**1.**

**Question:** Spark and Smartphone/Watch Application

Implement a smart application with big data analytics related to your project showing the collaboration between Spark and Smart Apps. Implement Twitter Streaming and perform word count on it and publish the results and showcase it in your Smart Phone/Watch Application.

**Description:**

This assignment can be approached by developing two different applications. One is an android application which will run in mobile device in a network and another application will run in IntelliJ application in desktop which will run in the same network.

The android application will start as a socket client where it will create a port and assign to the application and will wait for incoming connection. Then if we run IntelliJ application with the IP address same as mobile, then the application will connect to mobile and will send the twitter word count data output to mobile in form of streaming until the application was terminated.

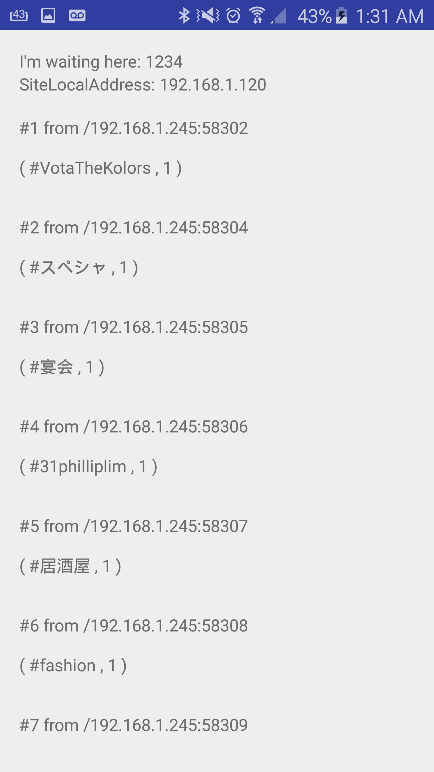
**IntelliJ Code TwitterStreaming.scala Explanation:**

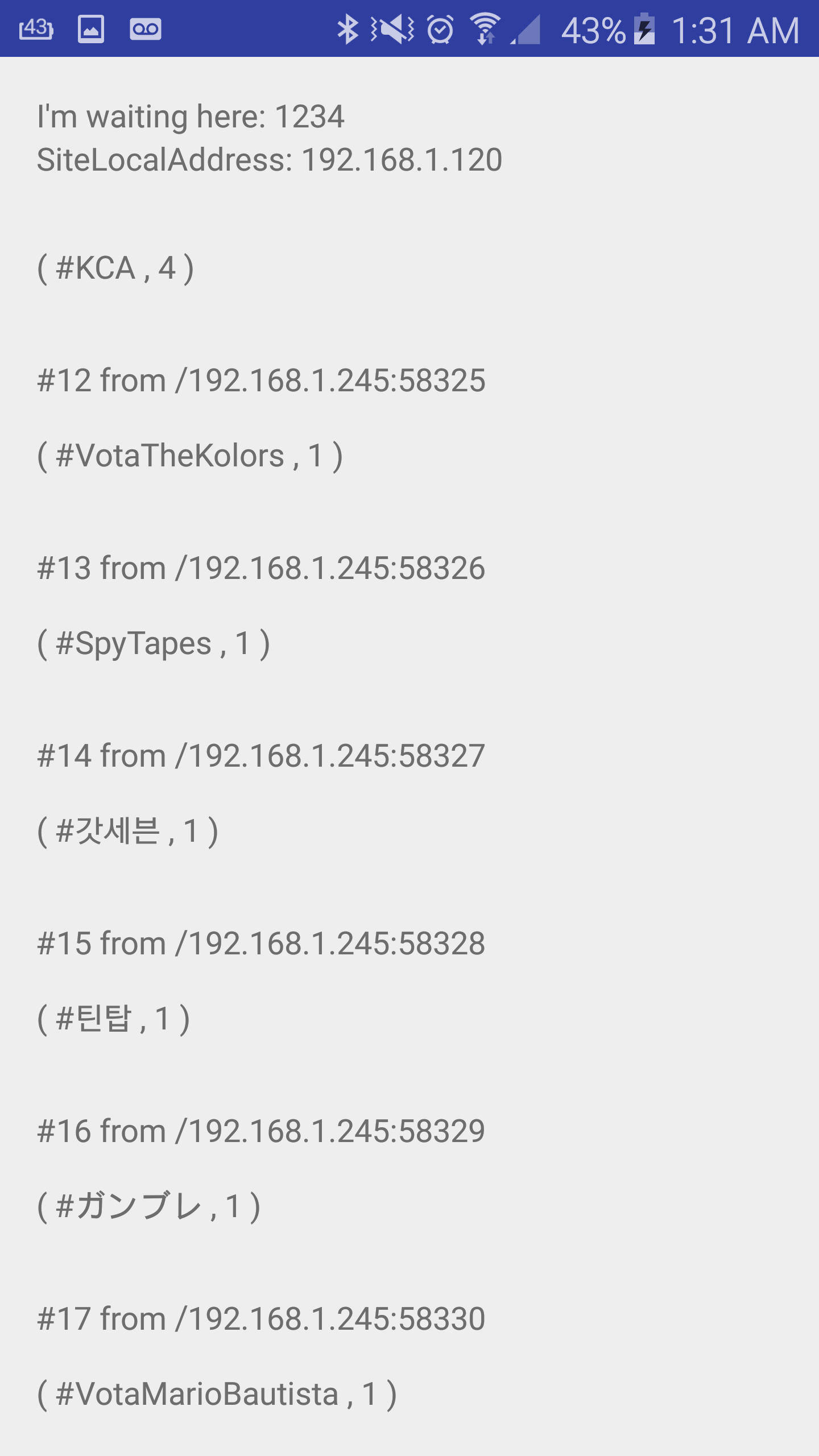
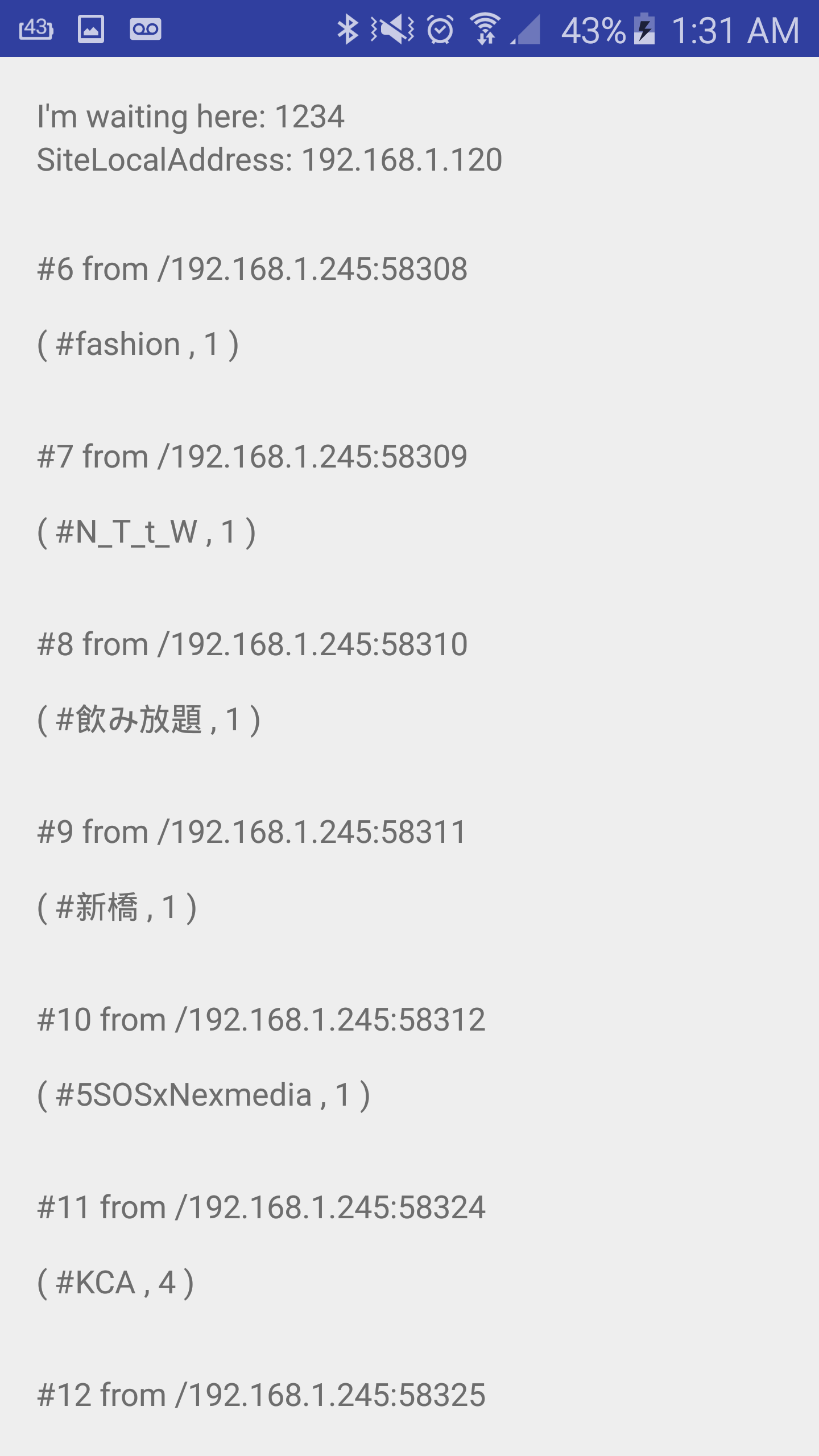
*//Setting the API keys of twitter to variables***val** Consumer\_Key=**"XJPlyOuAo7tC8YBxE1GDiTq8n"  
val** Consumer\_Secret=**"Zeyv8qYR9u5wmOqvBStqLV8HMA1Qvh3R9d3vDj02gfpTfAzrgs"  
val** Access\_Token=**"143736380-2VpESDsrWMcOipe7zLIMBr0ncHwnDbMxjckpU7nT"  
val** Access\_Token\_Secret=**"jEdqt83lwfuDO1rPFpUyGmS4OexheFFsru7CZEH9iqnss"***//Assigning the API keys*System.*setProperty*(**"twitter4j.oauth.consumerKey"**, Consumer\_Key)  
System.*setProperty*(**"twitter4j.oauth.consumerSecret"**, Consumer\_Secret)  
System.*setProperty*(**"twitter4j.oauth.accessToken"**, Access\_Token)  
System.*setProperty*(**"twitter4j.oauth.accessTokenSecret"**, Access\_Token\_Secret)  
  
*//Creating Spark Configuration***val** sparkConf = **new** SparkConf().setAppName(**"STweetsApp"**).setMaster(**"local[\*]"**)  
*//Creating streaming context***val** ssc = **new** StreamingContext(sparkConf, *Seconds*(2))  
**val** stream = TwitterUtils.*createStream*(ssc, None, filters)  
stream.print()  
*//Getting Hashtags stream data based on criteria like "#"***val** hashTags = stream.flatMap(status => status.getText.split(**" "**).filter(\_.startsWith(**"#"**)))  
*//Getting top hashtags for 30 seconds***val** topCounts30 = hashTags.map((\_, 1)).reduceByKeyAndWindow(\_ + \_, *Seconds*(30))  
 .map{**case** (topic, count) => (count, topic)}  
 .transform(\_.sortByKey(**false**))  
*//Searching for each RDD and sending the values to SocketClient.scala*topCounts30.foreachRDD(rdd => {  
 **val** topList = rdd.take(10)  
 *println*(**"\nPopular topics in last 30 seconds (%s total):"**.format(rdd.count()))  
 topList.foreach{**case** (count, tag) => *println*(**"%s (%s tweets)"**.format(tag, count))}  
 topList.foreach{**case** (count, tag) => SocketClient.*sendCommandToRobot*(**"\n( "** + tag +**" , "**+ count+**" )"**)}  
})  
*//Starting streaming context*ssc.start()  
*//Stop the streaming context either by terminating or if the timeout happens for 1000000 seconds*ssc.awaitTerminationOrTimeout(100000)

**IntelliJ Code SocketClient.scala Explanation:**

**def** sendCommandToRobot(string: String)  
{  
 **try** {  
  
 *//Assigning the IP value of the mobile to address variable* **lazy val** address: Array[Byte] = *Array*(192.toByte, 168.toByte, 1.toByte, 120.toByte)  
 **val** ia = InetAddress.*getByAddress*(address)  
  
 *//Create port address* **val** socket = **new** Socket(ia, 1234)  
  
 *//Creating out variable and print the stream to* **val** out = **new** PrintStream(socket.getOutputStream)  
  
 *//Printing the string and sending to Mobile Client* out.print(string)  
 out.flush()  
 out.close()  
 socket.close()  
 }

Screenshots for Question 1:





**2**

**Question:** Spark ML Lib Application

Perform a machine learning algorithm with the Twitter Streaming data to categorize each Tweet

1) Training datasets: Collect different categories of Tweets related to your project.(Categories can be based on HashTags /Subjects etc.)

2) Test data: The upcoming twitter stream

**Description:**