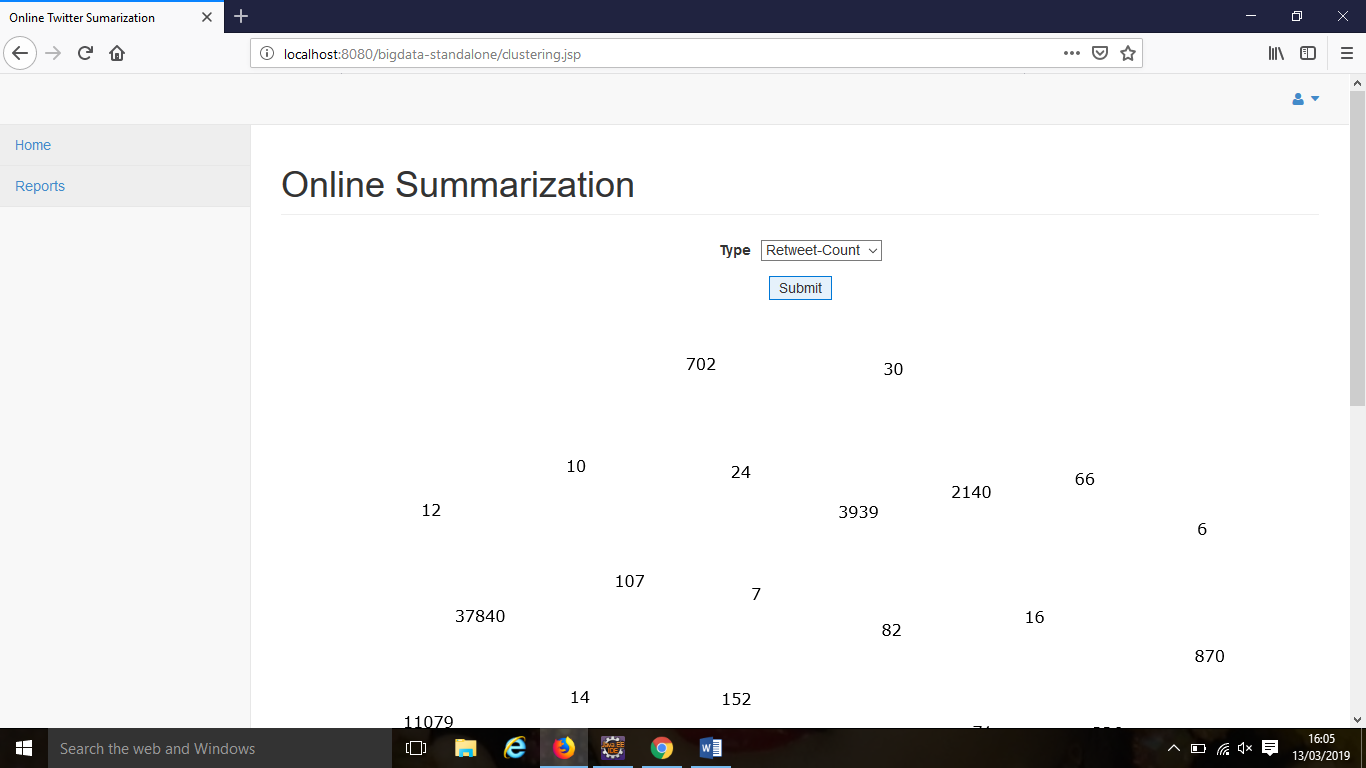


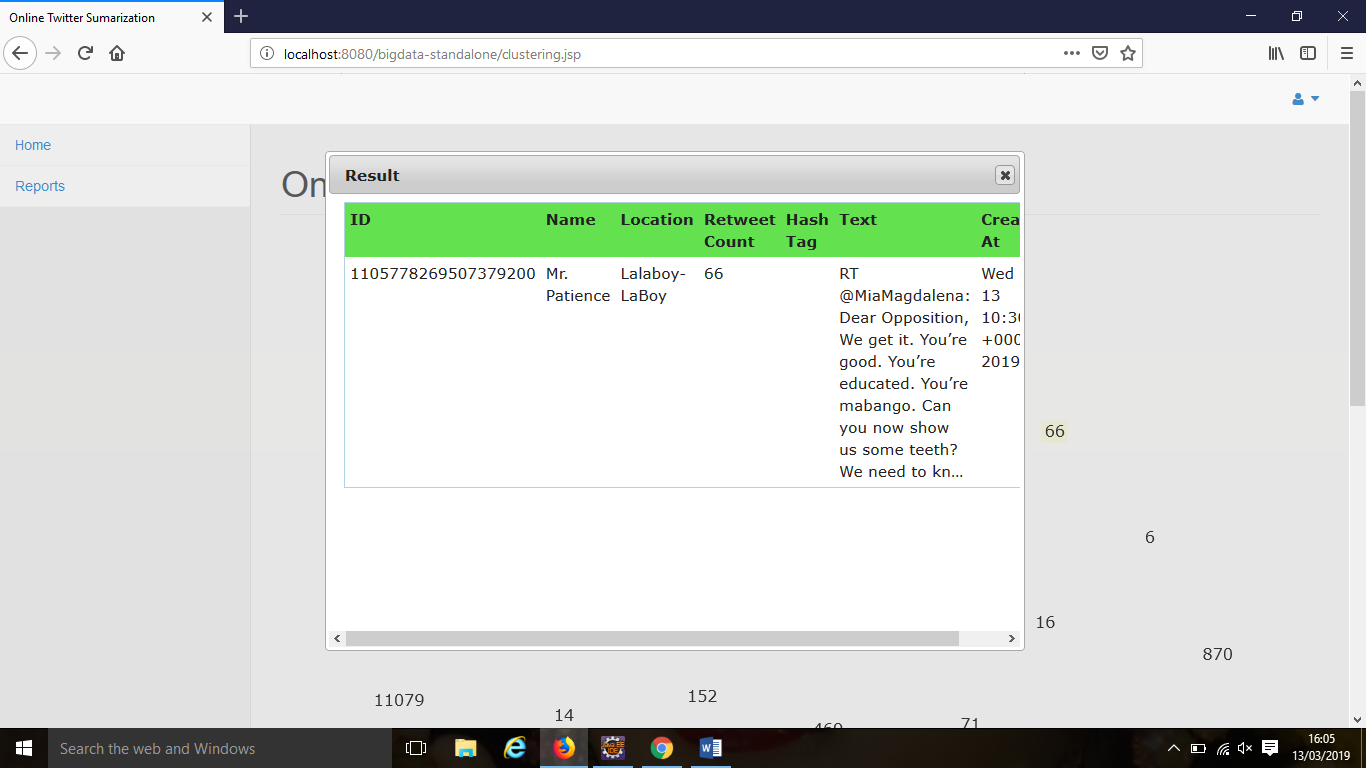


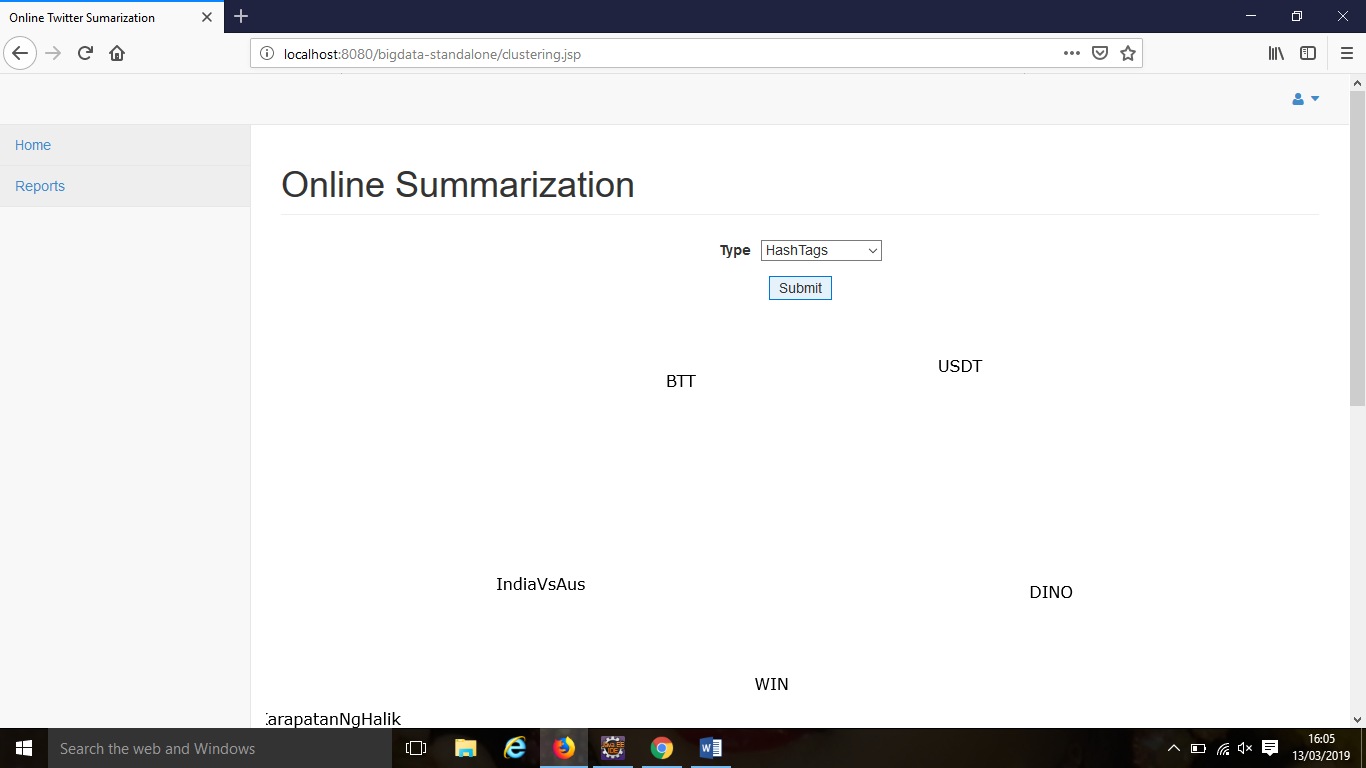


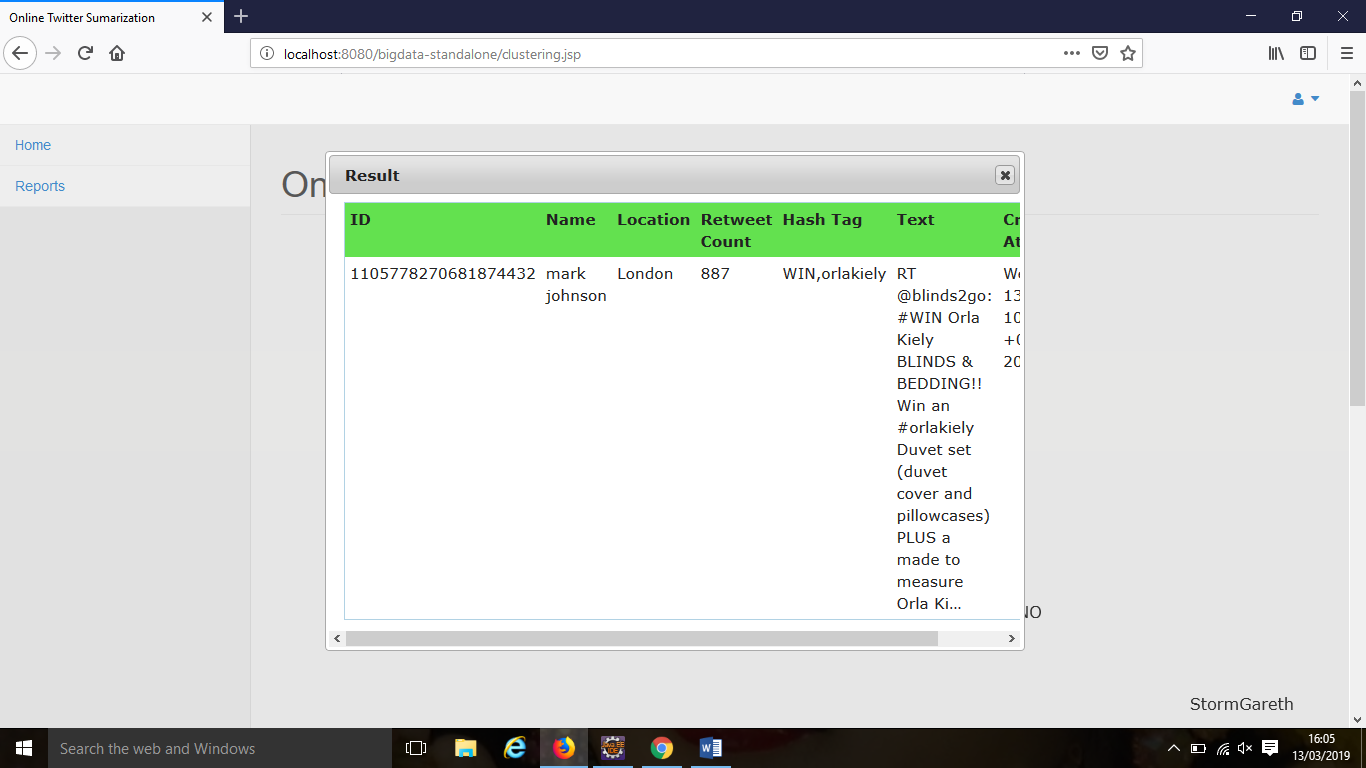


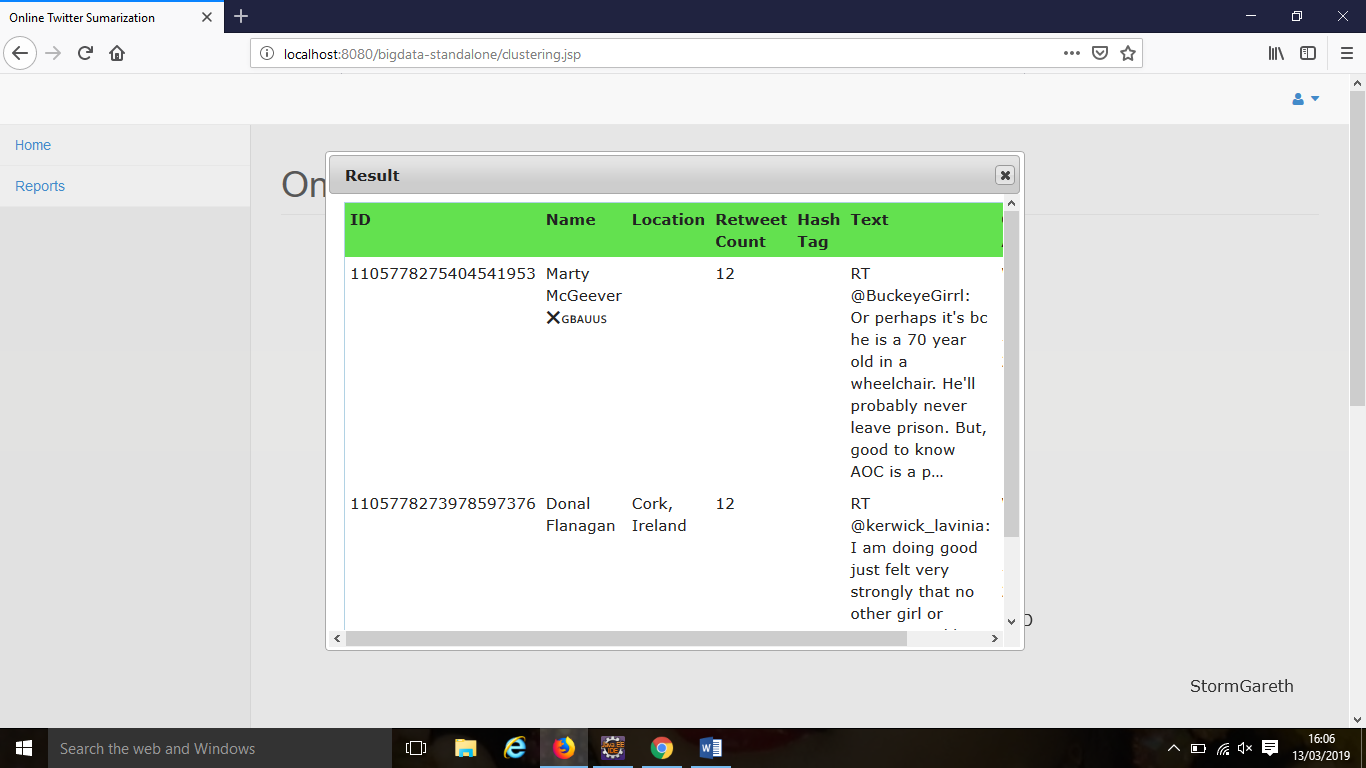












**CHAPTER 7**

**TESTING**

**7.1 SYSTEM TESTING**

System Testing is the testing of a complete and fully integrated software product. Usually software is the only one element of a larger computer based system. Ultimately, software is interfaced with other software/hardware systems. System Testing is actually a series of different tests whose sole purpose is to exercise the full computer based system. Software Testing is an important review of specification, design and coding. The increasing visibility of software as a system element and costs associated with the software failure are motivating forces for well-planned through testing.

Though the test phase is often thought of separate and distinct from the development effort first developers and then testing is a concurrent process that provides valuable information for the development team. There are at least three options for integrating project builder into the test phase.

* Testers do not install project builder, use project builder functionality to compile and source-control the modules to be tested and hand them off to the tester, whose process remains unchanged.
* The testers import the same project or the project that the developer uses.
* Create the project based on the development project but customized for the testers (For example- It does not include support documents, source) who imports it.

**7.1.1 TESTING OBJECTIVES**

There are several rules that can serve as testing objectives. They are,

* Testing is executing a program with the intent of finding an error.
* A good test case is one that has a high probability of finding an undiscovered error.
* A successful test is one of that uncovers the undiscovered error.

If testing is conducted successfully according to the objective stated above, It will uncover the error in the software.

**7.1.2 TYPES OF TESTING**

Testing is the process of executing the program with the intent of finding errors. Testing cannot show the absence of defects, It can only show that software errors are present. The testing principles used are

* Tests are traceable to customer requirements.
* 80% of errors will likely be traceable to 20% of program modules.
* Testing should begin ‘in-small’ and progress towards testing ‘in-large’.

The types of testing are

* Unit testing
* Integration testing
* White box testing
* Black box testing

In our project, we used unit testing and validation testing for checking the system.

**7.2 UNIT TESTING**

**Unit testing**is a level of software testing where individual units/ components of a software are tested. The purpose is to validate that each unit of the software performs as designed. A unit is the smallest testable part of any software. It usually has one or a few inputs and usually a single output. In procedural programming, a unit may be an individual program, function, procedure, etc. In object-oriented programming, the smallest unit is a method, which may belong to a base/ super class, abstract class or derived/ child class. Unit testing frameworks, drivers, stubs, and mock/ fake objects are used to assist in unit testing.

In our system, we tested each module as a separate unit. Each program from the modules are tested and corrected.

**7.3 VALIDATION TESTING**

Validation Testing ensures that the product actually meets the client's needs. It can also be defined as to demonstrate that the product fulfils its intended use when deployed on appropriate environment. Validation Testing is carried out as a whole system to ensure the system/product works well and that meets the requirements of the customer. We validated our project by doing this validation testing. It gives the desired result and satisfies the customer need.

**CHAPTER 8**

**CONCLUSION AND FUTURE WORK**

Social media produces enormous amount of data, now-a-days. Handling and analysing the big data became difficult. So this system provides the effective way for analysing the data and prepares a report. It is easy to find the posts and provide a detailed view about that post. It is very useful to find the location. The future work of this system is enhancing this project by using Artificial Intelligence to find the trending issue and automatically prepare the report without giving the keyword.

**CHAPTER 9**

**REFFERENCES**

1. T. Sakaki, M. Okazaki, and Y. Matsuo, “Earthquake shakes twitter users: real-time event detection by social sensors,” in WWW, 2010.
2. R. McCreadie, C. Macdonald, I. Ounis, M. Osborne, and S. Petrovic, “Scalable distributed event detection for twitter,” in IEEE BigData, 2013.
3. Y. Chen, H. Amiri, Z. Li, and T.-S. Chua, “Emerging topic detection for organizations from microblogs,” in SIGIR, 2013.
4. W. Xie, F. Zhu, J. Jiang, E.-P. Lim, and K.Wang, “Topicsketch: Realtime bursty topic detection from twitter,” in ICDM, 2013.
5. E. Schubert, M. Weiler, and H.-P. Kriegel, “Signitrend: scalable detection of emerging topics in textual streams by hashed significance thresholds,” in KDD, 2014.
6. P. Lee, L. V. Lakshmanan, and E. E. Milios, “Incremental cluster evolution tracking from highly dynamic network data,” in IEEE International Conference on Data Engineering (ICDE), 2014, pp. 3–14.
7. H. Liu, Y. Zhang, H. Lin, J. Wu, Z. Wu, and X. Zhang, “How many zombies around you?” in ICDM, 2013.
8. H. Huang, J. Li, R. Zhang, W. Yu, and W. Ju, “Liveindex: A distributed online index system for temporal microblog data,” in IEEE HPCC, 2015.
9. P. Lee, L. V. Lakshmanan, and E. Milios, “Cast: A context-aware story-teller for streaming social content,” in CIKM, 2014.
10. L. Gao, C. Song, Z. Gao, A.-L. Barab´asi, J. P. Bagrow, and D. Wang, “Quantifying information flow during emergencies,” Nature Scientific Reports, vol. 4, 2014.