Real-time Event and Context Analytics Using Audio and Video Streaming

An Automatic Event Analytics Application using Stream Analysis

Project Report - 4

Team #3

Team Members:

Nikhita Sharma - 22

Ganesh Taduri - 24

Aparna Pamidi - 18

Harsha Komalla - 11

1. PROJECT OBJECTIVES:

Objective:

In an improving educational world it's been very difficult to keep track of the student's attendance for a lecturer with their limited time for the class. Though we have many other systems to keep track of the attendance they can be easily manipulated. Therefore our objective of this project is to develop a system which keeps track of the attendance automatically using face recognition algorithms with the help of live video.

Significance:

"Automatic Attendance Management System using Face Recognition" is useful not only to keep track of the attendance system but also to reduce paperwork, save time, eliminate duplicate data entry and errors and auto generate various types of reports of class for student attendance. We can also implement this system to other general public meetings to generate statistical information of the members attended and also to keep track of the security surveillance of the meeting.

Features:

It is a mobile as well as a static application, where a user can record a live video and generate statistical reports out of it. The various reports that can be generated are attendance report, total revenue generated out of the meetings, etc. It also provides features to detect an unusual activity in public gathering.

2. APPROACH

Data Sources:

We collected a sample classroom videos and seminar videos for training the model. The various sources for the data are:

- Youtube.com
- Ted.com

Analytical Tools:

We used latest tools and various API's for developing this project.
Major tools used:

- Spark
- Storm
- Kafka

Major API's used:

- Microsoft Cognitive Services
- Califai

Analytical Task:

By using this system we can process a live video of a seminar and can generate various interesting and useful reports through some analytical functions, Attendance report, revenue generated from a seminar, reports about males vs females in a hall, are the major analytical tasks for this project.

Expected Inputs/Outputs:

Input:

A scanned video of a classroom or a seminar hall. Input can also be a live video through a camera in hall.

Output:

Expected output will be the analysis report of attendance of a class/auditorium .And if it's an auditorium it will identify if it is music or a speech or a any other auditorium.

Algorithms:

We used a few Machine Learning algorithms provided by Spark MLlib during training the model to classify the data. The algorithms are:

1. Decision Tree
2. RandomForest

3. RELATED WORK:

With the increasing number of videos, their necessity to analyse and extract useful information there were many frameworks being introduced for this purpose. We studied around 13 research papers related to Big data Analytics on Streaming data and some important frameworks used to achieve big data processing. Some of them frameworks researched were: Storm, Spark, Kafka, Pipeline61 Framework, Heron, SummingBird, S4 framework, etc. And their applications Human Emergency Mobility Simulator, Traffic Flow Prediction. Among these Storm, Spark, Heron, SummingBird are a few that support real time streaming processing and provides results in microseconds and seconds. Based on the requirements of the application the select of a frameworks depends. In our application we used the Storm, Spark and Kafka. Where the Spark is used for model prediction, Kafka as messaging system to the consumer which our case is the storm, and Storm is used for the predicting the new test features data.

Open Source Projects:

- https://github.com/opencv-java/face-detection
- https://github.com/levackt/face-detection

Literature Reviews:

- http://klresearch.org/IJMSTM/papers/v2i3_2.pdf
- http://blog.mashape.com/list-of-10-face-detection-recognition-apis/
- http://ieeexplore.ieee.org.proxy.library.umkc.edu/stamp/stamp.jsp?tp=&arnumber=6227479

4. APPLICATION SPECIFICATION:

System Specification:

Software Architecture:

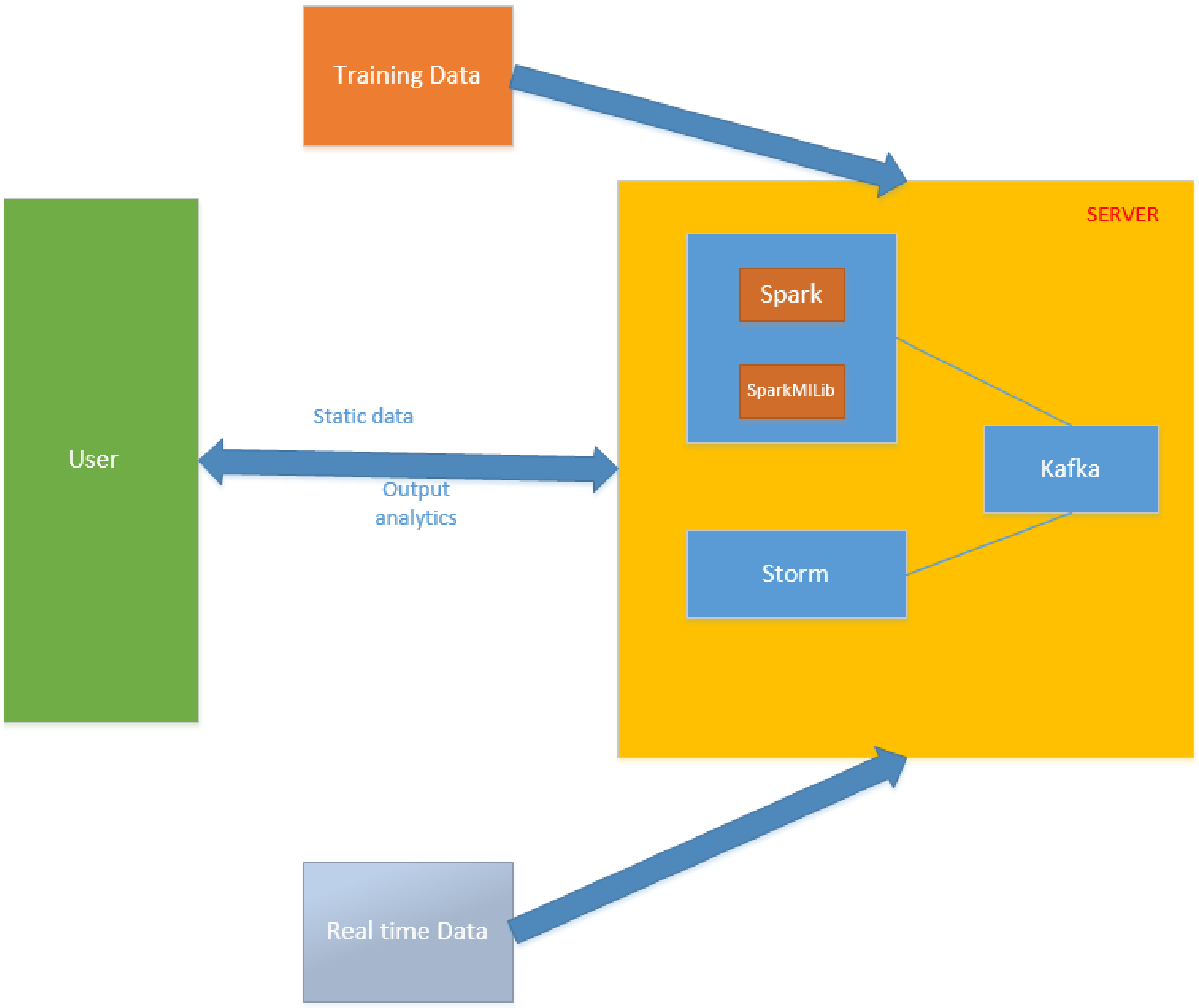


Fig 1: Architecture Diagram

<u>Software Architecture:</u>

Spark:

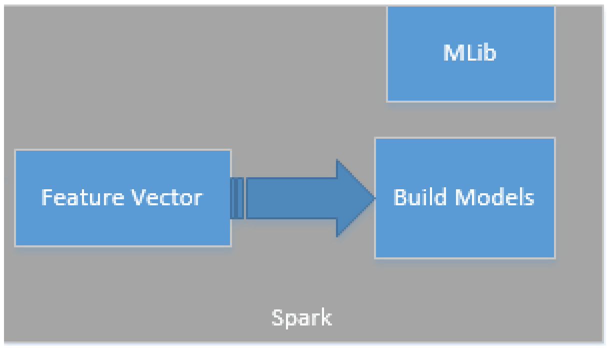


Fig: Spark Architecture

Storm:

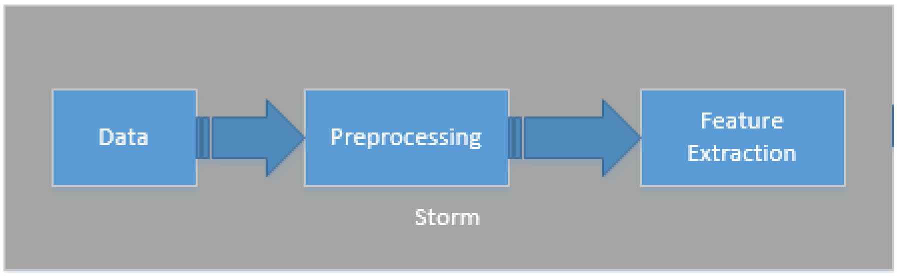


Fig: Storm Architecture

Kafka:

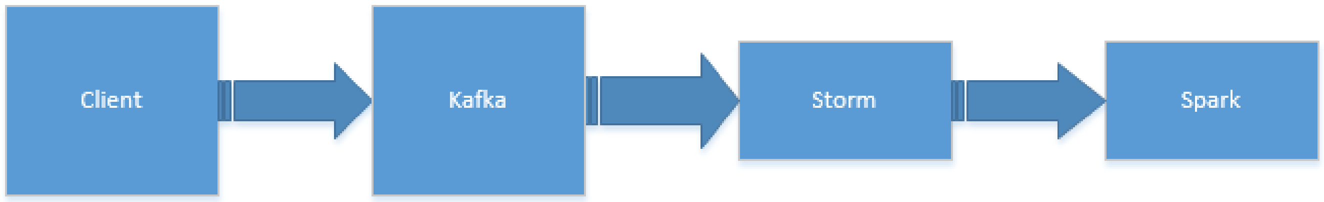


Fig: Kafka Architecture

<u>System Flow For finding number of people:</u>

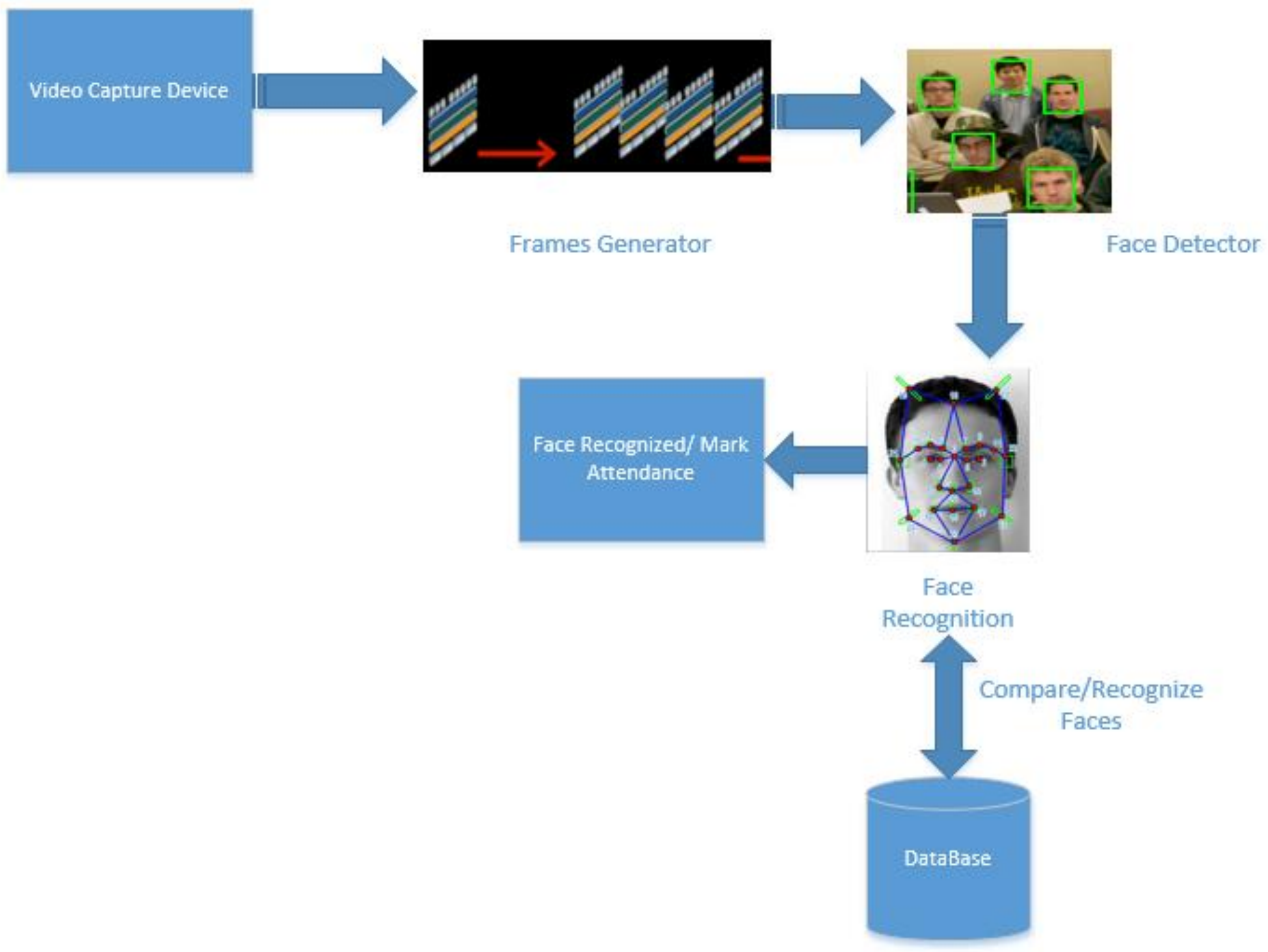


Fig 2: System Flow Diagram

Class Diagram:

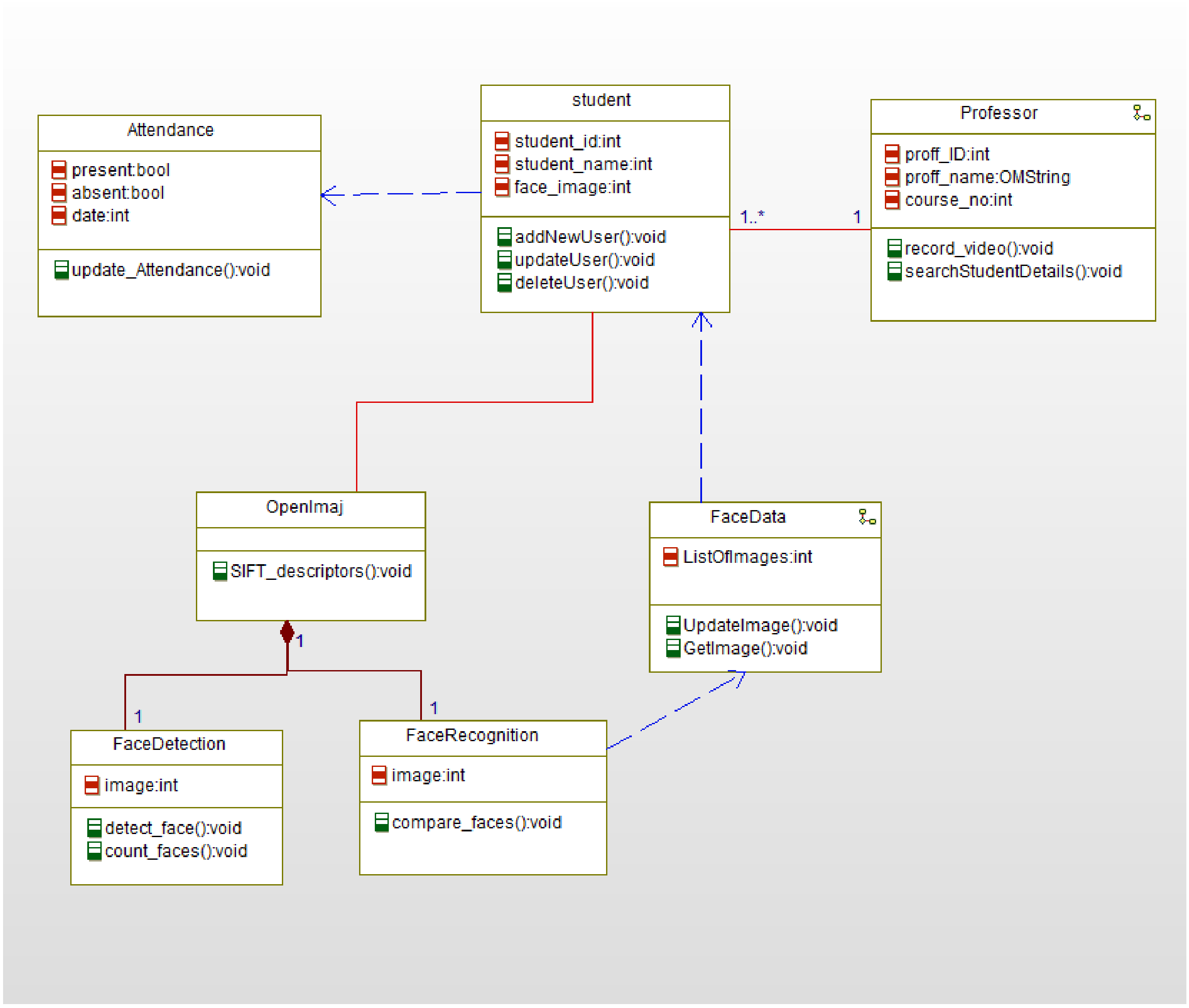


Fig 3: Class Diagram

Sequence Diagram of Whole System:

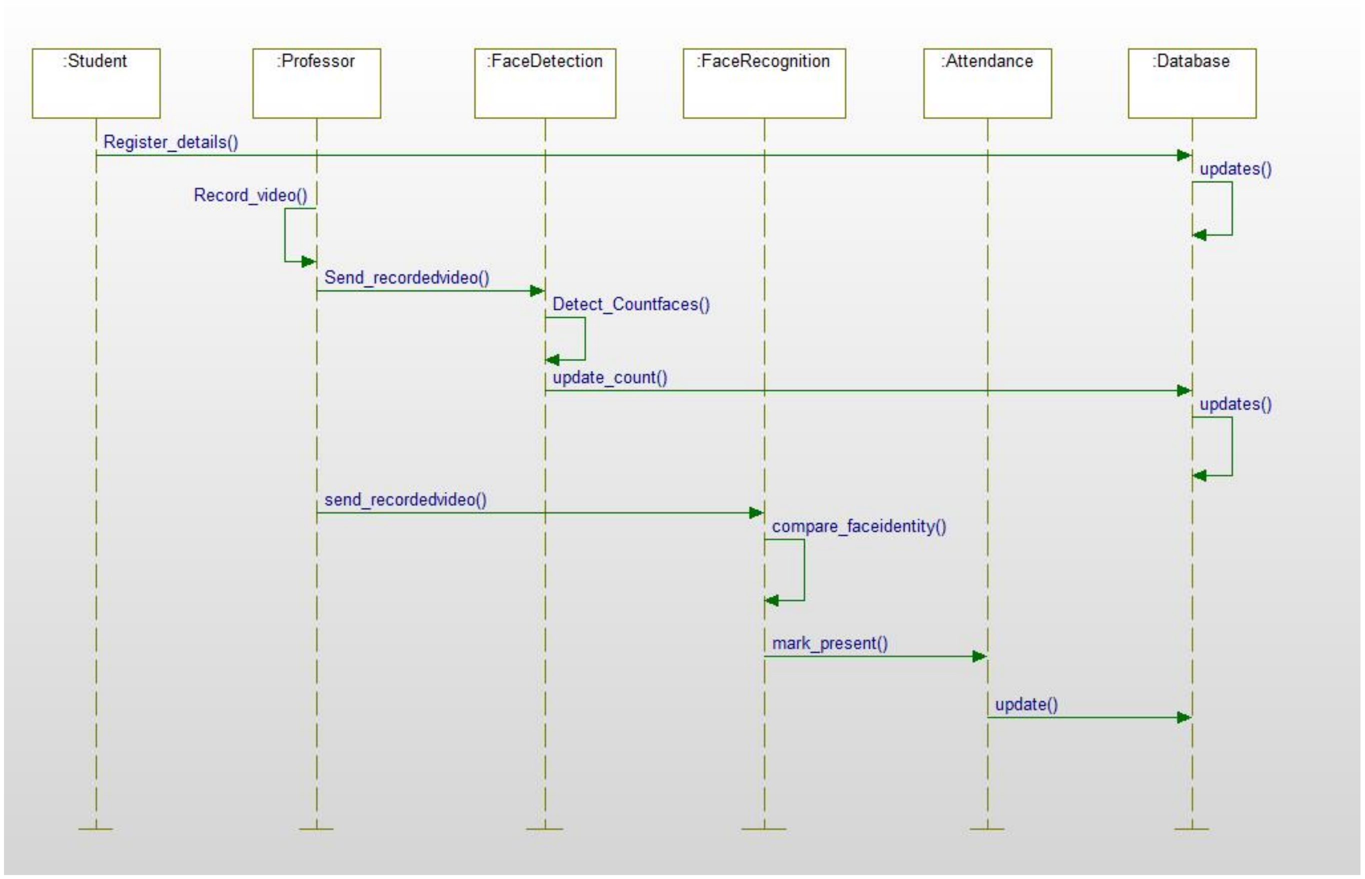


Fig 4: Sequence Diagram

Activity Diagram:

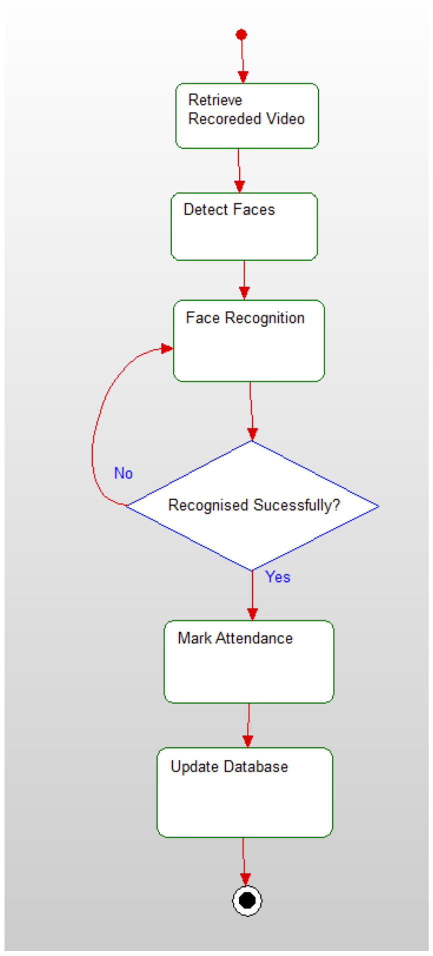


Fig 5: Server Activity Diagram

Sequence diagram:

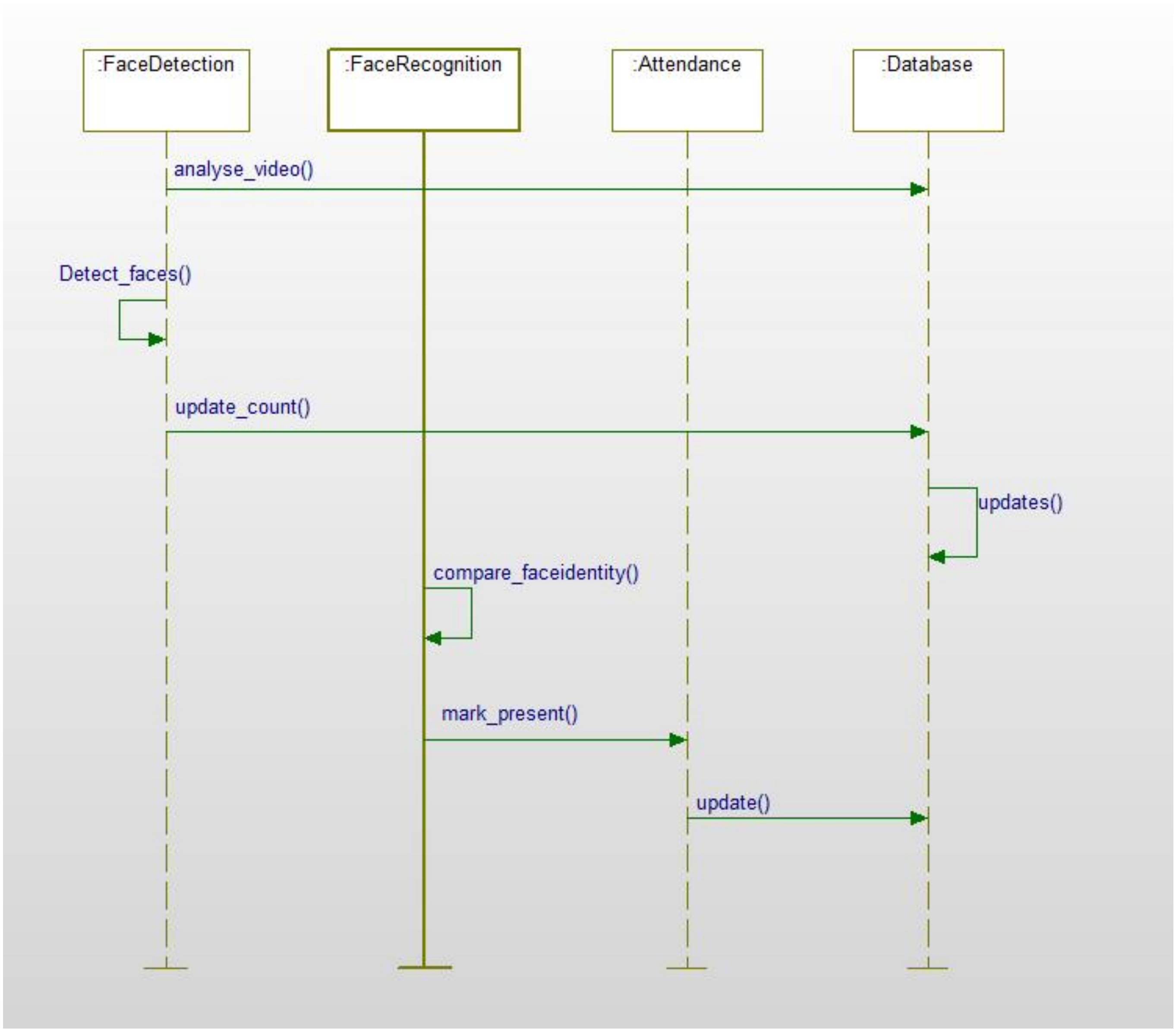


Fig 6: Server Sequence Diagram

Design of Mobile Client:

Activity Diagram:

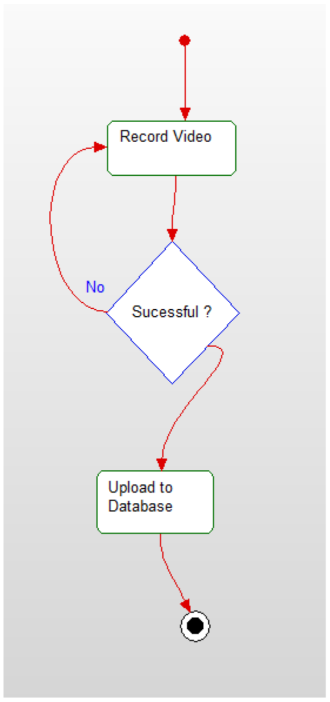


Fig 7: Client Activity Diagram

Sequence diagram:

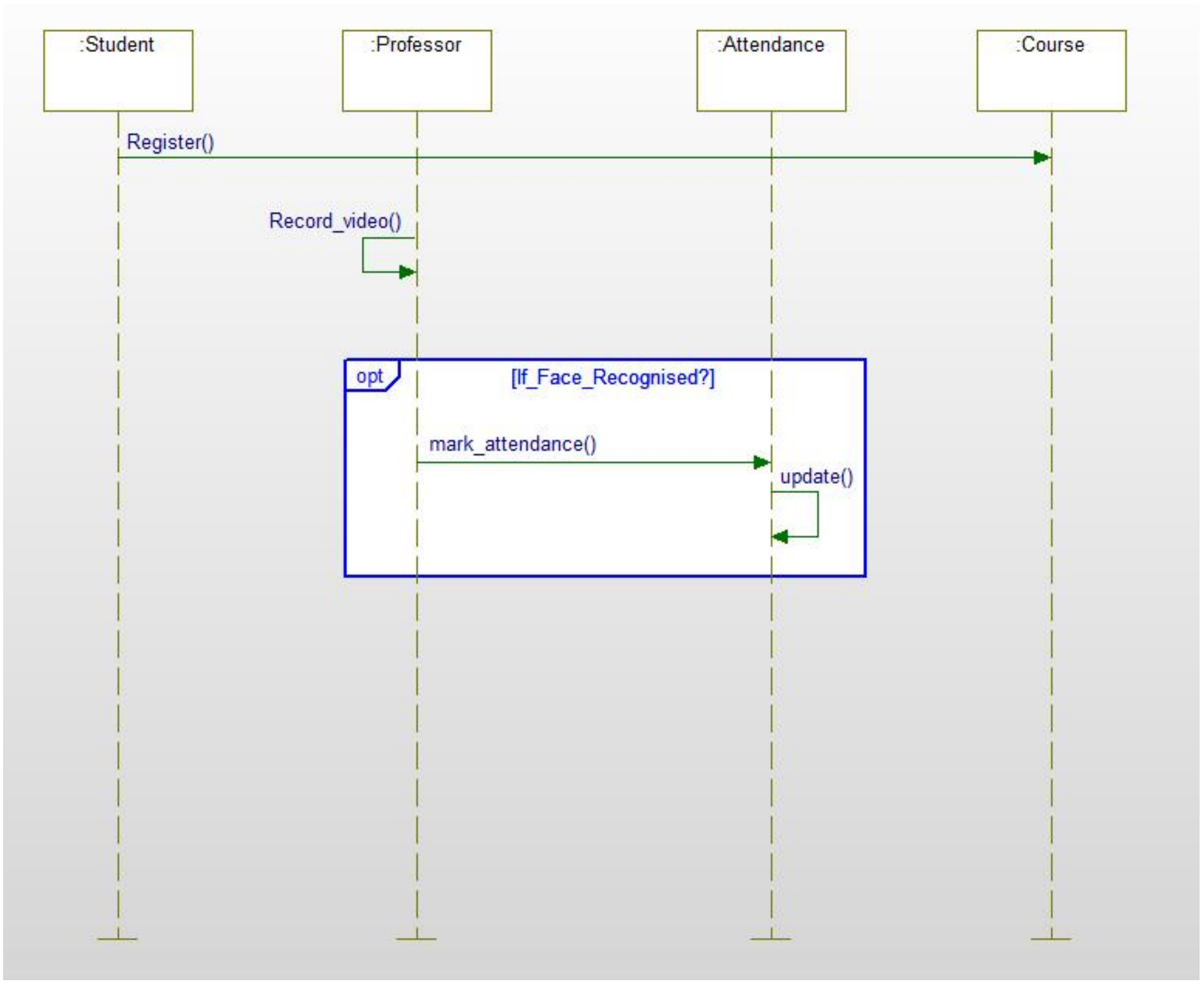


Fig 8: Client Sequence Diagram

<u>Existing Services/APIs:</u>

- Microsoft Cognitive Services - Face API
- Califai - Object Detection

5. IMPLEMENTATION AND DOCUMENTATION:

Related Work:

Implementation of Solution In four Phases:

PHASE 1:

Implementation of Sever using Spark and Storm:

Used OpenIMAJ (which has an inbuilt face detection and recognition algorithms) at the server side to detect the number of faces present per frame and also compare the detected faces with the list of images present in the training data, recognise them.

Implementation of Mobile Client:

The Client UI is developed using the HTML, CSS, Angular JS and Java Script, which provides the user with a option to record a live video to know the number attendees and recognise each individual person.

PHASE 2:

In this phase we tried to implement whole work flow in various phases. The system first takes input from a client which is a video data/image data and then sends this data to Storm topology where various bolts performs various operations for extracting the features out of it. The Kafka architecture (pub-sub system) used as an intermediate broker to transfer the data from client to Storm Topology. We divided the whole architecture into different parts and executed them separately. After identifying the results we thought we could integrate them to build a better architecture according to the results. Below is the detailed explanation of various sub tasks.

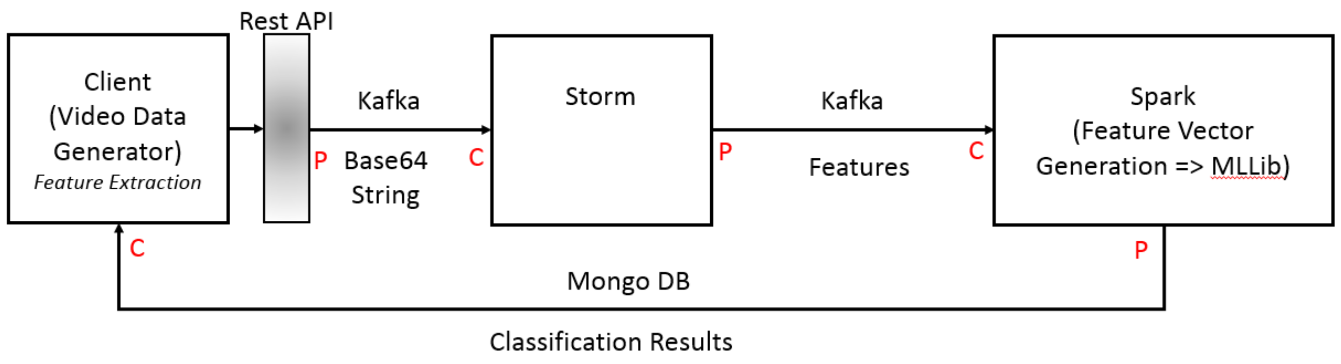


Fig: Overview of Work-Flow Architecture

Implementation of Kafka Producer Consumer System:

The main goal of this task is to send video data along with some meta data from kafka producer to Consumer. We recorded a classroom video and sent it to kafka producer and then published it into a broker by creating a topic. And we created a consumer which subscribed for a particular topic and extracted data from the broker. Before publishing the data from kafka producer into broker we encoded our video into Base64 format and then published it. Therefore the data received by a Kafka consumer was also in Base64 format so it decodes it and then stores into local machine.

As a part of our project in addition to video we also need individual students photos for face recognition. Therefore we created a model which takes photo from client, decodes it into Base64 format then publish it into Kafka broker from consumer. And then Consumer receives the data and decodes it into regular image and stores it locally.

The main motivation of this task is to send video data and image data from client to Storm for feature extraction implementing pub-sub architecture of kafka as a broker between client and Storm.

Implementation of Rest API Services to store and retrieve data to/from Mongodb:

The main aim of this task is to store results in a database to use it for training the model and analysing the results. We chose MongoDb to store the data because the data can be accessed through API calls. We couldn't implement this task totally as the results from a model are not completely perfect yet. Therefore to test the process we have built code to send and store messages from Tomcat Client to kafka Consumer using the publish and subscribe process in Kafka for storing messages in JSON format in MongoLabs.

Implementation of Rest API Services to send data from Client to Kafka Producer:

In this task we used REST API services to send data from client to kafka producer. The approach behind this is to enable real time data processing. In the first task mentioned above we used static data. In order to implement the task to process real time data(video/image), we could use REST API services. Therefore we built a model which accepts real time streaming data using Kafka producer, and consumer (pub-sub) system and passes the data to Storm using REST API calls.

Implementation of Storm Topology:

The goal of this task is to process the data received from Kafka producer and perform a specific action by implementing the storm Topology(collection of Spout and bolts). Here, the Kafka consumer in the first process which receives data from the Kafka broker and acts as a Storm Spout. In addition to this we created some bolts to perform a specific action. Ideally this action should extract features from a video/image and send it to Spark for further machine learning and model generation process. But due to serialization issue we couldn't implement the feature extraction part on videos in Storm. Therefore to gain some knowledge on how the Storm topology works we executed word count on a sample data using a bolt.

Note: In this phase we could execute these subtasks separately successfully but due to some technical issues we could not integrate them together to build a whole model. We are trying to identify the bugs in integration as of now. Therefore, we are planning to complete the integration part in next phase.

PHASE 3:

In this phase we connected all the tasks that were executed in phase 3. We followed the work flow that is shown in the diagram below. The main work flow is divided into two parts .One for training and one for testing. For training the model we collected videos files, extracted audio from them and then extracted features from audio files, sent these features from client to kafka producer and saved the features into a vector file, generated the model, and pushed the model into mongoDB. For testing the model we sent audio features from client to Kafka spout and classified the data using three bolts in the storm topology and save the results into Mongodb. The whole process is explained in detail in various sub tasks below.

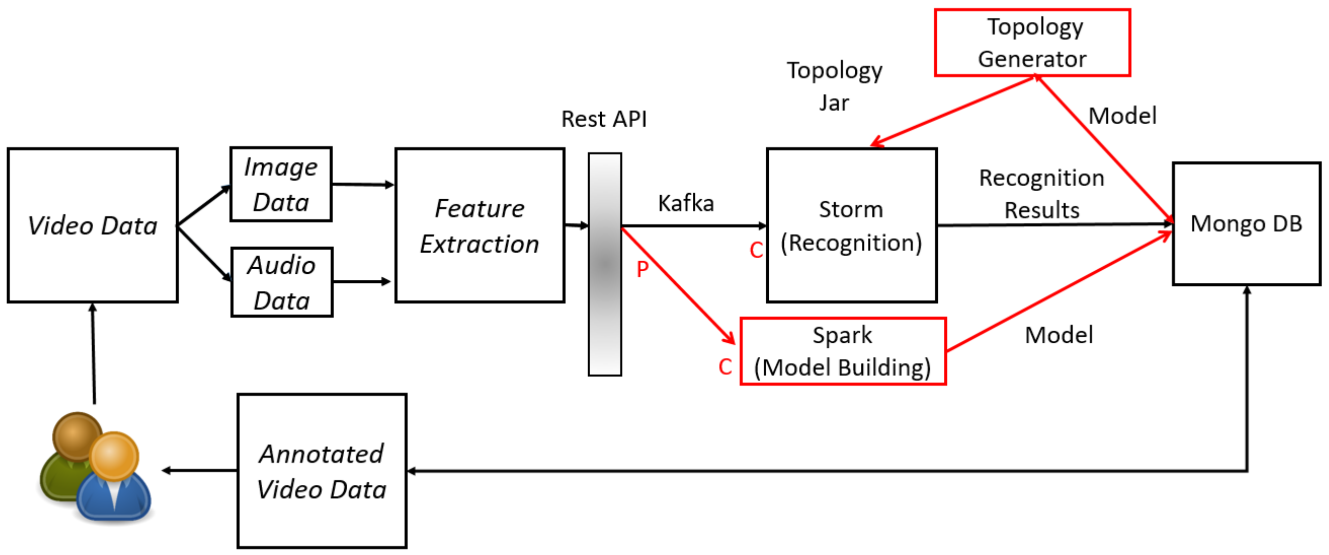


Fig: WorkFlow Architecture for Phase3 Implementation

A. **TRAINING THE MODEL:**

   This part mainly deals with generating/training the model.

1. **Extraction of audio from Video:**

   In this task we extracted audio from video for extracting features. we collected several video files online mostly music videos and ted talks. our aim is to build a model which recognises if an auditorium is a music auditorium or a general talk and count the number of audience in it. As our model requires two classes we classified the collected video into music data(music videos) and speech data(Ted Talks). we almost collected 25 videos for this task

2. **Extraction of features from audio:**

In this task we extracted features from the audio data we extracted from video files. We used Jaudio for the feature extraction. we classified the features into music features and speech features. We selected a set of six features which suited best for classifying our data. The features we selected are meanZCR, meanMFCC, meanSpectralRollOff, meanPeakValue, meanRMS, meanCompactness. The features were then pushed to kafka producer and then kafka consumer receives the data and saves them in a vector file which is later used as input for the model generation.

3. **Generating the model and pushing it into Mongodb.**
   In this task we trained the model using the features file saved at kafka consumer which was received from kafka consumer. We implemented decision tree algorithm for generating the model. We used Spark MLlib for implementing the decision tree. As we have only two classes for the decision tree the model generated was pretty simple and easy. This model is then pushed to mongoDB which can be accessed later during the testing phase.

B. TESTING THE MODEL:

This deals with the testing part of the decision tree that is generated in the training phase. For the testing purpose we considered a video data as input file.

1. **Send extracted features to storm topology:**
   Here we extracted the audio features from the given input file. The extracted features are passed from the client to storm (kafka-consumer) through kafka.

2. **Topology Generation:**
   The storm creates a topology which includes a spout and two bolts, the spout acts as a data source i.e it receives the features to be tested through kafka. The bolts perform the actual task in our topology so, we created two bolts as we classified the labels of decision tree in two different classes each bolts the depicts one class. The bolts receives the input features data from the spout processes them through the decision tree generated after the processing the bolts outputs a class label to which the given input data belongs to, in our scenario the class labels are either Music or Speech.

3. **Saving the results:** Once the bolt outputs the respective class label (Speech or Music) it is stored to the MongoDB along with the features of a particular input file.

PHASE 4:

In this phase we modified our topology that we generated in phase 3 and extended it to three classes. One for audio, one for speech and one for all other types. Also, we rebuilt our decision tree model accordingly with new training data and pushed it into mongoDB. The major new tasks we implemented for this phase are:

1. **Annotating the video with results.**

   In this task we gave a real time video or a static video to the system and as the the output we annotated the video with the results accordingly. The major steps involved are
   - Counting the number of people using face detection with the help of OpenCV libraries.
   - Annotating the video with number of people in it.
   - Extracting the results from mongoDB to identify the type of the audio.
   - Annotating the video with result of the classification.

2. **Validation of the model:**
   - After we tested our model using the topology generated we collected the results and analyzed them by plotting a confusion matrix. And, with the help of the confusion matrix we calculated the accuracy of the model as well as recall and precision.

RESULTS AND EVALUATION :

Confusion Matrix:

Actual/Predicted	Music	Speech	Other	
Music	2	0	2	4
Speech	0	2	0	2
Other	0	0	2	2
	2	2	4	8

Accuracy:

tp + tn / total = 6/8 = 0.75 (75%)

Precision (Music):

tp / (tp + fp) = 2/2 = 1

Precision (Speech):

tp / (tp + fp) = 2/2 = 1

Precision (Other):

tp / (tp + fp) = 2/4 = 0.5

Recall (Music):

tp / (tp + fn) = 2/4 = 0.5

Recall (Speech):

tp / (tp + fn) =2/2 = 1

Recall (Other):

tp / (tp + fn) = 2/2 = 1

F-measure:

2 * (Precision * Recall) / (Precision + Recall)

F-measure Music = 0.66
F-measure Speech = 1
F-measure Other = 0.66

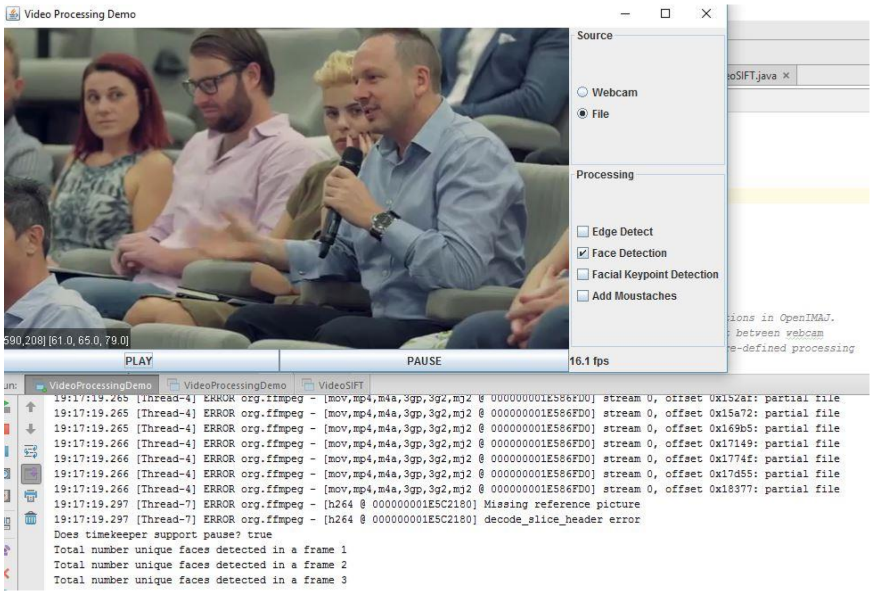
Error Rate:
(fp + fn) / total = 2/8 = 0.25 (25%)

DOCUMENTATION:

PHASE1:

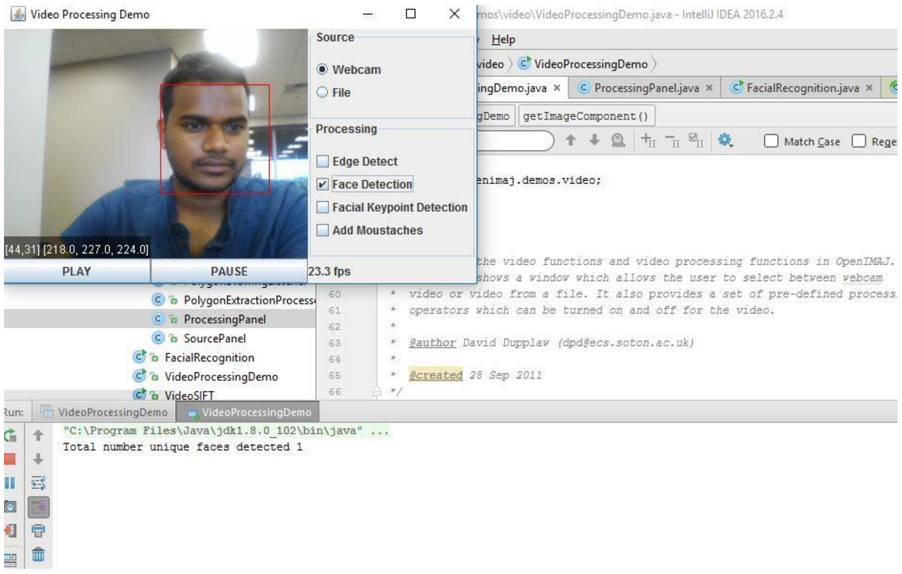
RESULTS USING OPENIMAJ:

1. Static Classroom Video input - 3 faces detecting showing number of students count:



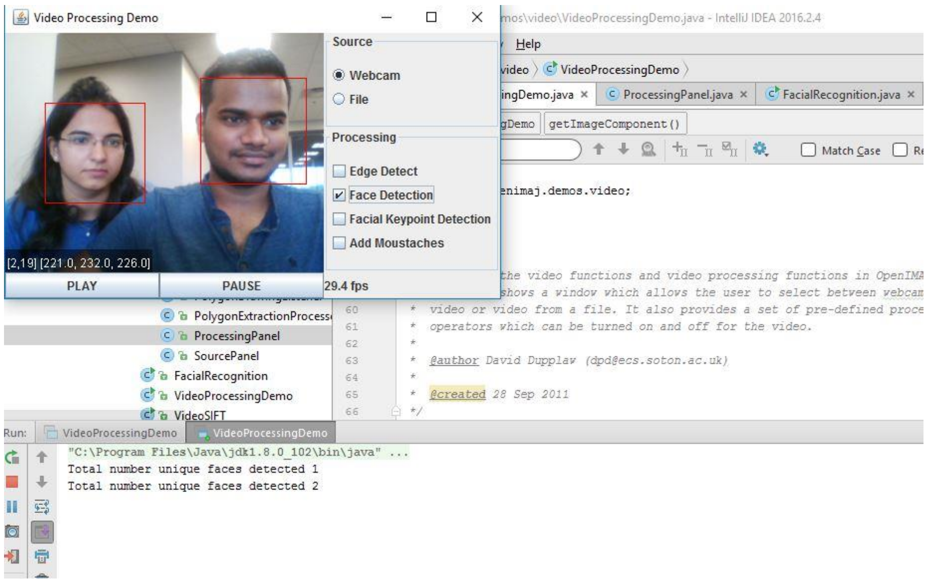
```
19:17:19.265 [Thread-4] ERROR org.ffmpeg - [mov,mp4,m4a,3gp,3g2,mj2 @ 000000001E586FD0] stream 0, offset 0x152af: partial file
19:17:19.265 [Thread-4] ERROR org.ffmpeg - [mov,mp4,m4a,3gp,3g2,mj2 @ 000000001E586FD0] stream 0, offset 0x15a72: partial file
19:17:19.265 [Thread-4] ERROR org.ffmpeg - [mov,mp4,m4a,3gp,3g2,mj2 @ 000000001E586FD0] stream 0, offset 0x169b5: partial file
19:17:19.266 [Thread-4] ERROR org.ffmpeg - [mov,mp4,m4a,3gp,3g2,mj2 @ 000000001E586FD0] stream 0, offset 0x17149: partial file
19:17:19.266 [Thread-4] ERROR org.ffmpeg - [mov,mp4,m4a,3gp,3g2,mj2 @ 000000001E586FD0] stream 0, offset 0x1774f: partial file
19:17:19.266 [Thread-4] ERROR org.ffmpeg - [mov,mp4,m4a,3gp,3g2,mj2 @ 000000001E586FD0] stream 0, offset 0x17d55: partial file
19:17:19.266 [Thread-4] ERROR org.ffmpeg - [mov,mp4,m4a,3gp,3g2,mj2 @ 000000001E586FD0] stream 0, offset 0x18377: partial file
19:17:19.297 [Thread-7] ERROR org.ffmpeg - [h264 @ 000000001E5C2180] Missing reference picture
19:17:19.297 [Thread-7] ERROR org.ffmpeg - [h264 @ 000000001E5C2180] decode_slice_header error
Does timekeeper support pause? true
Total number unique faces detected in a frame 1
Total number unique faces detected in a frame 2
Total number unique faces detected in a frame 3
```

2. Live input using WebCam showing One face detected:



```
"C:\Program Files\Java\jdk1.8.0_102\bin\java" ...
Total number unique faces detected 1
```

3. Live WebCam input showing Two faces detected:


Video Processing Demo
Source
Webcam
File
Processing
Edge Detect
Face Detection
Facial Keypoint Detection
Add Moustaches
PLAY
PAUSE
29.4 fps
[2,19] [221.0, 232.0, 226.0]
mos\video\VideoProcessingDemo.java - IntelliJ IDEA 2016.2.4
Help
video 〉 VideoProcessingDemo 〉
ingDemo.java × ProcessingPanel.java × FacialRecognition.java ×
gDemo getImageComponent()
Match Case
PolygonExtractionProcess
ProcessingPanel
SourcePanel
FacialRecognition
VideoProcessingDemo
VideoSIFT
Run: VideoProcessingDemo VideoProcessingDemo

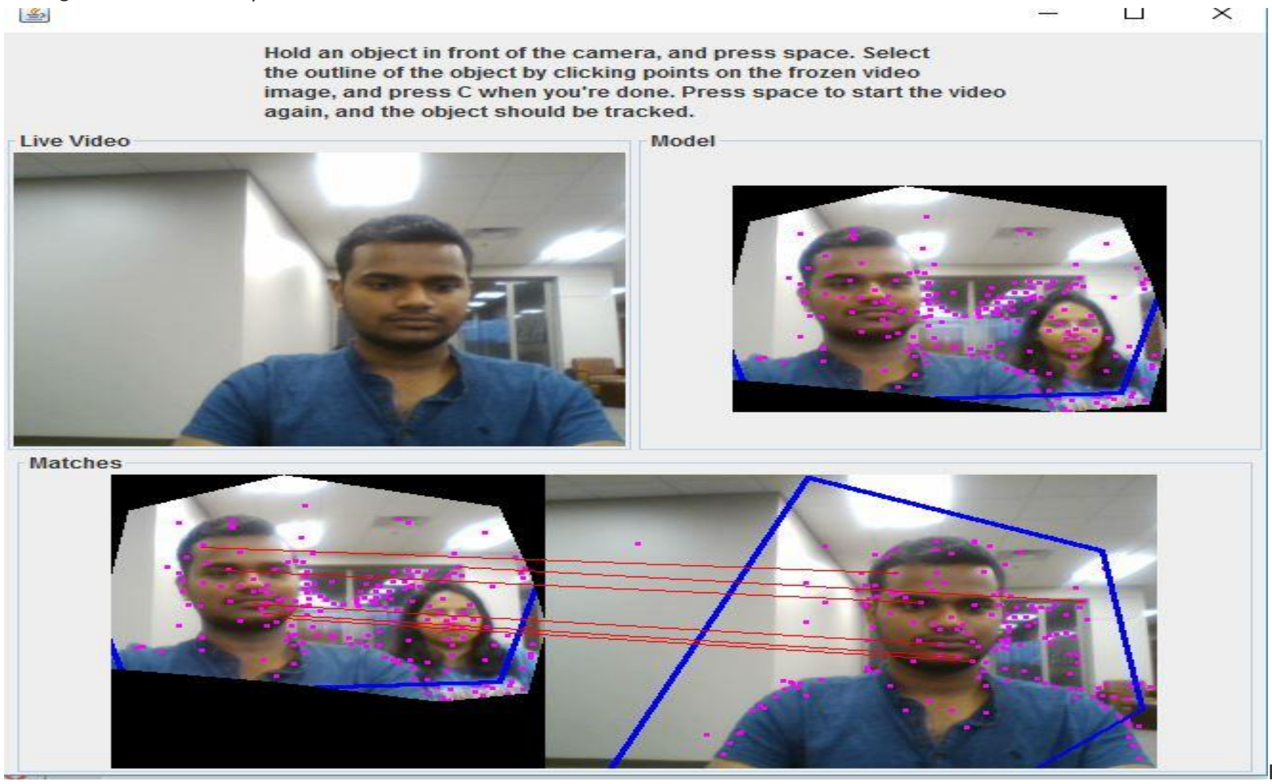
Source: [✓ Webcam] File

Processing: Edge Detect, [✓ Face Detection], Facial Keypoint Detection, Add Moustaches

```
enimaj.demos.video;

 the video functions and video processing functions in OpenIMA
 shows a window which allows the user to select between webcam
 * video or video from a file. It also provides a set of pre-defined proce
 * operators which can be turned on and off for the video.
 *
 * @author David Dupplaw (dpd@ecs.soton.ac.uk)
 *
 * @created 28 Sep 2011
 */
```

```
"C:\Program Files\Java\jdk1.8.0_102\bin\java" ...
Total number unique faces detected 1
Total number unique faces detected 2
```

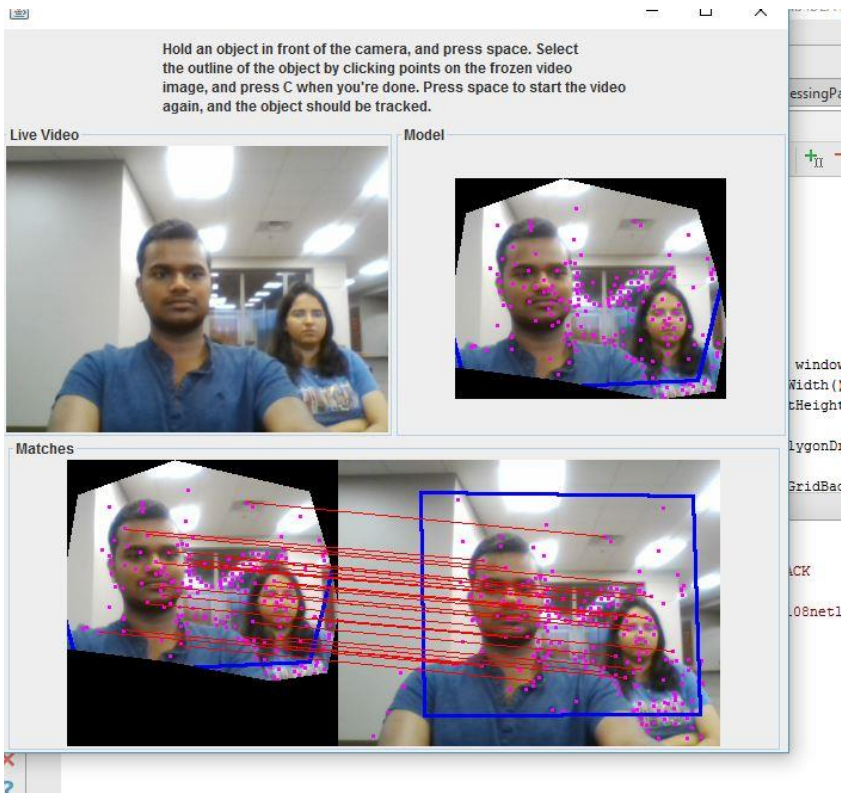
4. Live video input showing Face Recognition (Recognizing students facial features and matching them using SIFT). One match when one student p

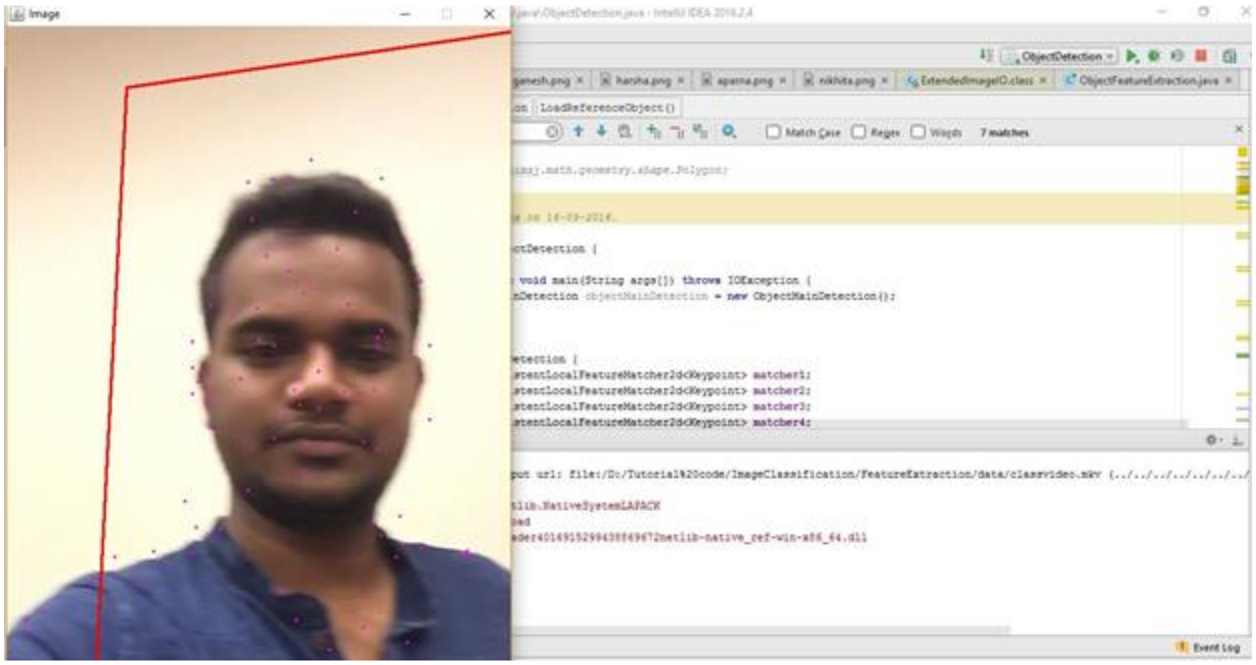


Hold an object in front of the camera, and press space. Select the outline of the object by clicking points on the frozen video image, and press C when you're done. Press space to start the video again, and the object should be tracked.

resent in class.

4. Live video input showing Face Recognition (Recognizing students facial features and matching them using SIFT). Two matches when two student present in class.


Hold an object in front of the camera, and press space. Select the outline of the object by clicking points on the frozen video image, and press C when you're done. Press space to start the video again, and the object should be tracked.

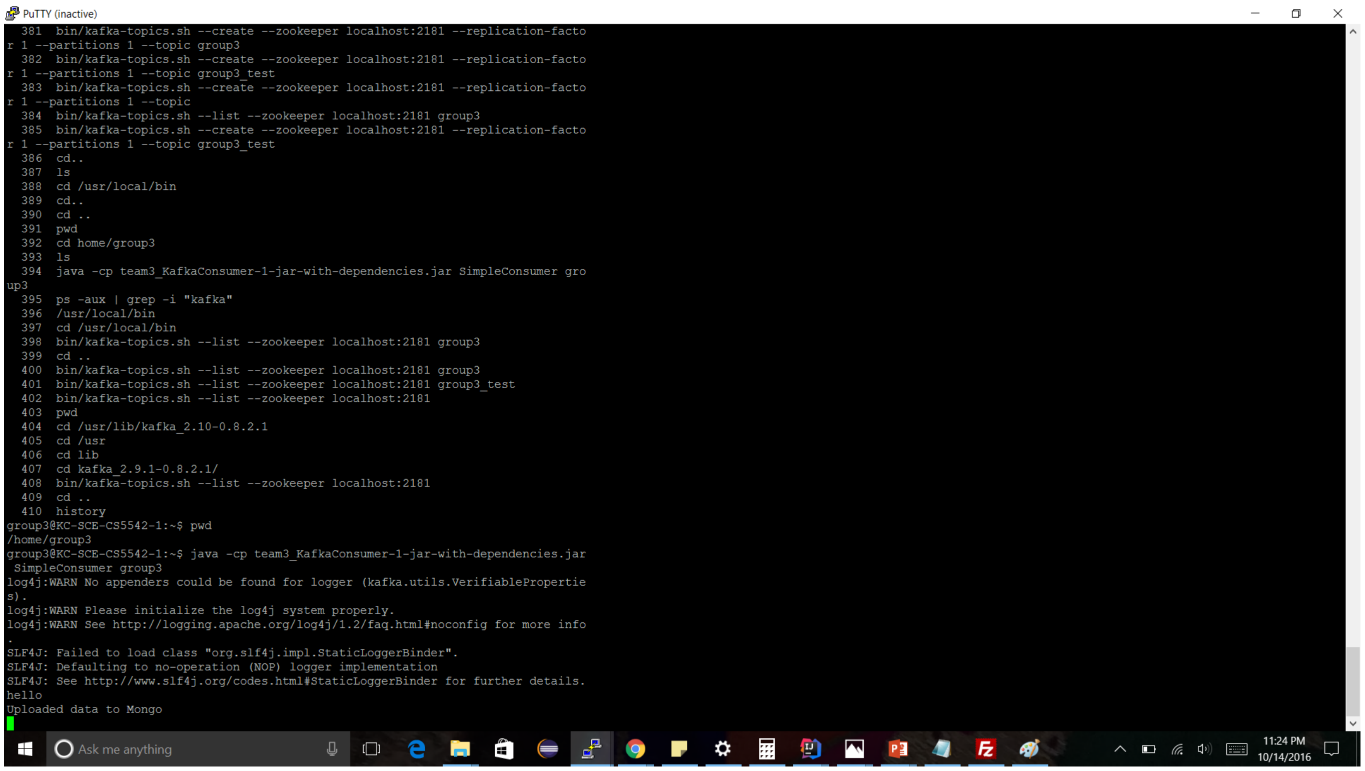


```
imaj.math.geometry.shape.Polygon;

e on 14-03-2016.

ctDetection {

 void main(String args[]) throws IOException {
nDetection objectMainDetection = new ObjectMainDetection();


etection {
stentLocalFeatureMatcher2d<Keypoint> matcher1;
stentLocalFeatureMatcher2d<Keypoint> matcher2;
stentLocalFeatureMatcher2d<Keypoint> matcher3;
stentLocalFeatureMatcher2d<Keypoint> matcher4;
```

```
put url: file:/D:/Tutorial%20code/ImageClassification/FeatureExtraction/data/classvideo.mkv (../../../../../../

tlib.NativeSystemLAPACK
oad
ader4016915299438849672netlib-native_ref-win-x86_64.dll
```

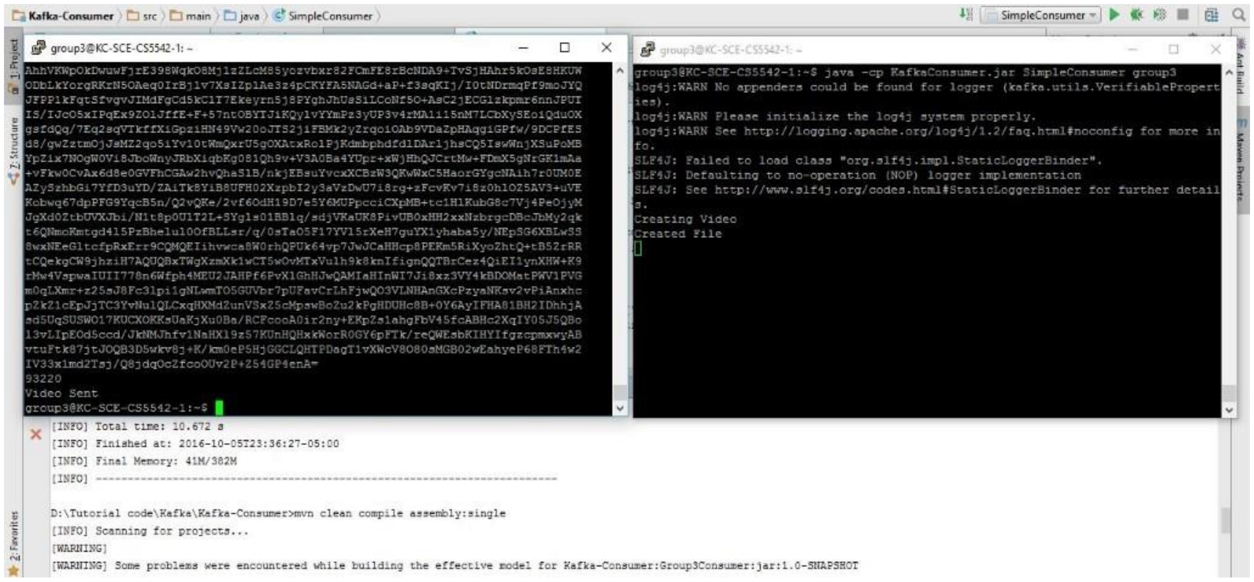
Fig: Classification of individual student in a video data from the model generated using the image key features

Phase2 :



```
  381  bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-facto
r 1 --partitions 1 --topic group3
  382  bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-facto
r 1 --partitions 1 --topic group3_test
  383  bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-facto
r 1 --partitions 1 --topic
  384  bin/kafka-topics.sh --list --zookeeper localhost:2181 group3
  385  bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-facto
r 1 --partitions 1 --topic group3_test
  386  cd..
  387  ls
  388  cd /usr/local/bin
  389  cd..
  390  cd ..
  391  pwd
  392  cd home/group3
  393  ls
  394  java -cp team3_KafkaConsumer-1-jar-with-dependencies.jar SimpleConsumer gro
up3
  395  ps -aux | grep -i "kafka"
  396  /usr/local/bin
  397  cd /usr/local/bin
  398  bin/kafka-topics.sh --list --zookeeper localhost:2181 group3
  399  cd ..
  400  bin/kafka-topics.sh --list --zookeeper localhost:2181 group3
  401  bin/kafka-topics.sh --list --zookeeper localhost:2181 group3_test
  402  bin/kafka-topics.sh --list --zookeeper localhost:2181
  403  pwd
  404  cd /usr/lib/kafka_2.10-0.8.2.1
  405  cd /usr
  406  cd lib
  407  cd kafka_2.9.1-0.8.2.1/
  408  bin/kafka-topics.sh --list --zookeeper localhost:2181
  409  cd ..
  410  history
group3@KC-SCE-CS5542-1:~$ pwd
/home/group3
group3@KC-SCE-CS5542-1:~$ java -cp team3_KafkaConsumer-1-jar-with-dependencies.jar
 SimpleConsumer group3
log4j:WARN No appenders could be found for logger (kafka.utils.VerifiablePropertie
s).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info
.
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
hello
Uploaded data to Mongo
```

Fig: The data sent to the Kafka using RestApi



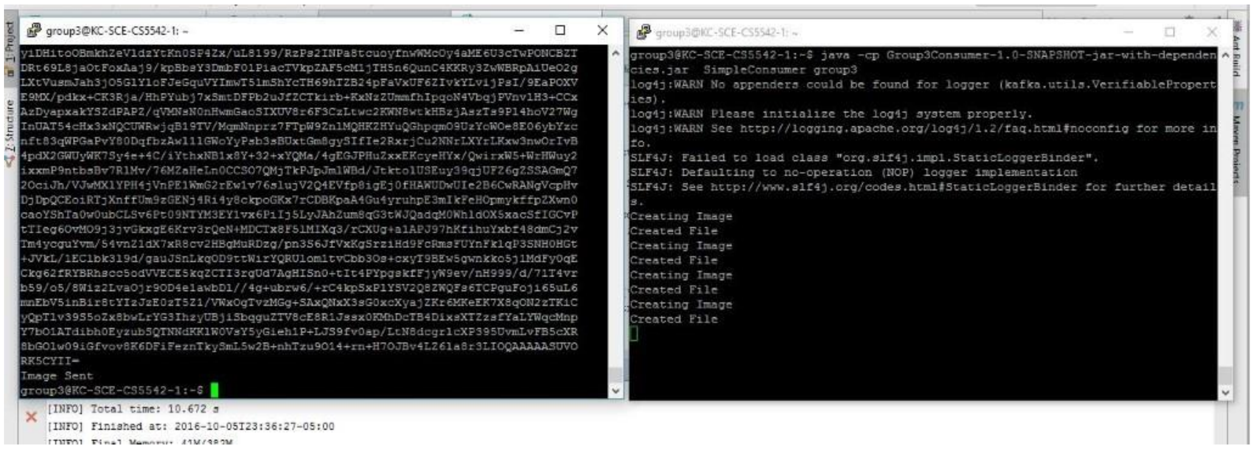
```
AhhVKWpOkDwuwFjrE398WqkO8Mj1zZLcM85yozvbxr82FCmFE8rBcNDA9+TvSjHAhr5kOsE8HKUW
ODbLkYorgRKrN5OAeq0IrBjlv7XsIZplAe3z4pCKYFA5NAGd+aP+f3sqKIj/I0tNDrmqPf9moJYQ
JFPPlkFqtSfvgvJIMdFgCd5kC1I7Ekeyrn5j8PYghJhUsSiLCoNf5O+AsC2jECGlzkpmr6nnJPUI
IS/IJcO5xIPqEx9ZOlJffE+F+57ntOBYTJiKQylvYYmPz3yUP3v4rMA1i15nM7LCbXySEoiQduOX
gsfdQg/7Eq2sqVTkffXiGpziHN49Vw20oJTS2jiFBMk2yZrqoiOAb9VDaZpHAqgiGPfw/9DCPfES
d8/gwZztmOjJsMZ2qo5iYv10tWmQxrU5gOXAtxRolPjKdmbphdfdlDArljhsCQ5IswWnjXSuPoMB
YpZix7NOgW0Vi8JboWnyJRbXiqbKg081Qh9v+V3A0Ba4YUpr+xWjHhQJCrtMw+FDmX5gNrGK1mAa
+vFkw0CvAx6d8e0GVFhCGAw2hvQhaS1B/nkjEBsuYvcxXCBzW3QKwWxC5HaorGYgcNAih7r0UM0E
AZySzhbGi7YfD3uYD/ZAiTk8YiB8UFH02XzpbI2y3aVzDwU7i8rg+zFcvKv7i8z0hlOZ5AV3+uVE
Kobwq67dpPFG9YqcB5n/Q2vQKe/2vf6OdH19D7e5Y6MUPpcciCXpMB+tc1HlKubG8c7Vj4PeOjyM
JgXd0ZtbUVXJbi/N1t8p0UlT2L+5Ygls01BBlq/sdjVKaUK8PivUB0xHH2xxNzbrgcDBcJbMy2qk
t6QNmoKmtgd415PzBhelul0OfBLLsr/q/0sTaO5F17YV15rXeH7guYX1yhaba5y/NEpSG6XBLwSS
8wxNEeG1tcfpRxErr9CQMQEIihvwca8W0rhQPUk64vp7JwJCaHHcp8PEKm5RiXyoZhtQ+tB5ZrRR
tCQekgCW9jhziH7AQUQBxTWgXzmXk1wCT5wOvMTxVulh9k8knIfignQQTBrCez4QiEI1ynXHW+K9
rMw4VspwaIUII778n6Wfph4MEU2JAHPf6PvX1GhHJwQAMIaHInWI7Ji8xz3VY4kBDOMatPWV1PVG
m0qLXmr+z25sJ8Fc3lpi1gNLwmTO5GUVbr7pUFavCrLhFjwQO3VLNHAnGXcPzyaNKsv2vPiAnxhc
pZkZ1cEpJjTC3YvNulQLCxqHXMdZunVSx25cMpswBoZu2kPgHDUHc8B+0Y6AyIFHA81BH2IDhhjA
sd5UqSUSWO17KUCXOKKsUaKjXu0Ba/RCFcooA0ir2ny+EKpZs1ahgFbV45fcABHc2XqIY05J5QBo
13vLIpEOd5ccd/JkNMJhfv1NaHX19z57KUnHQHxkWorR0GY6pFTk/reQWEsbKIHYIfgzcpmxwyAB
vtuFtk87jtJOQB3D5wkv8j+K/km0eP5HjGGCLQHTPDagT1vXWcV8O80sMGB02wEahyeP68FTh4w2
IV33x1md2Tsj/Q8jdqOcZfcoOUv2P+Z54GP4enA=
```

```
93220
Video Sent
group3@KC-SCE-CS5542-1:~$
```

```
group3@KC-SCE-CS5542-1:~$ java -cp KafkaConsumer.jar SimpleConsumer group3
log4j:WARN No appenders could be found for logger (kafka.utils.VerifiablePropert
ies).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more in
fo.
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further detail
s.
Creating Video
Created File
```

```
[INFO] Total time: 10.672 s
[INFO] Finished at: 2016-10-05T23:36:27-05:00
[INFO] Final Memory: 41M/382M
[INFO] ------------------------------------------------------------------------

D:\Tutorial code\Kafka\Kafka-Consumer>mvn clean compile assembly:single
[INFO] Scanning for projects...
[WARNING]
[WARNING] Some problems were encountered while building the effective model for Kafka-Consumer:Group3Consumer:jar:1.0-SNAPSHOT
```

Fig: Video sent to the Storm using Kafka



```
yiDHitoOBmkhZeVldzYtKn0SP4Zx/uL8199/RzPs2INPa8tcuoyfnwWMcOy4aME6U3cTwPONCBZT
DRt69L8jaOtFoxAaj9/kpBbsY3DmbF01PiacTVkpZAF5cM1jTH5n6QunC4KKRy3ZwWBRpAiUeO2g
LXtVusmJah3jO5G1YloFJeGquVYImwT51mShYcTH69hTZB24pFaVxUF6ZIvkYLvijPsI/9EaPOXV
E9MX/pdkx+CK3Rja/HhPYubj7xSmtDFPb2uJfZCTkirb+KxNzZUmmfhIpqoN4VbqjFVnvlH3+CCx
AzDyapxakYSZdPAPZ/qVMNsN0nHwmGaoSIXUV8r6F3CzLtwc2KWN8wtkHBzjAszTs9Pl4hoV27Wg
InUAT54cHx3xNQCUWRwjqB19TV/MgmNnprz7FTpW9ZnlMQHKZHYuQGhpqm09UzYoWOe8E06ybYzc
nft83qWPGaPvY80DqfbzAw111GWoYyPsb3sBUxtGm8gySIfIe2RxrjCu2NNrLXYrLKxw3nwOrIvB
4pdX2GWUyWK7Sy4e+4C/iYthxNB1x8Y+32+xYQMa/4gEGJPHuZxxEKcyeHYx/QwirxW5+WrHWuy2
ixxmP9ntbsBv7RlMv/76MZaHeLn0CCSO7QMjTkPJpJmlWBd/JtktolUSEuy39qjUFZ6gZSSAGmQ7
2OciJh/VJwMXlYPH4jVnPE1WmG2rEw1v76slujV2Q4EVfp8igEj0fHAWUDwUIe2B6CwRANgVcpHv
DjDpQCEoiRTjXnffUm9zGENj4Ri4y8ckpoGKx7rCDBKpaA4Gu4yruhpE3mIkFeHOpmykffpZXwn0
caoYShTa0w0ubCLSv6Pt09NTYM3EY1vx6PiIj5LyJAhZum8qG3tWJQadqM0Whld0X5xacSfIGCvP
tTIeg6OvMO9j3jvGkxgE6Krv3rQeN+MDCTx8F5lMIXq3/rCXUg+alAPJ97hKfihuYxbf48dmCj2v
Tm4ycguYvm/54vnZ1dX7xR8cv2HBgMuRDzg/pn356JfVxKg5rziHd9FcRmsFUYnFklqP35NH0HGt
+JVkL/1EClbk319d/gauJSnLkqOD9ttWirYQRUlomltvCbb3Os+cxyT9BEw5gwnkko5jlMdFy0qE
Ckg62fRYBRhscc5odVVECE5kqZCTI3rgUd7AgHISn0+tIt4PYpgskfFjyW9ev/nH999/d/71T4vr
b59/o5/8Wiz2LvaOjr9OD4elawbDl//4g+ubrw6/+rC4kp5xPlYSV2Q8ZWQFs6TCPguFoji65uL6
mnEbV5inBir8tYIzJzE0zT5Z1/VWxOgTvzMGg+SAxQNxX3sG0xcXyajZKr6MKeEK7X8qON2zTKiC
yQpTlv39S5oZx8bwLrYG3IhzyUBjiSbqguZTV8cE8R1Jssx0KMhDcTB4DixsXTZzsfYaLYWqcMnp
Y7bO1ATdibh0EyzubSQTNNdKK1W0VsY5yGieh1P+LJS9fv0ap/LtN8dcgrlcXP395UvmLvFB5cXR
8bGOlw09iGfvov8K6DFiFeznTkySmL5w2B+nhTzu9O14+rn+H7OJBv4LZ61a8r3LIOQAAAAASUVO
RK5CYII=
```

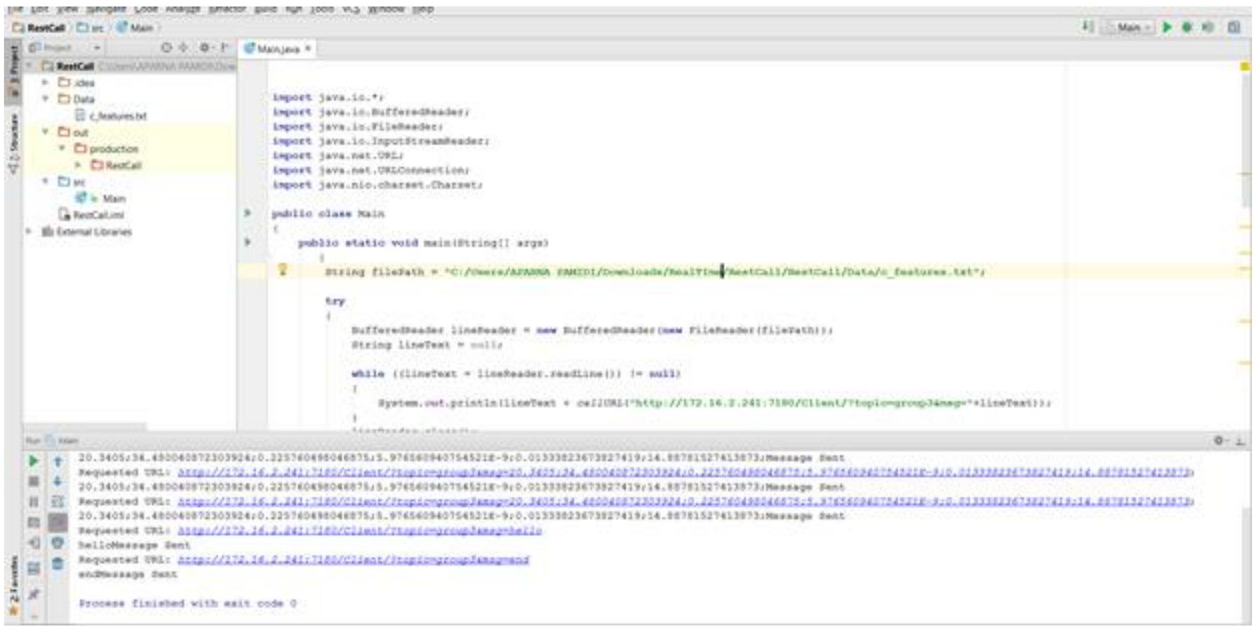
```
Image Sent
group3@KC-SCE-CS5542-1:~$
```

```
[INFO] Total time: 10.672 s
[INFO] Finished at: 2016-10-05T23:36:27-05:00
[INFO] Final Memory: 41M/382M
```

```
group3@KC-SCE-CS5542-1:~$ java -cp Group3Consumer-1.0-SNAPSHOT-jar-with-dependen
cies.jar  SimpleConsumer group3
log4j:WARN No appenders could be found for logger (kafka.utils.VerifiablePropert
ies).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more in
fo.
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further detail
s.
Creating Image
Created File
Creating Image
Created File
Creating Image
Created File
Creating Image
Created File
```

Fig: Image sent to the Storm using Kafka

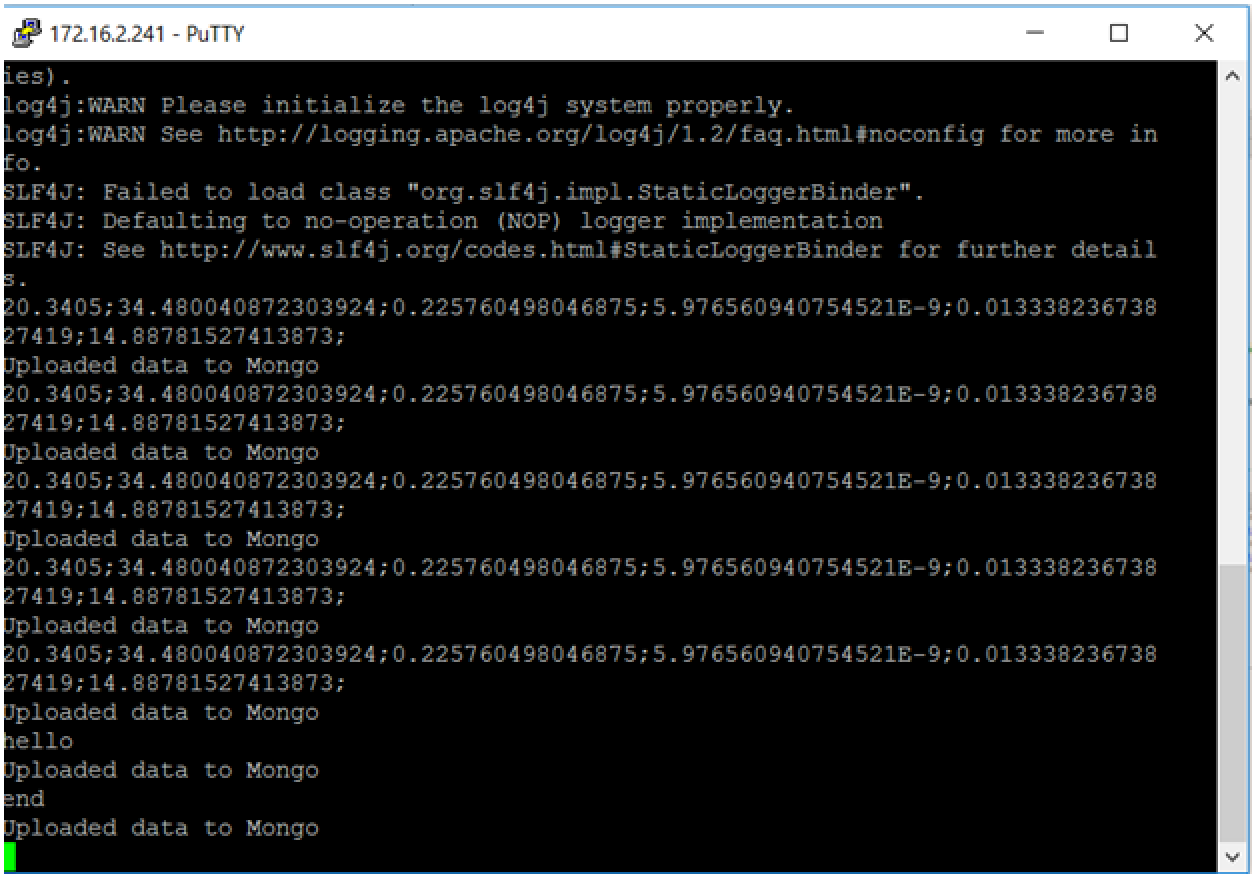
Phase 3:



```
import java.io.*;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.InputStreamReader;
import java.net.URL;
import java.net.URLConnection;
import java.nio.charset.Charset;

public class Main
{
    public static void main(String[] args)
    {
        String filePath = "C:/Users/APARNA PAMIDI/Downloads/RealTime/RestCall/RestCall/Data/c_features.txt";

        try
        {
            BufferedReader lineReader = new BufferedReader(new FileReader(filePath));
            String lineText = null;

            while ((lineText = lineReader.readLine()) != null)
            {
                System.out.println(lineText + callURL("http://172.16.2.241:7180/Client/?topic=group3&msg="+lineText));
            }
```

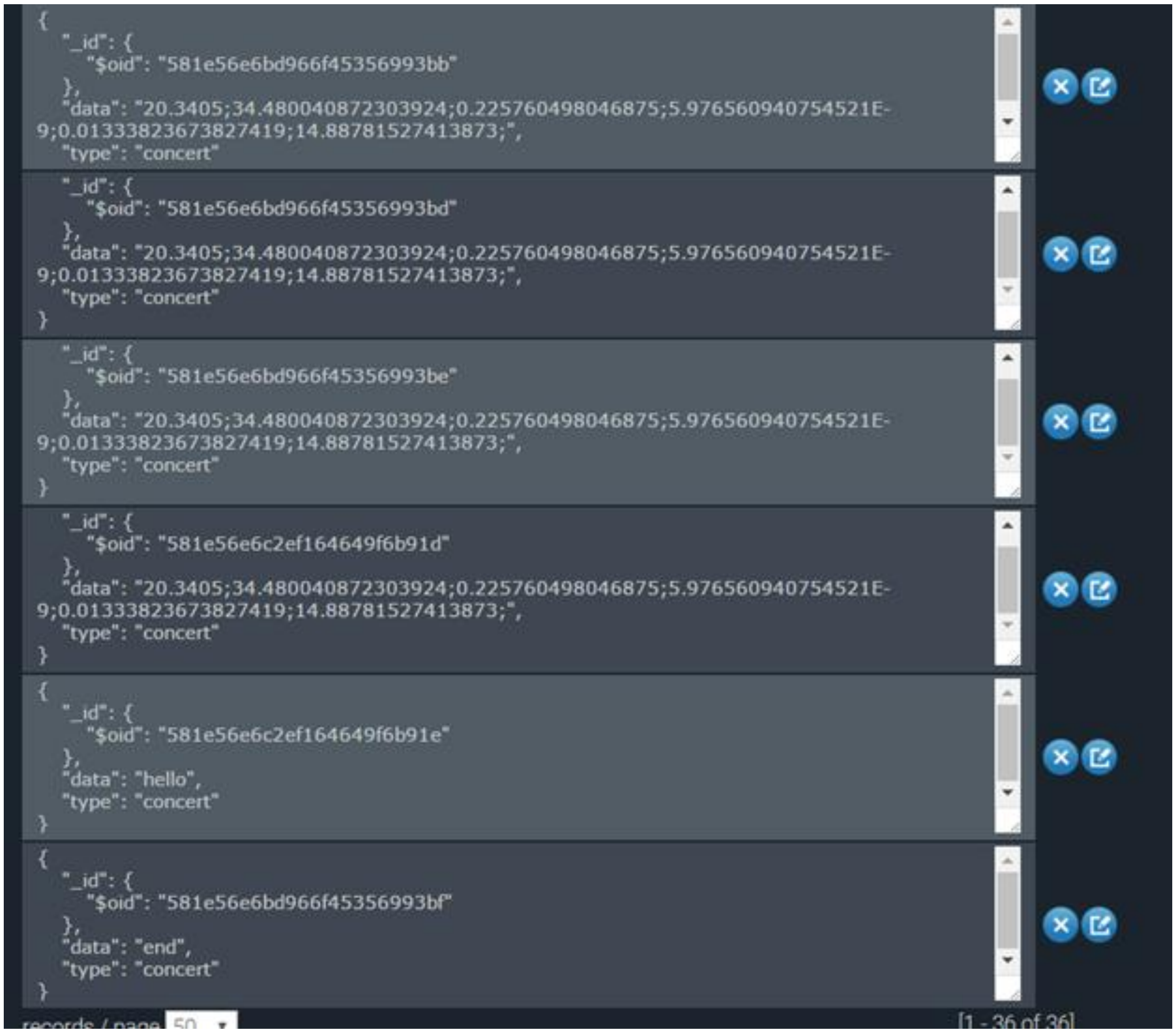
```
20.3405;34.480040872303924;0.225760498046875;5.976560940754521E-9;0.01333823673827419;14.88781527413873;Message Sent
Requested URL: http://172.16.2.241:7180/Client/?topic=group3&msg=20.3405;34.480040872303924;0.225760498046875;5.976560940754521E-9;0.01333823673827419;14.88781527413873;
20.3405;34.480040872303924;0.225760498046875;5.976560940754521E-9;0.01333823673827419;14.88781527413873;Message Sent
Requested URL: http://172.16.2.241:7180/Client/?topic=group3&msg=20.3405;34.480040872303924;0.225760498046875;5.976560940754521E-9;0.01333823673827419;14.88781527413873;
20.3405;34.480040872303924;0.225760498046875;5.976560940754521E-9;0.01333823673827419;14.88781527413873;Message Sent
Requested URL: http://172.16.2.241:7180/Client/?topic=group3&msg=hello
helloMessage Sent
Requested URL: http://172.16.2.241:7180/Client/?topic=group3&msg=end
endMessage Sent

Process finished with exit code 0
```

Fig: The REST API call to send the features file to MongoDB



```
ies).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more in
fo.
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further detail
s.
20.3405;34.480040872303924;0.225760498046875;5.976560940754521E-9;0.013338236738
27419;14.88781527413873;
Uploaded data to Mongo
20.3405;34.480040872303924;0.225760498046875;5.976560940754521E-9;0.013338236738
27419;14.88781527413873;
Uploaded data to Mongo
20.3405;34.480040872303924;0.225760498046875;5.976560940754521E-9;0.013338236738
27419;14.88781527413873;
Uploaded data to Mongo
20.3405;34.480040872303924;0.225760498046875;5.976560940754521E-9;0.013338236738
27419;14.88781527413873;
Uploaded data to Mongo
20.3405;34.480040872303924;0.225760498046875;5.976560940754521E-9;0.013338236738
27419;14.88781527413873;
Uploaded data to Mongo
hello
Uploaded data to Mongo
end
Uploaded data to Mongo
```

Fig: The figure depicts that the data has been successfully uploaded to the MongoDB



```
{
   "_id": {
      "$oid": "581e56e6bd966f45356993bb"
   },
   "data": "20.3405;34.480040872303924;0.225760498046875;5.976560940754521E-
9;0.01333823673827419;14.88781527413873;",
   "type": "concert"
```

```
   "_id": {
      "$oid": "581e56e6bd966f45356993bd"
   },
   "data": "20.3405;34.480040872303924;0.225760498046875;5.976560940754521E-
9;0.01333823673827419;14.88781527413873;",
   "type": "concert"
}
```

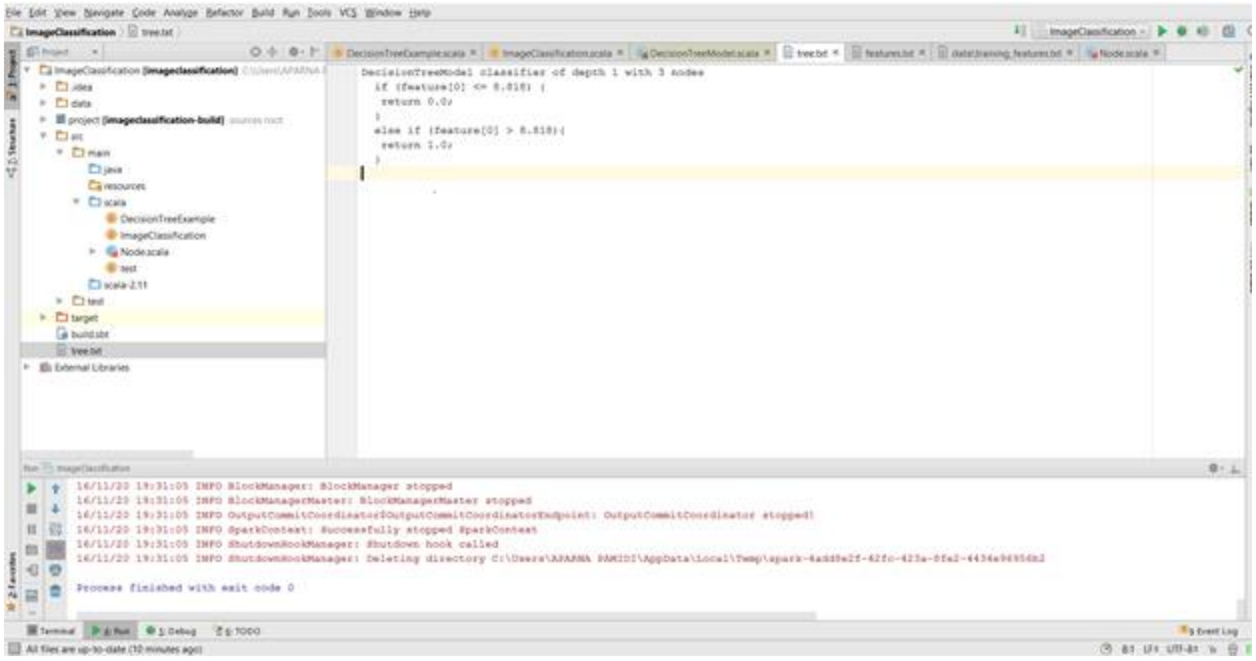
```
   "_id": {
      "$oid": "581e56e6bd966f45356993be"
   },
   "data": "20.3405;34.480040872303924;0.225760498046875;5.976560940754521E-
9;0.01333823673827419;14.88781527413873;",
   "type": "concert"
}
```

```
   "_id": {
      "$oid": "581e56e6c2ef164649f6b91d"
   },
   "data": "20.3405;34.480040872303924;0.225760498046875;5.976560940754521E-
9;0.01333823673827419;14.88781527413873;",
   "type": "concert"
}
```

```
{
   "_id": {
      "$oid": "581e56e6c2ef164649f6b91e"
   },
   "data": "hello",
   "type": "concert"
}
```

```
{
   "_id": {
      "$oid": "581e56e6bd966f45356993bf"
   },
   "data": "end",
   "type": "concert"
}
```

Fig: Screenshot of data being stored in tuples format in mongoDB


```
DecisionTreeModel classifier of depth 1 with 3 nodes
  If (feature[0] <= 8.818) {
   return 0.0;
  }
  else if (feature[0] > 8.818){
   return 1.0;
  }
```

```
16/11/20 19:31:05 INFO BlockManager: BlockManager stopped
16/11/20 19:31:05 INFO BlockManagerMaster: BlockManagerMaster stopped
16/11/20 19:31:05 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
16/11/20 19:31:05 INFO SparkContext: Successfully stopped SparkContext
16/11/20 19:31:05 INFO ShutdownHookManager: Shutdown hook called
16/11/20 19:31:05 INFO ShutdownHookManager: Deleting directory C:\Users\APARNA PAMIDI\AppData\Local\Temp\spark-4add0e2f-42fc-423a-8fe2-4434e96956b2

Process finished with exit code 0
```

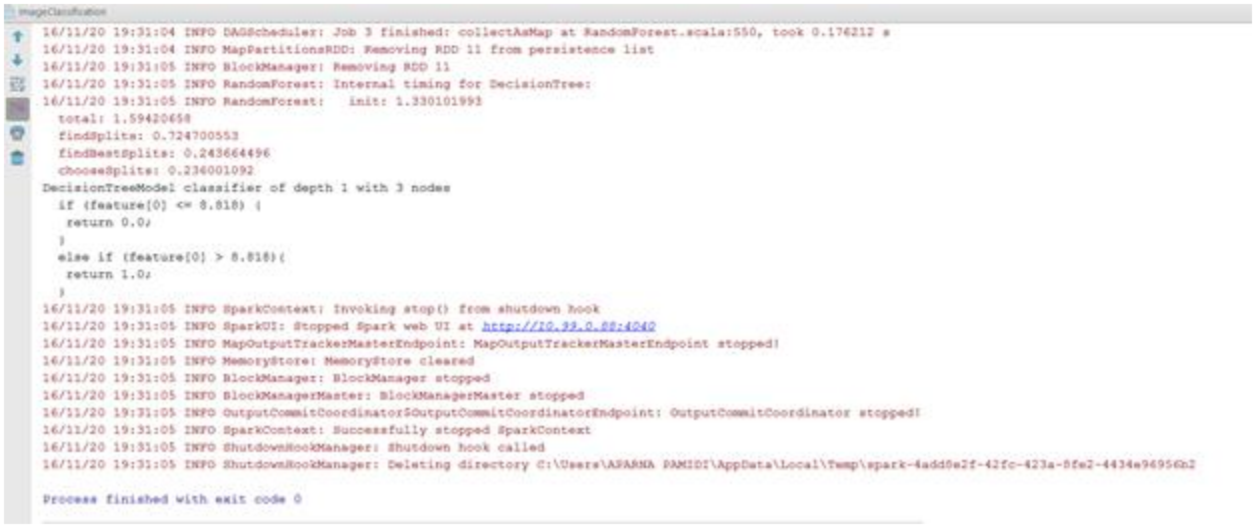
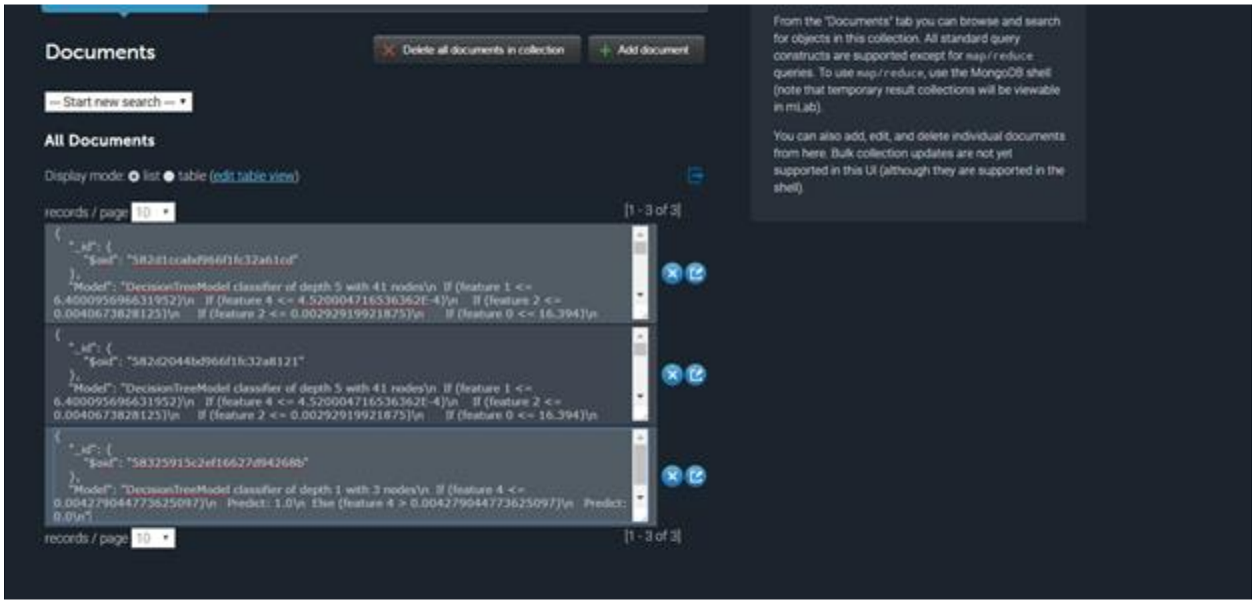

```
16/11/20 19:31:04 INFO DAGScheduler: Job 3 finished: collectAsMap at RandomForest.scala:550, took 0.176212 s
16/11/20 19:31:04 INFO MapPartitionsRDD: Removing RDD 11 from persistence list
16/11/20 19:31:05 INFO BlockManager: Removing RDD 11
16/11/20 19:31:05 INFO RandomForest: Internal timing for DecisionTree:
16/11/20 19:31:05 INFO RandomForest:   init: 1.330101993
  total: 1.59420658
  findSplits: 0.724700553
  findBestSplits: 0.243664496
  chooseSplits: 0.236001092
DecisionTreeModel classifier of depth 1 with 3 nodes
  If (feature[0] <= 8.818) {
   return 0.0;
  }
  else if (feature[0] > 8.818){
   return 1.0;
  }
16/11/20 19:31:05 INFO SparkContext: Invoking stop() from shutdown hook
16/11/20 19:31:05 INFO SparkUI: Stopped Spark web UI at http://10.99.0.88:4040
16/11/20 19:31:05 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
16/11/20 19:31:05 INFO MemoryStore: MemoryStore cleared
16/11/20 19:31:05 INFO BlockManager: BlockManager stopped
16/11/20 19:31:05 INFO BlockManagerMaster: BlockManagerMaster stopped
16/11/20 19:31:05 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
16/11/20 19:31:05 INFO SparkContext: Successfully stopped SparkContext
16/11/20 19:31:05 INFO ShutdownHookManager: Shutdown hook called
16/11/20 19:31:05 INFO ShutdownHookManager: Deleting directory C:\Users\APARNA PAMIDI\AppData\Local\Temp\spark-4add0e2f-42fc-423a-8fe2-4434e96956b2

Process finished with exit code 0
```

Fig: The figure depicts that generation of decision tree from the input features file



```
{
  "_id": {
    "$oid": "582d1ccabd966f1fc32a61cd"
  },
  "Model": "DecisionTreeModel classifier of depth 5 with 41 nodes\n  If (feature 1 <=
6.400095696631952)\n   If (feature 4 <= 4.520004716536362E-4)\n    If (feature 2 <=
0.00406738281251)\n     If (feature 2 <= 0.002929199218751)\n      If (feature 0 <= 16.394)\n
```

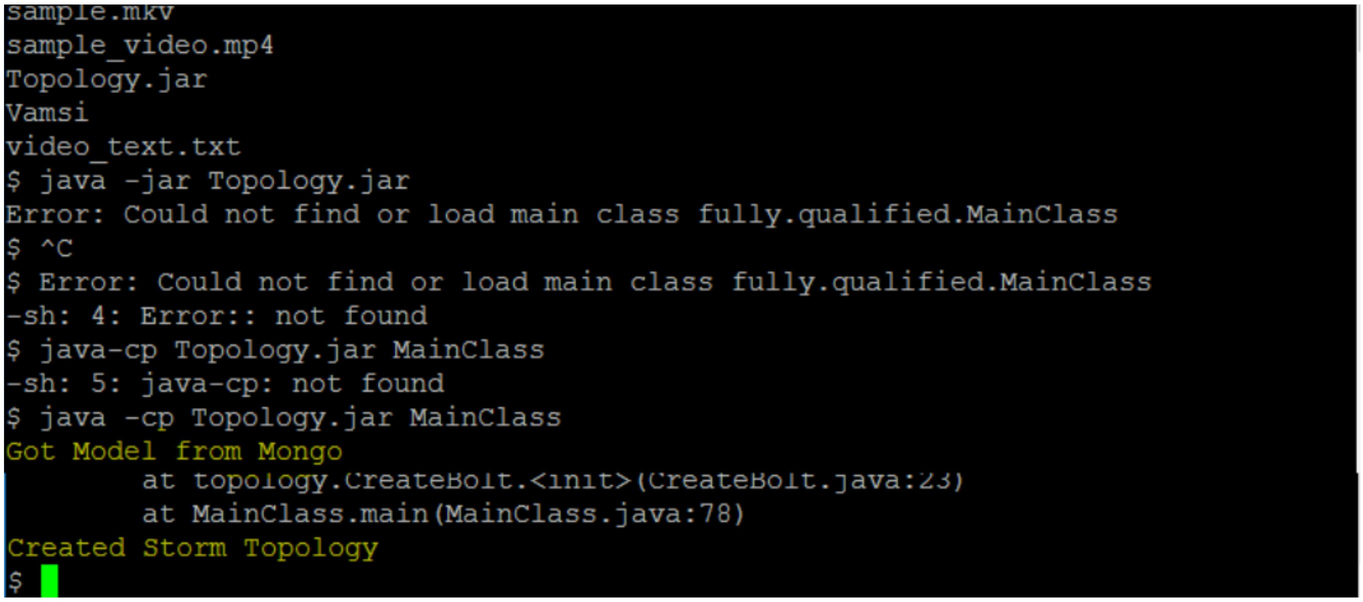
```
{
  "_id": {
    "$oid": "582d2044bd966f1fc32a8121"
  },
  "Model": "DecisionTreeModel classifier of depth 5 with 41 nodes\n  If (feature 1 <=
6.400095696631952)\n   If (feature 4 <= 4.520004716536362E-4)\n    If (feature 2 <=
0.00406738281251)\n     If (feature 2 <= 0.002929199218751)\n      If (feature 0 <= 16.394)\n
```

```
{
  "_id": {
    "$oid": "58325915c2ef16627d94268b"
  },
  "Model": "DecisionTreeModel classifier of depth 1 with 3 nodes\n  If (feature 4 <=
0.004279044773625097)\n   Predict: 1.0\n  Else (feature 4 > 0.004279044773625097)\n   Predict:
0.0\n"
```

From the "Documents" tab you can browse and search for objects in this collection. All standard query constructs are supported except for map/reduce queries. To use map/reduce, use the MongoDB shell (note that temporary result collections will be viewable in mLab).

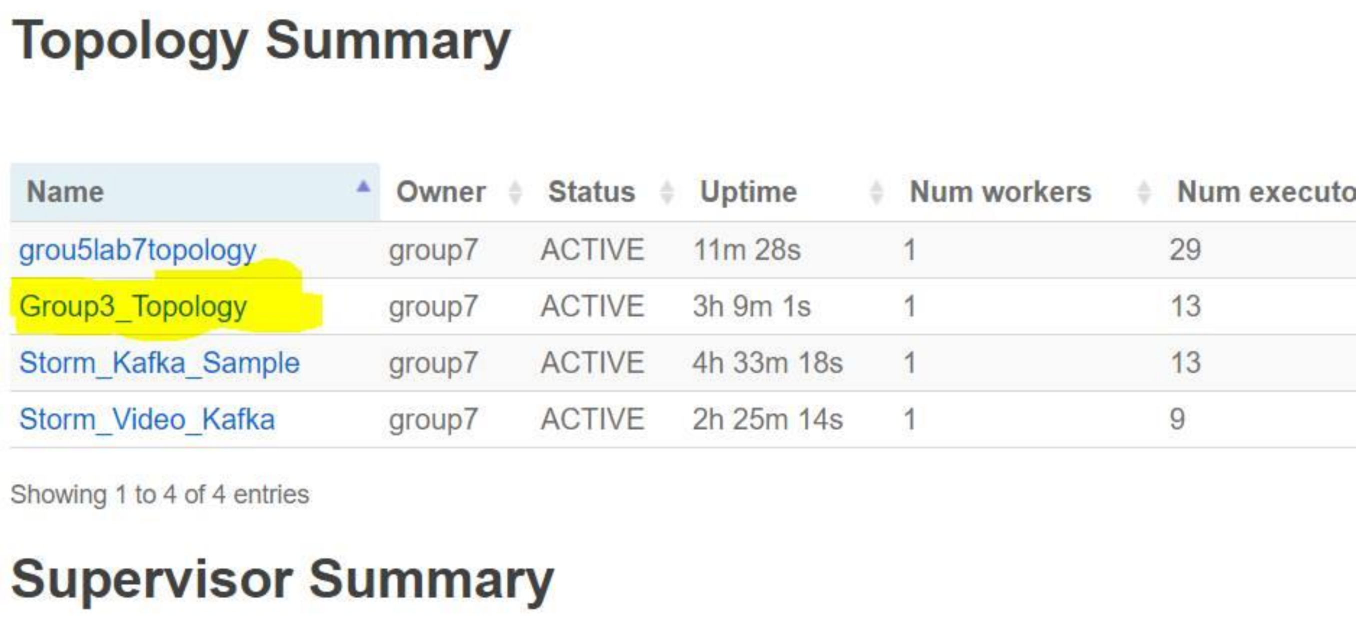
You can also add, edit, and delete individual documents from here. Bulk collection updates are not yet supported in this UI (although they are supported in the shell).

Fig: The figure depicts the decision tree stored in MongoDB



```
sample.mkv
sample_video.mp4
Topology.jar
Vamsi
video_text.txt
$ java -jar Topology.jar
Error: Could not find or load main class fully.qualified.MainClass
$ ^C
$ Error: Could not find or load main class fully.qualified.MainClass
-sh: 4: Error:: not found
$ java-cp Topology.jar MainClass
-sh: 5: java-cp: not found
$ java -cp Topology.jar MainClass
Got Model from Mongo
        at topology.CreateBolt.<init>(CreateBolt.java:23)
        at MainClass.main(MainClass.java:78)
Created Storm Topology
$
```

Fig: Kafka consumer received the model from the MongoDB and generated topology



Topology Summary

Name	Owner	Status	Uptime	Num workers	Num executo
grou5lab7topology	group7	ACTIVE	11m 28s	1	29
Group3_Topology	group7	ACTIVE	3h 9m 1s	1	13
Storm_Kafka_Sample	group7	ACTIVE	4h 33m 18s	1	13
Storm_Video_Kafka	group7	ACTIVE	2h 25m 14s	1	9

Showing 1 to 4 of 4 entries

Supervisor Summary

Fig: Topology Generation

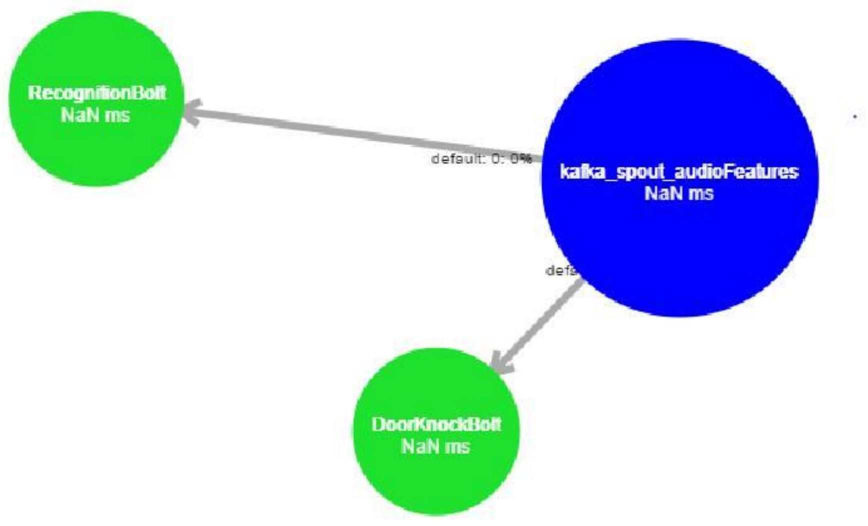
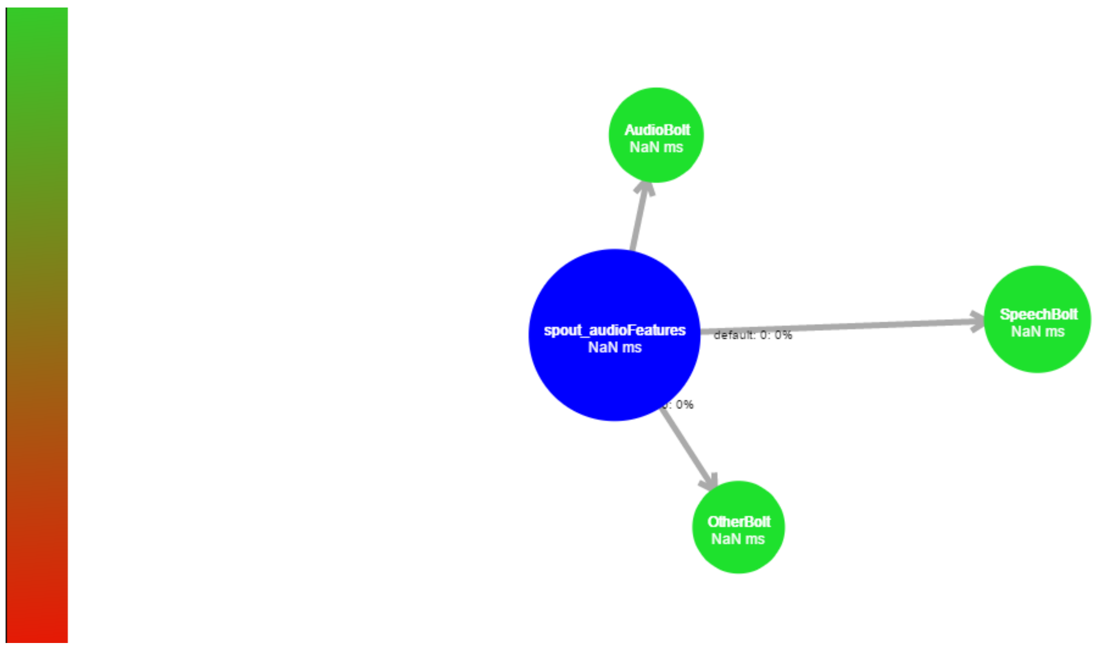
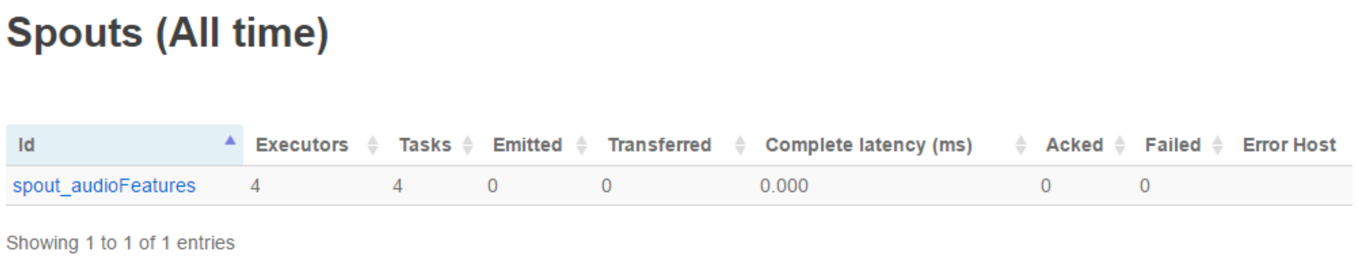


Fig: Figure depicts the creation of spout and bolts generation

PHASE 4:

UPDATED TOPOLOGY:

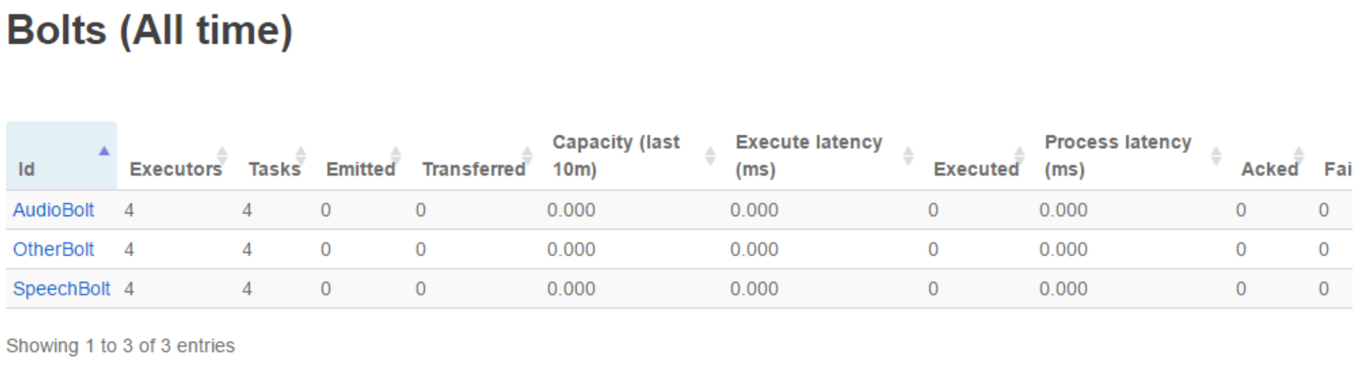




Spouts (All time)

Id	Executors	Tasks	Emitted	Transferred	Complete latency (ms)	Acked	Failed	Error Host
spout_audioFeatures	4	4	0	0	0.000	0	0	

Showing 1 to 1 of 1 entries

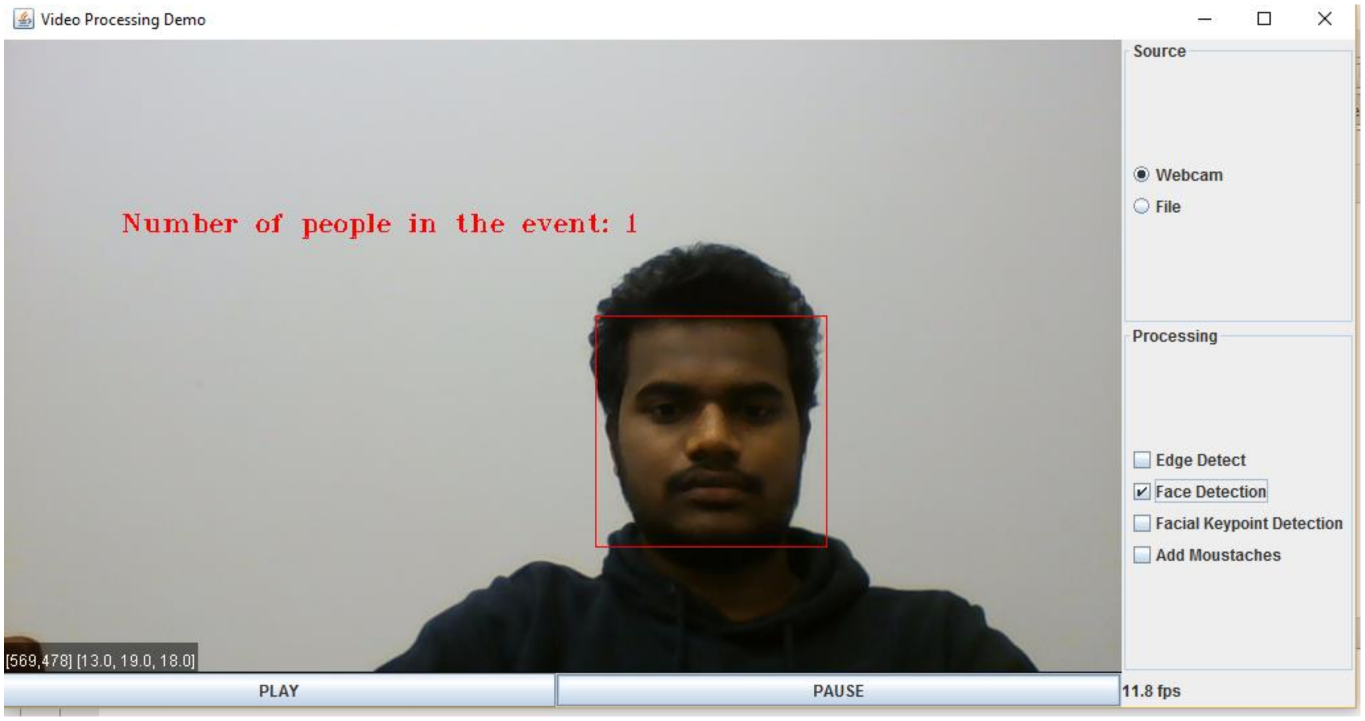


Bolts (All time)

Id	Executors	Tasks	Emitted	Transferred	Capacity (last 10m)	Execute latency (ms)	Executed	Process latency (ms)	Acked	Fai
AudioBolt	4	4	0	0	0.000	0.000	0	0.000	0	0
OtherBolt	4	4	0	0	0.000	0.000	0	0.000	0	0
SpeechBolt	4	4	0	0	0.000	0.000	0	0.000	0	0

Showing 1 to 3 of 3 entries

Video Annotations with One face:

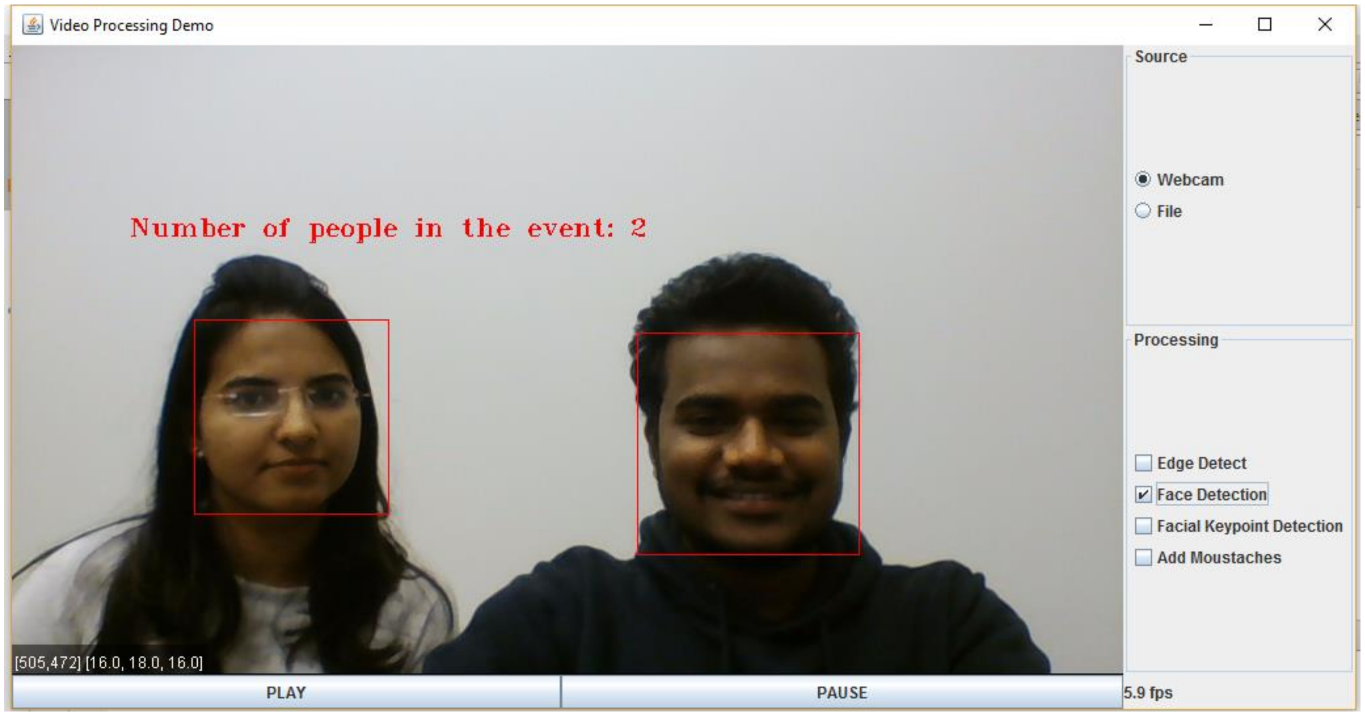


Number of people in the event: 1

[569,478] [13.0, 19.0, 18.0]

11.8 fps

Video Annotations with two faces

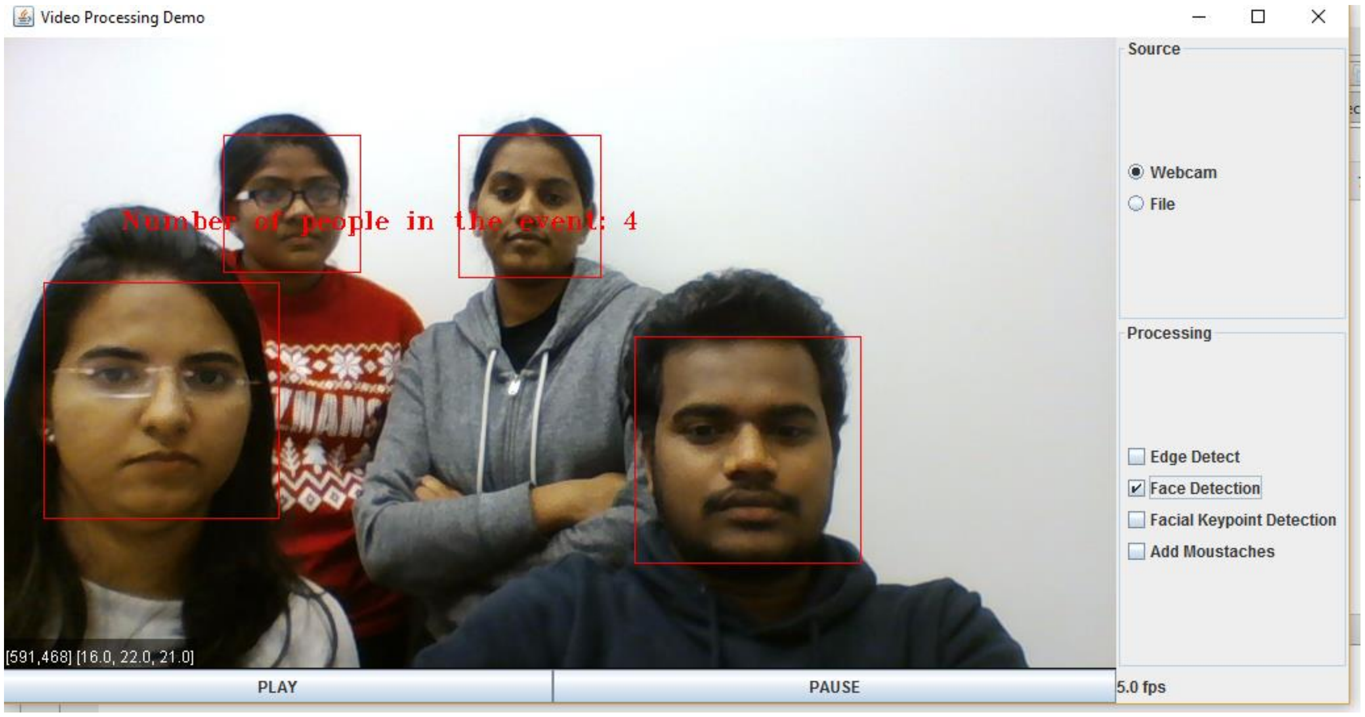


Number of people in the event: 2

[505,472] [16.0, 18.0, 16.0]

5.9 fps

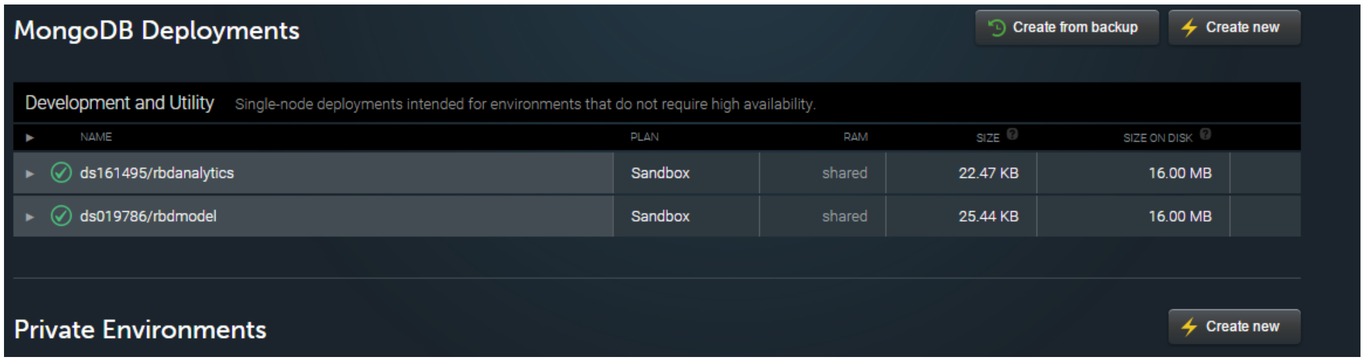
Video Annotations with 4 faces:



Webcam [✓]
File

Edge Detect
Face Detection [✓]
Facial Keypoint Detection
Add Moustaches

MongoDB:



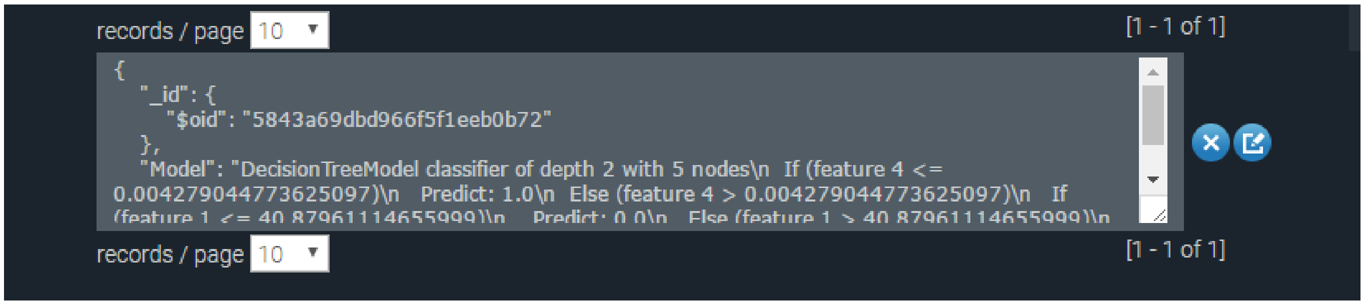
MongoDB Deployments

Development and Utility Single-node deployments intended for environments that do not require high availability.

NAME	PLAN	RAM	SIZE	SIZE ON DISK
ds161495/rbdanalytics	Sandbox	shared	22.47 KB	16.00 MB
ds019786/rbdmodel	Sandbox	shared	25.44 KB	16.00 MB

Private Environments

Model Generated:

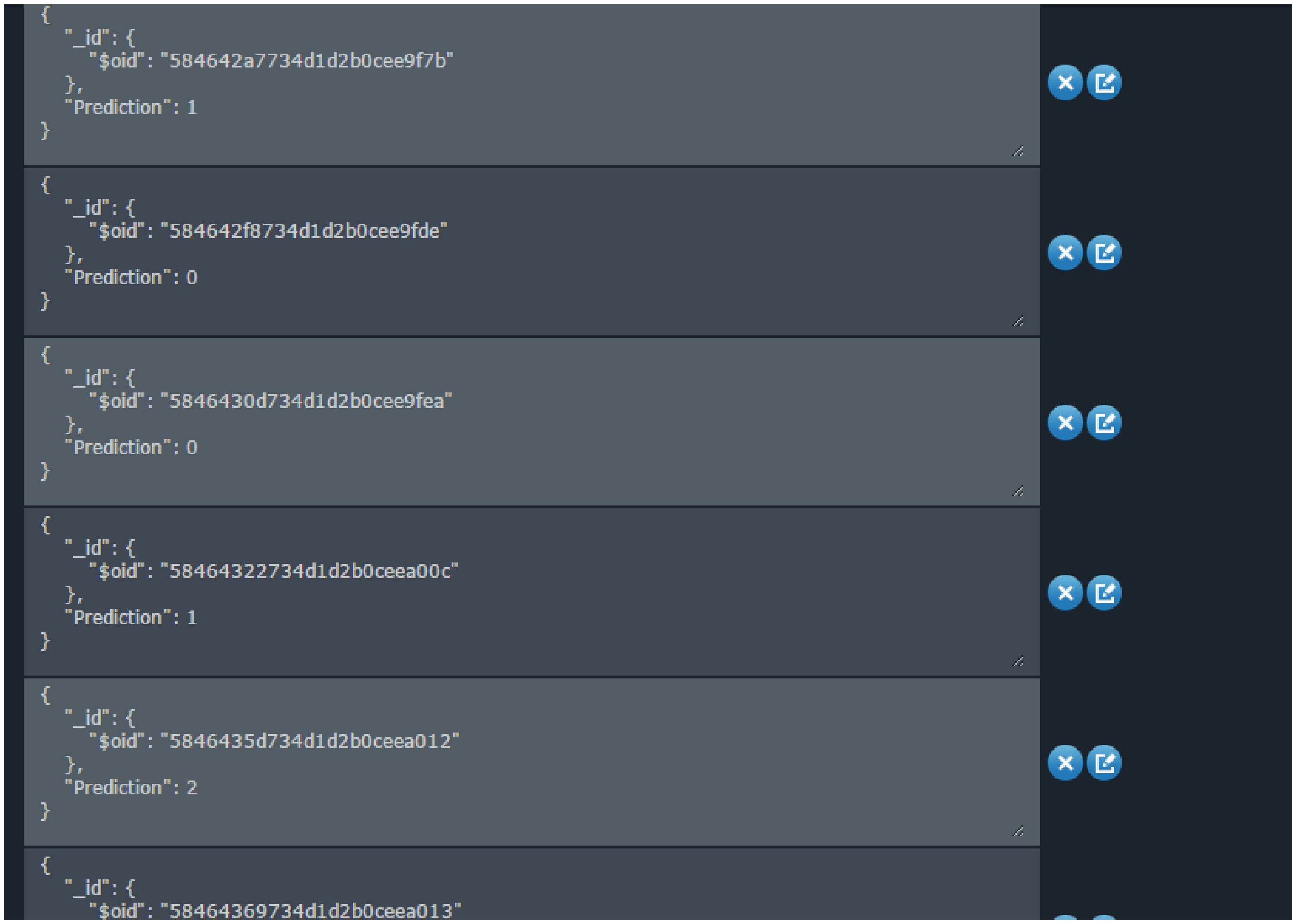


records / page 10 [1 - 1 of 1]

```
{
    "_id": {
        "$oid": "5843a69dbd966f5f1eeb0b72"
    },
    "Model": "DecisionTreeModel classifier of depth 2 with 5 nodes\n  If (feature 4 <=
0.004279044773625097)\n   Predict: 1.0\n  Else (feature 4 > 0.004279044773625097)\n   If
```

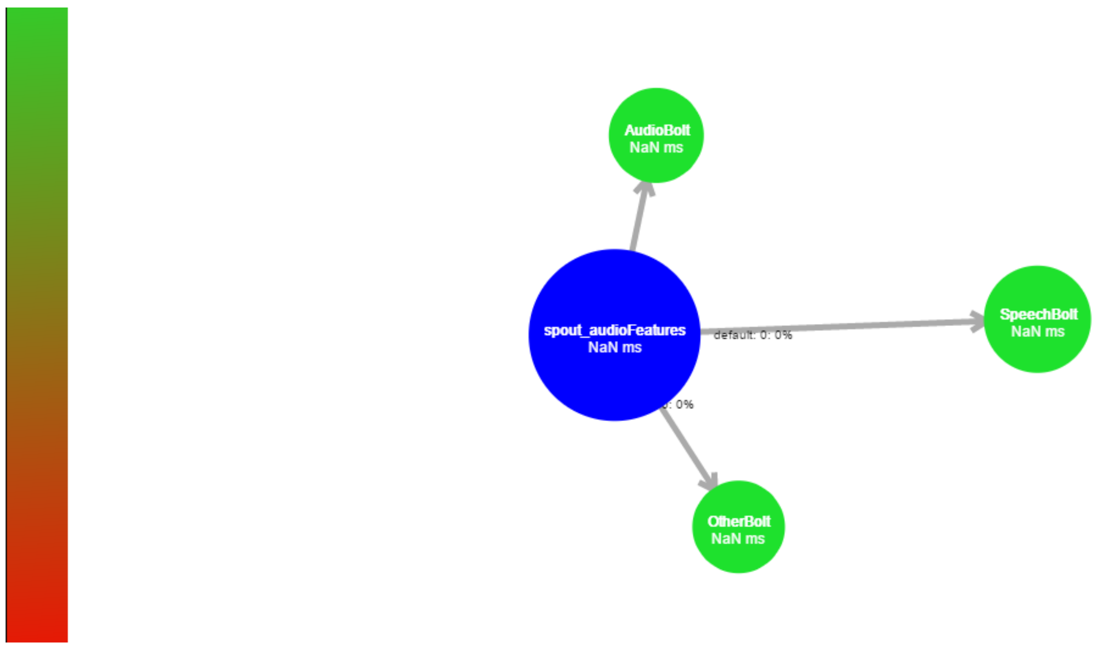
records / page 10 [1 - 1 of 1]

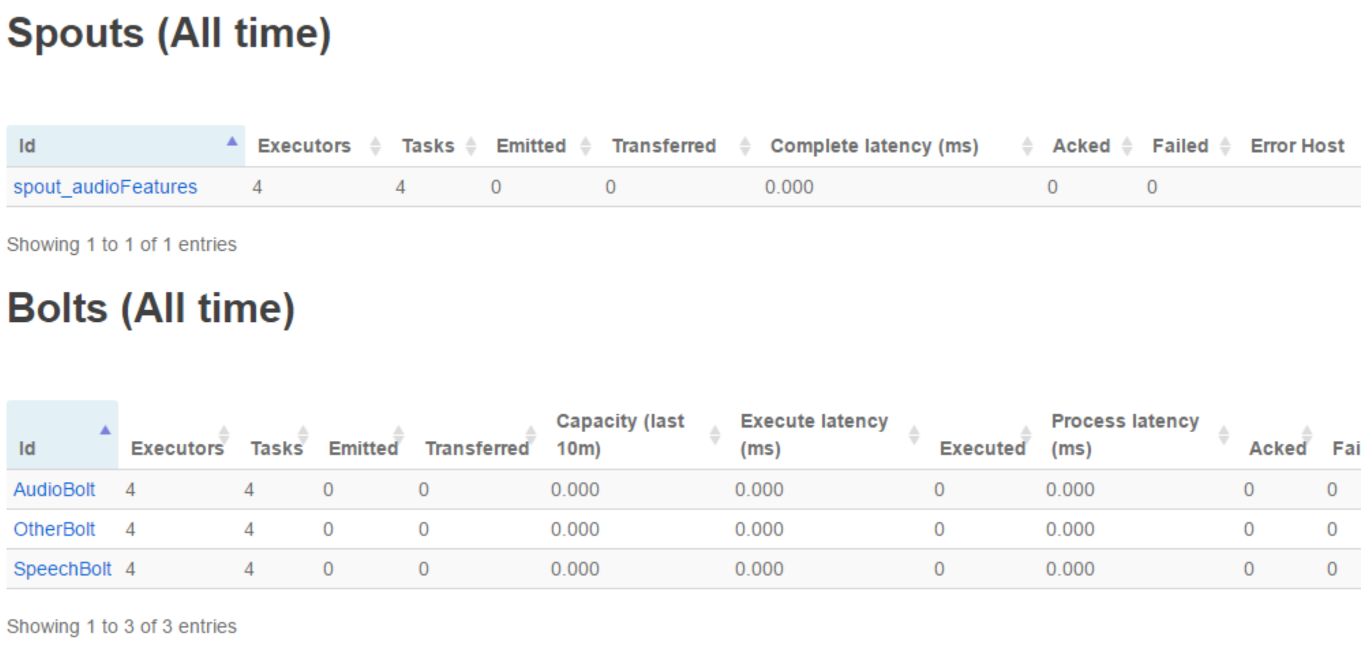
Results:



```
{
    "_id": {
        "$oid": "584642a7734d1d2b0cee9f7b"
    },
    "Prediction": 1
}
{
    "_id": {
        "$oid": "584642f8734d1d2b0cee9fde"
    },
    "Prediction": 0
}
{
    "_id": {
        "$oid": "5846430d734d1d2b0cee9fea"
    },
    "Prediction": 0
}
{
    "_id": {
        "$oid": "58464322734d1d2b0ceea00c"
    },
    "Prediction": 1
}
{
    "_id": {
        "$oid": "5846435d734d1d2b0ceea012"
    },
    "Prediction": 2
}
{
    "_id": {
        "$oid": "58464369734d1d2b0ceea013"
```

UPDATED TOPOLOGY:





Spouts (All time)

Id	Executors	Tasks	Emitted	Transferred	Complete latency (ms)	Acked	Failed	Error Host
spout_audioFeatures	4	4	0	0	0.000	0	0	

Showing 1 to 1 of 1 entries

Bolts (All time)

Id	Executors	Tasks	Emitted	Transferred	Capacity (last 10m)	Execute latency (ms)	Executed	Process latency (ms)	Acked	Fai
AudioBolt	4	4	0	0	0.000	0.000	0	0.000	0	0
OtherBolt	4	4	0	0	0.000	0.000	0	0.000	0	0
SpeechBolt	4	4	0	0	0.000	0.000	0	0.000	0	0

Showing 1 to 3 of 3 entries

6. PROJECT MANAGEMENT:

Work Completed:

- Phase 1:
    1. Successfully implemented Face identification and have count of number of students in the class using OpenIMAJ
    2. Used Microsoft Face API and Clarifai API to classify human faces and objects in a input image.
    3. Implemented Face recognition using OpenIMAJ by generating a model by selecting faces of the students.

- Phase 2:
    1. Successfully implemented Rest API call service and sending messages from Client to Kafka Consumer.
    2. Implemented Storm Topology creation by creating Spouts and Bolts for Data acceptance and processing in Storm.
    3. Implemented sending data from Apache Storm to Apache Spark. Tried running classification on input video by performing feature extraction and training and testing the model in Spark.
- Phase 3
    1. Successfully extracted the features from audio files and choose set of features for best classification of the data.
    2. Used Decision tree model to build/train the model.
    3. Pushed the model generated into mongoDB for further use.
    4. Successfully generated the storm topology with one spout for receiving the data and two bolts one for music features classification and other for speech features classification.
    5. Stored the results into mongoDB.
    6. Counted number of faces in a video frame.
- Responsibilities:
  For this phase everyone involved in all the tasks and contributed their parts.
  - **Nikhita Sharma** - Worked on all the tasks but mainly focused on the audio/video features extraction decision tree generation from the input features and pushed it to the MongoDB
  - **Ganesh Taduri** - Worked on all the tasks but mainly focused on accessing the decision tree from MongoDB and topology generation and testing the correctness of the model
  - **Aparna Pamidi** - Worked on all the tasks but mainly focused on the audio/video features extraction decision tree generation from the input features and pushed it to the MongoDB
  - **Harsha Komalla** - Worked on all the tasks but mainly focused on accessing the decision tree from MongoDB and topology generation and testing the correctness of the model

- Time Taken: 3 weeks
- Contributions:

Nikhita Sharma -	25%
Ganesh Taduri -	25%
Aparna Pamidi -	25%
Harsha Komalla -	25%

Work to be completed:

- Description:
    1. Still working to build a better architecture and data flow.
    2. Planning to generate more number of classes.
    3. Planning to use more training data to generate a better model.

- Responsibility:
  We plan to share the responsibilities dynamically according to the workload.
- Time to be taken: 10 weeks

Issues/Concerns:

Phase 1:

- While working with Microsoft Face API, most of the documentation of implementation was in C# or Python, which made it difficult to implement our modules in Java. Though we received some suggestions from Open Source support sites such as StackOverflow, we still had issues with the API keys.
- Most features provided by Clarifai API were not relevant to our project requirements. They were more generic to objects and not specific to Face Detection.
- Still working on the logic to generate a better model for classroom scenario.

Phase 2:

- We need more clarity with the workflow of the process
- Need some feedback on integration of Kafka, Storm and Spark and which process specific technology could be used for each analytics like feature extraction, training the model and testing the model.
- Concerns on where feature extraction should occur , for example on Client side or in Storm or Spark. If it is on Client side, then how do we utilise Apache Storm features in the workflow.

Phase 3:

- Faced some issues during generating the topology. Most of the time we couldn't identify the problems due to server down.
- Had some issues with video annotations.

Phase 4:

Project GitHub repository:
https://github.com/nikhitasharma/RTBigDataAnalytics_Project

Future Work:

- Addition of additional features such as surveillance system for exit and entry of students in the class at various points of time during the class

- This project can also be extended to surveillance systems for public meetings or gatherings.

REFERENCES:

[1]. http://openimaj.org/openimaj-image/faces/dependencies.html
[2]. http://openimaj.org/tutorial/sift-and-feature-matching.html
[3]. http://openimaj.org/tutorial/eigenfaces.html
[4]. http://klresearch.org/IJMSTM/papers/v2i3_2.pdf
[5]. http://blog.mashape.com/list-of-10-face-detection-recognition-apis/
[6]. http://ieeexplore.ieee.org.proxy.library.umkc.edu/stamp/stamp.jsp?tp=&arnumber=6227479
[7]. http://storm.apache.org/releases/1.0.2/Concepts.html
[8]. https://kafka.apache.org/documentation
[9]. http://spark.apache.org/docs/latest/mllib-decision-tree.html
[10]. https://tomcat.apache.org/tomcat-6.0-doc/deployer-howto.html