RACA- Real Time Automated Classroom Attendance

An Automatic Attendance Management System using Face Recognition

Project Report - 3

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/hall. In class room scenario the system gives information whether a particular student attended the class or not using face recognition.

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:

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:

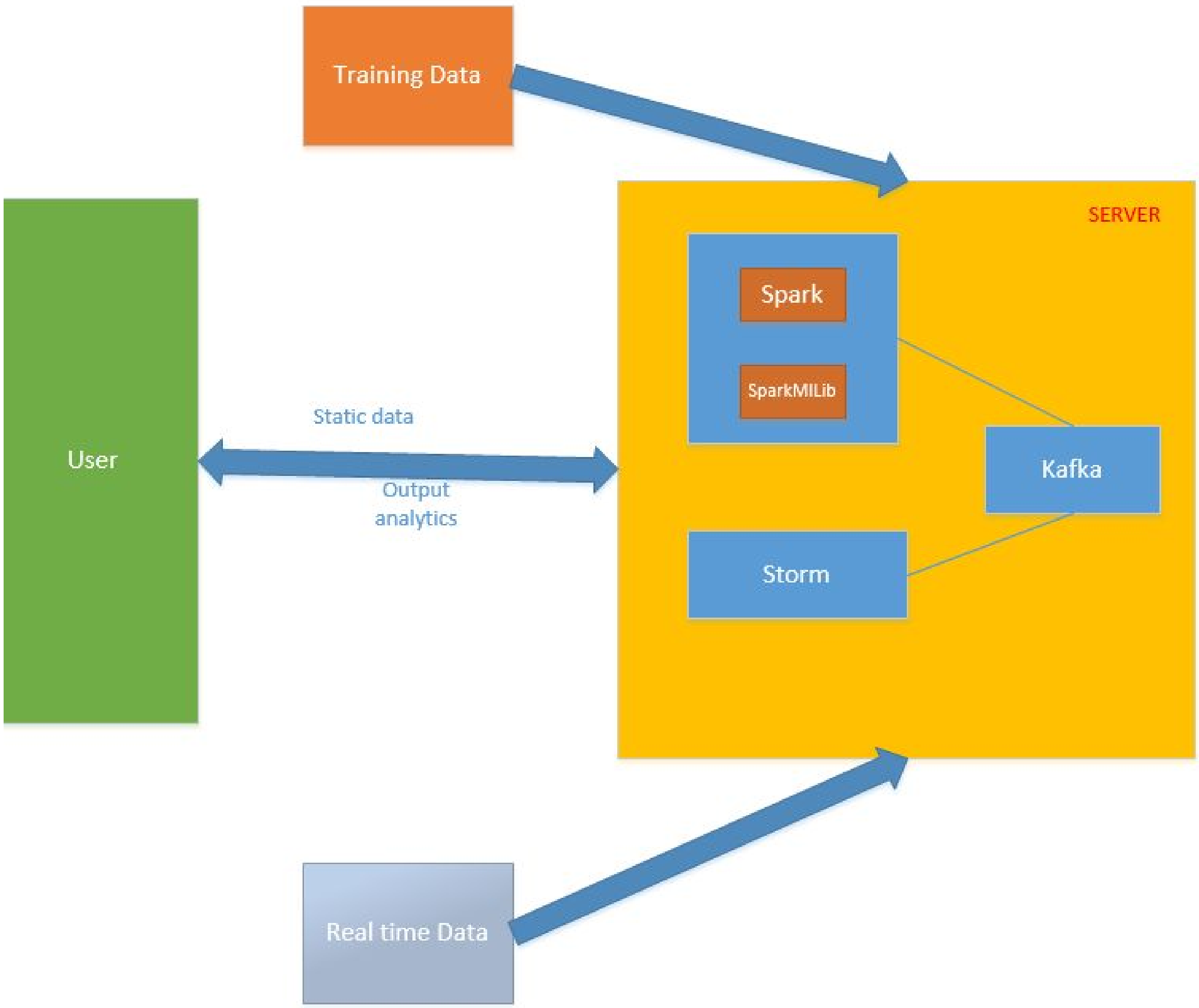


Fig 1: Architecture Diagram

Software Architecture:

Spark:

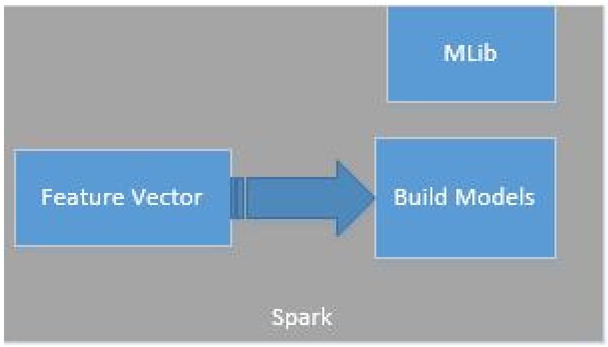


Fig: Spark Architecture

Storm:

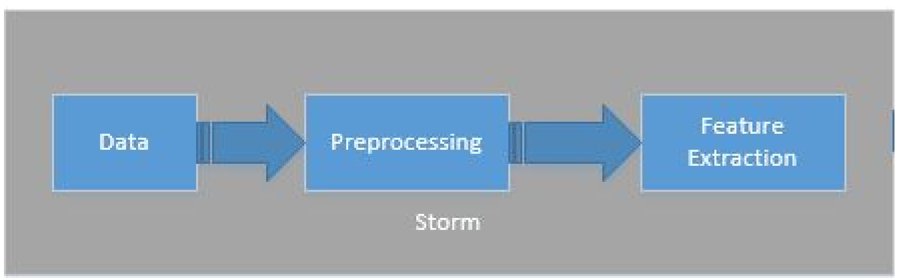


Fig: Storm Architecture

Kafka:

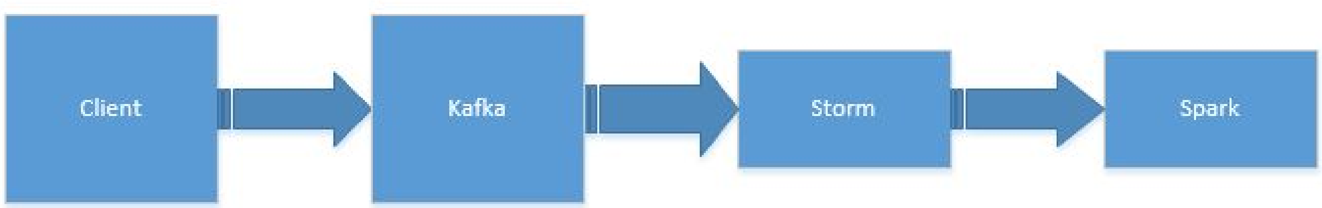


Fig: Kafka Architecture

System Flow

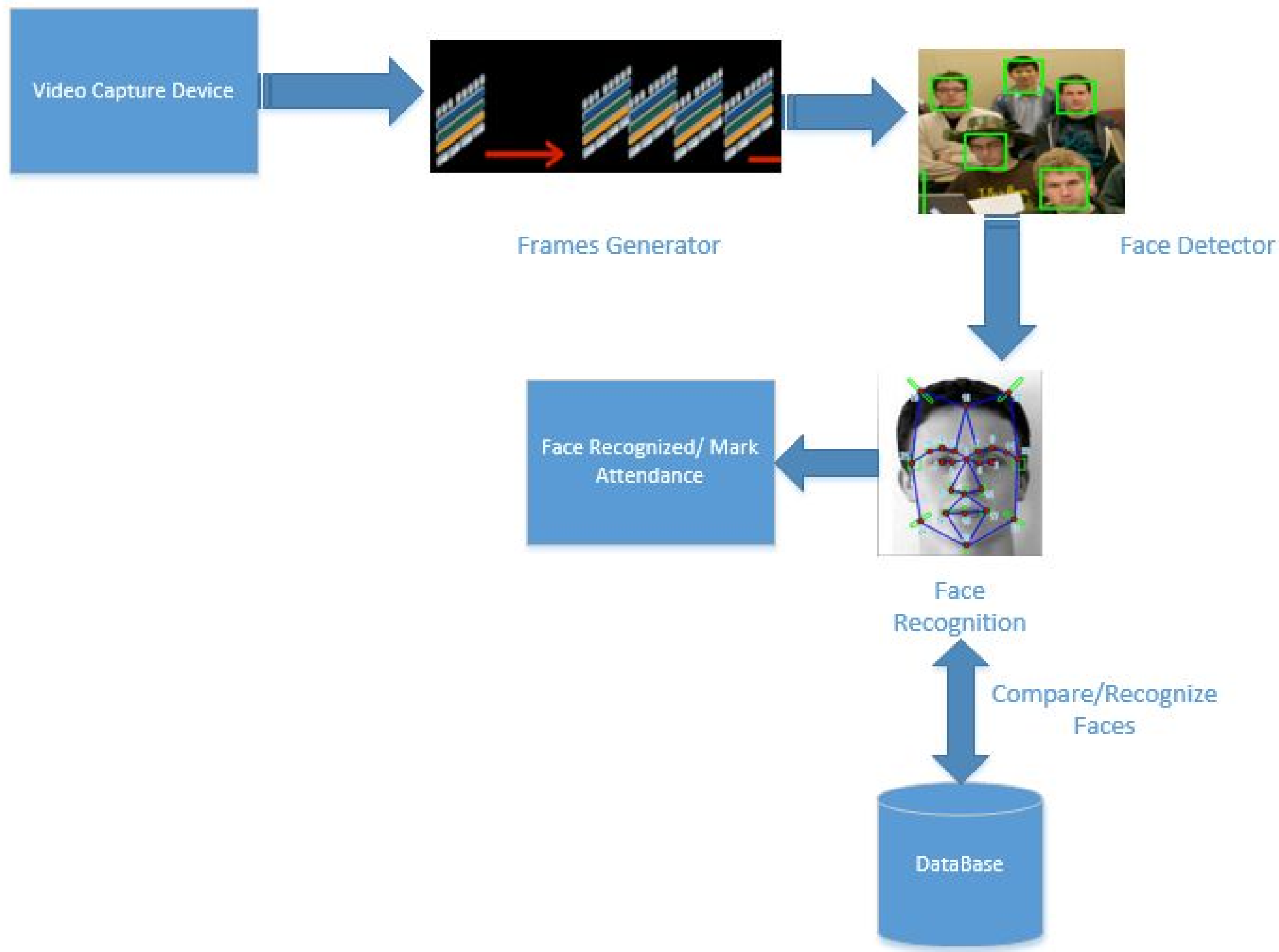


Fig 2: System Flow Diagram

Class Diagram:

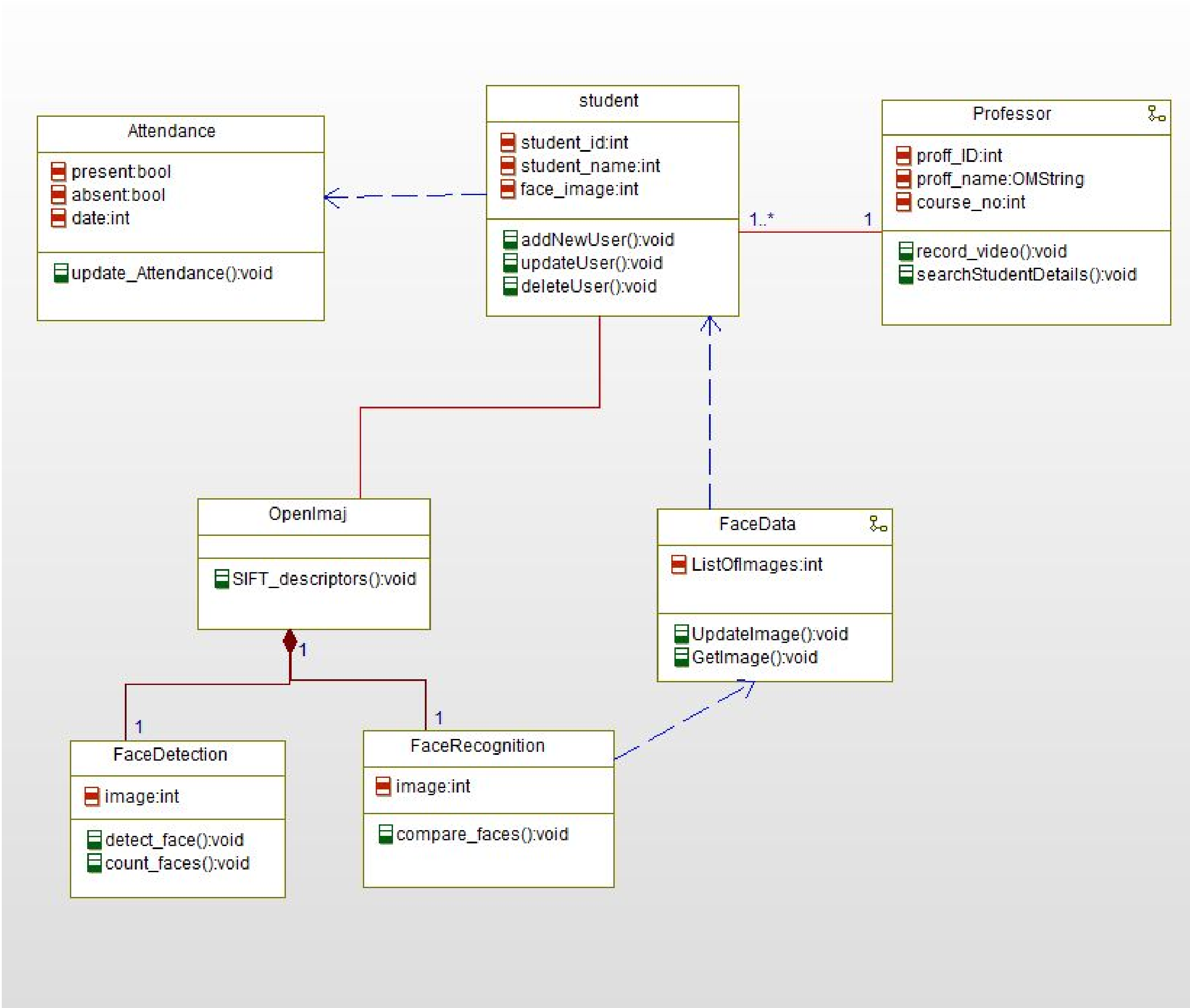


Fig 3: Class Diagram

Sequence Diagram of Whole System:

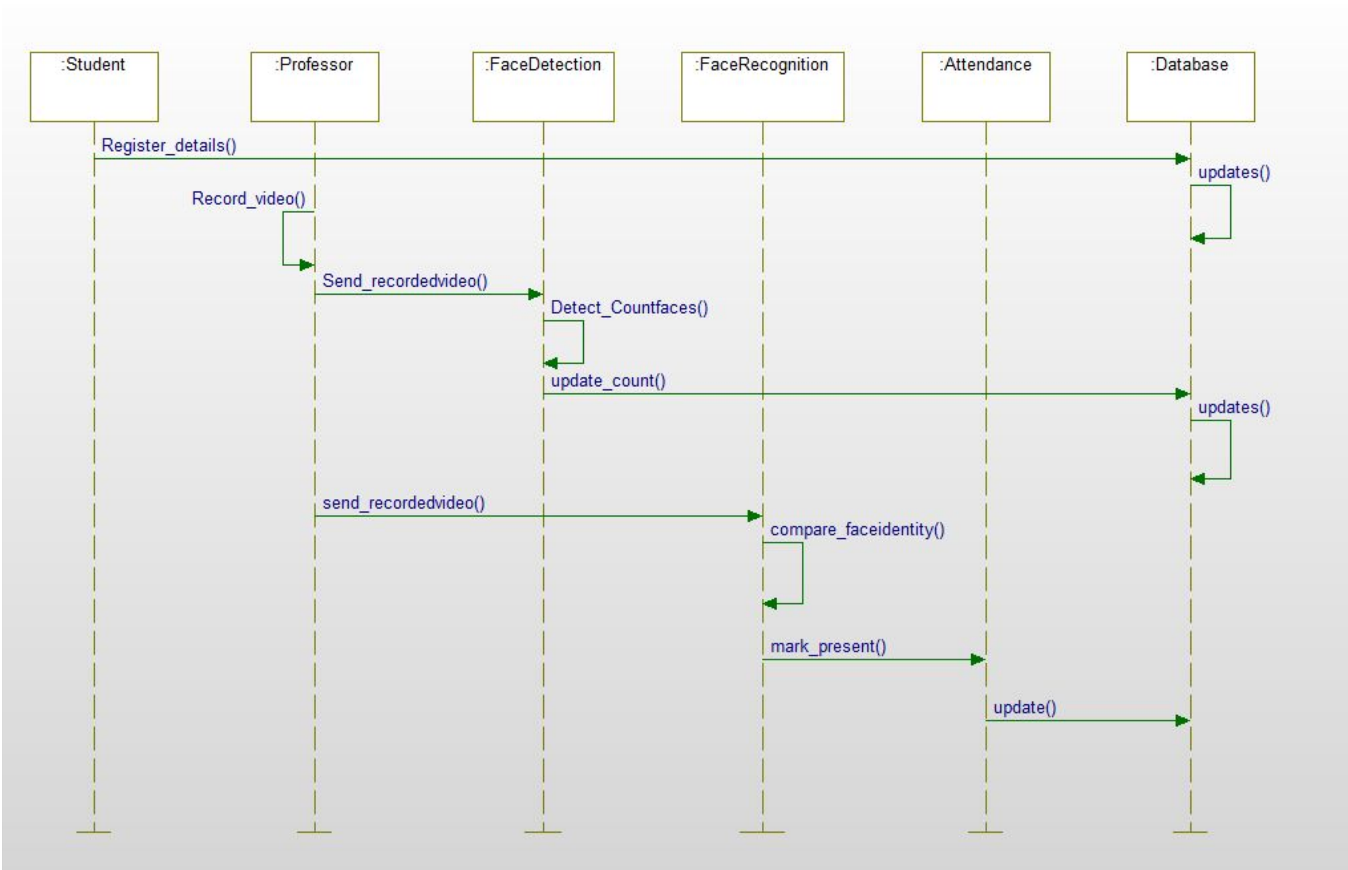


Fig 4: Sequence Diagram

Activity Diagram:

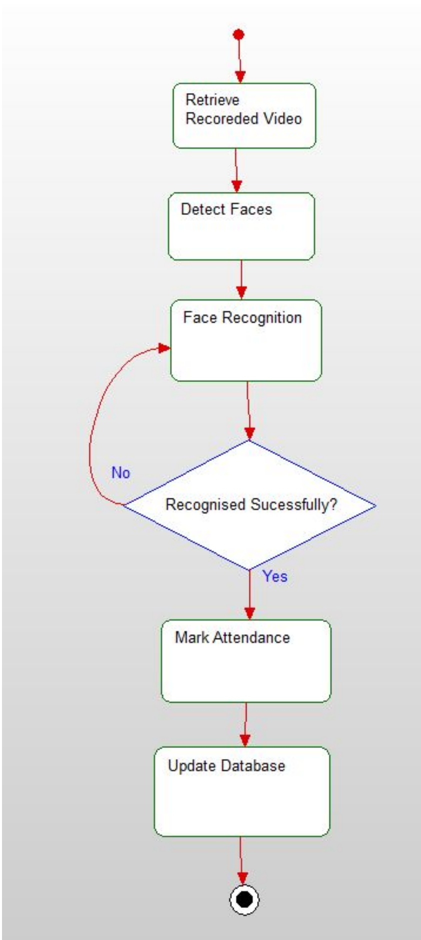


Fig 5: Server Activity Diagram

Sequence diagram:

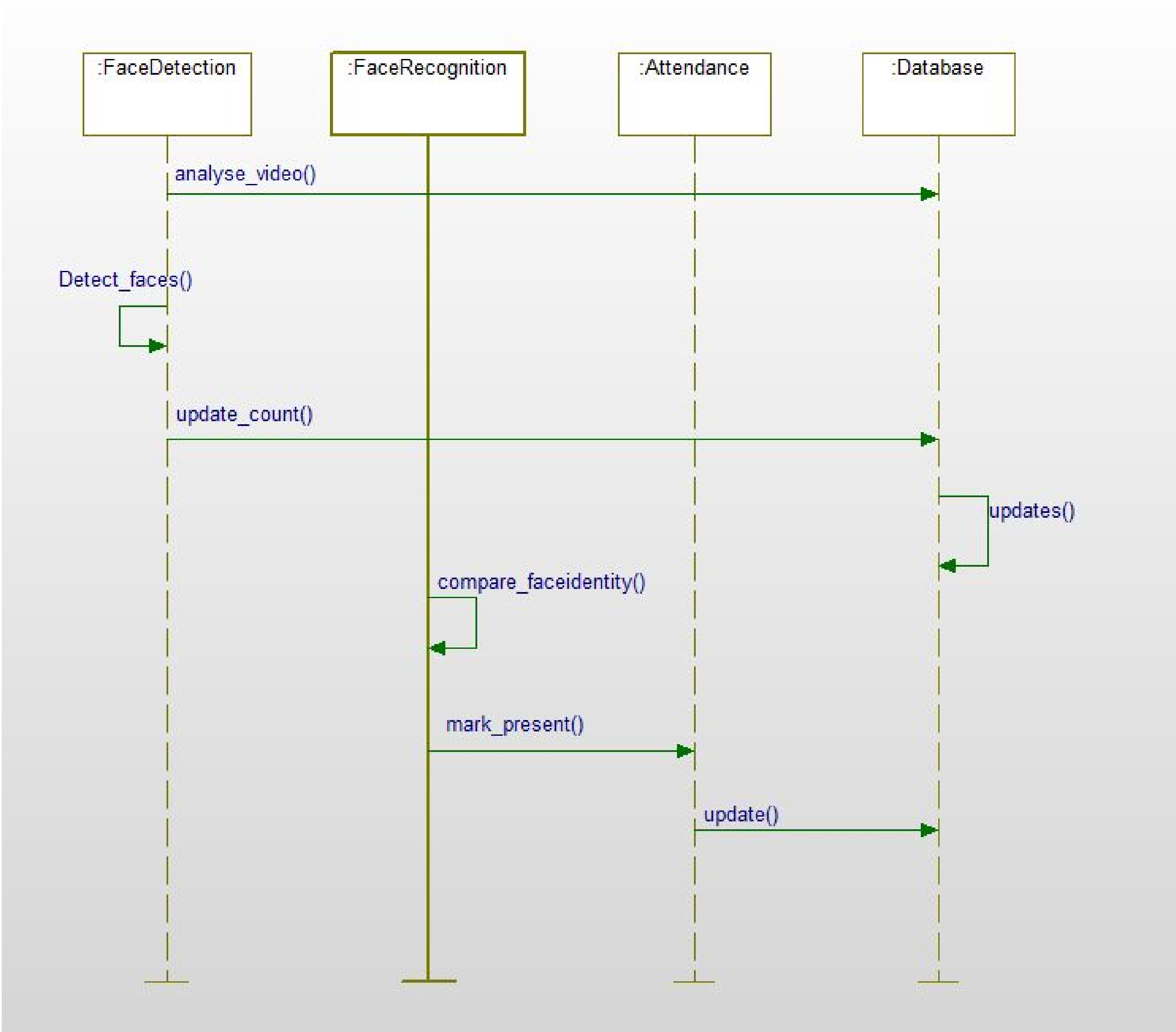


Fig 6: Server Sequence Diagram

Design of Mobile Client:

Activity Diagram:

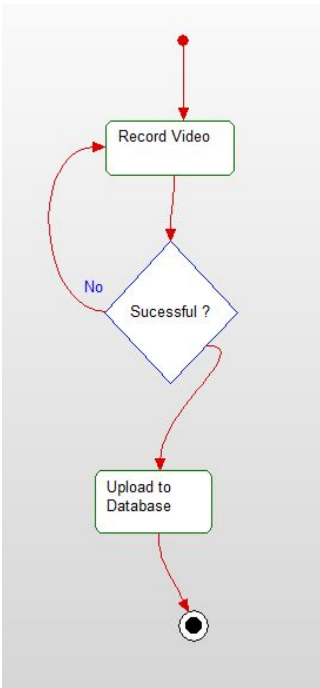


Fig 7: Client Activity Diagram

Sequence diagram:

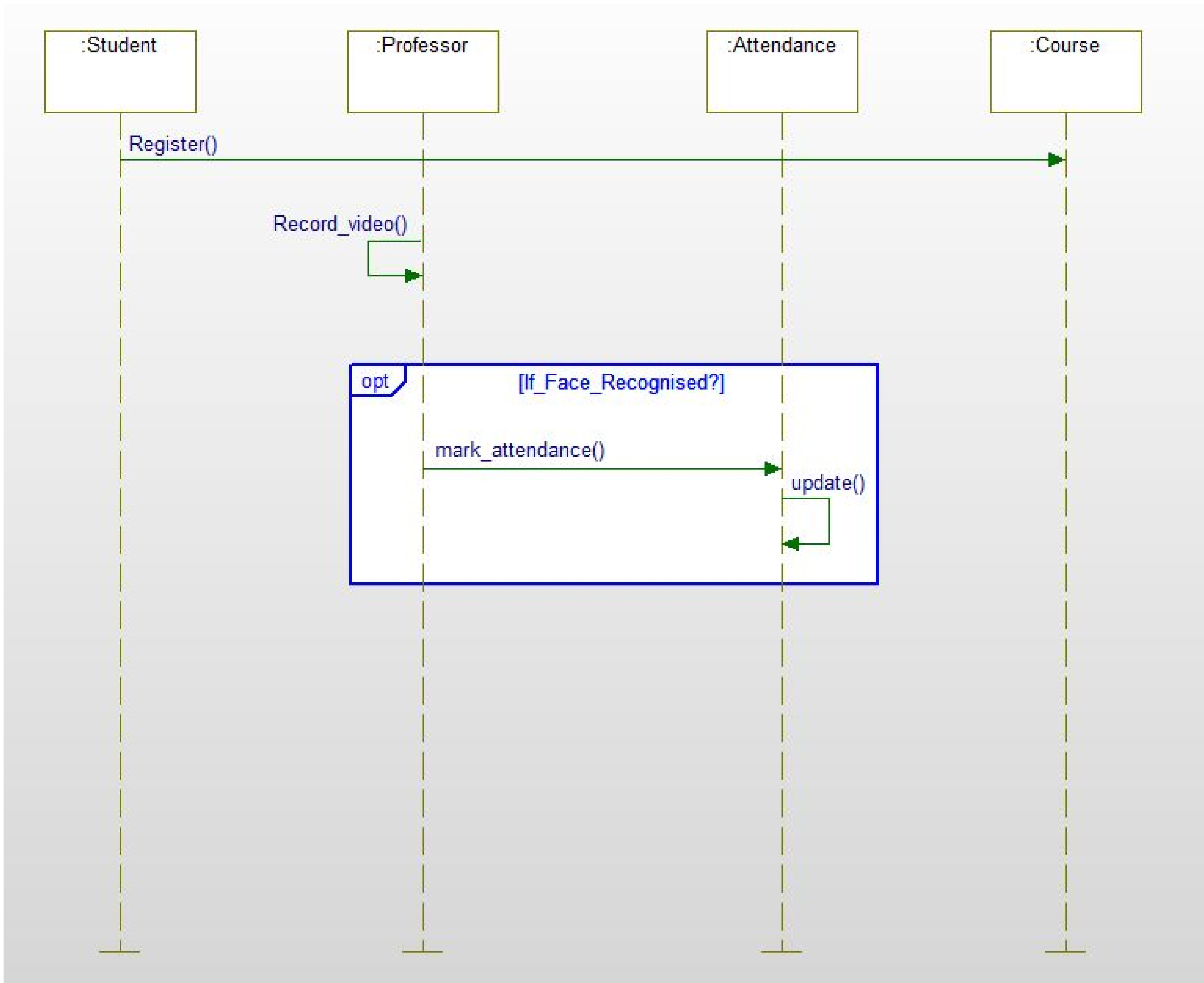


Fig 8: Client Sequence Diagram

Existing Services/APIs:

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

5. IMPLEMENTATION:

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.

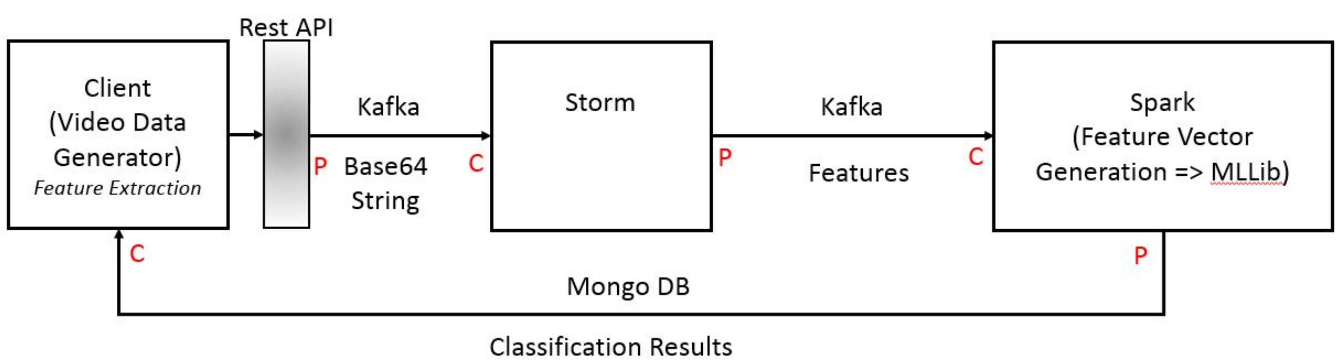


Fig: Overview of Work-Flow Architecture

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 two bolts in the storm topology and save the results into Mongodb. The whole process is explained in detail in various sub tasks below.

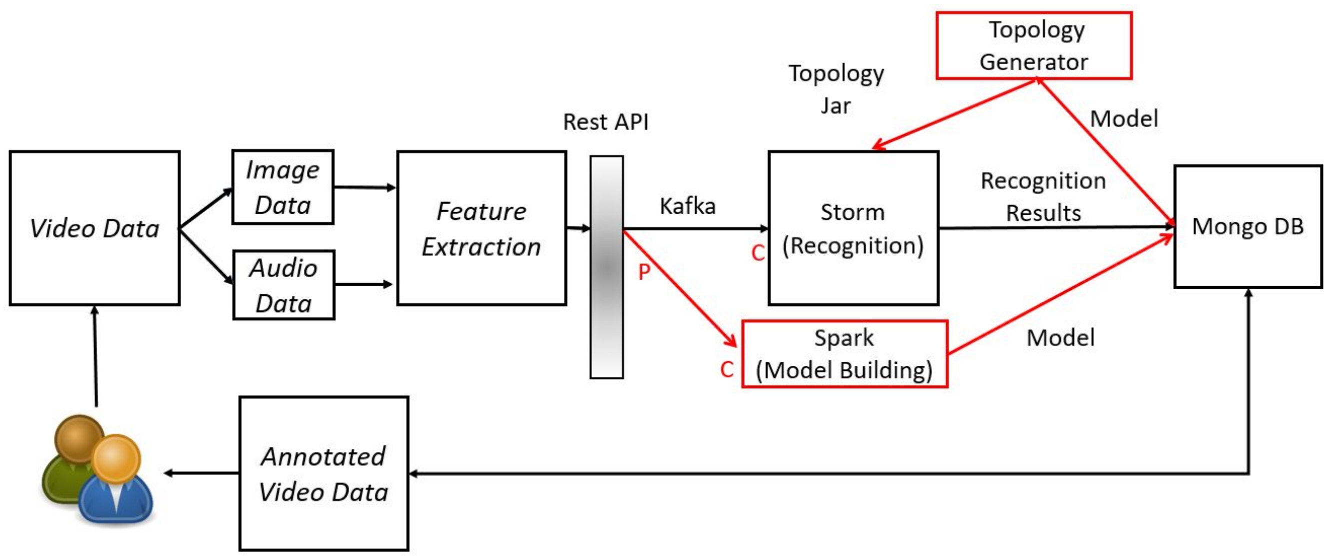


Fig: WorkFlow Architecture for Phase3 Implementation

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.

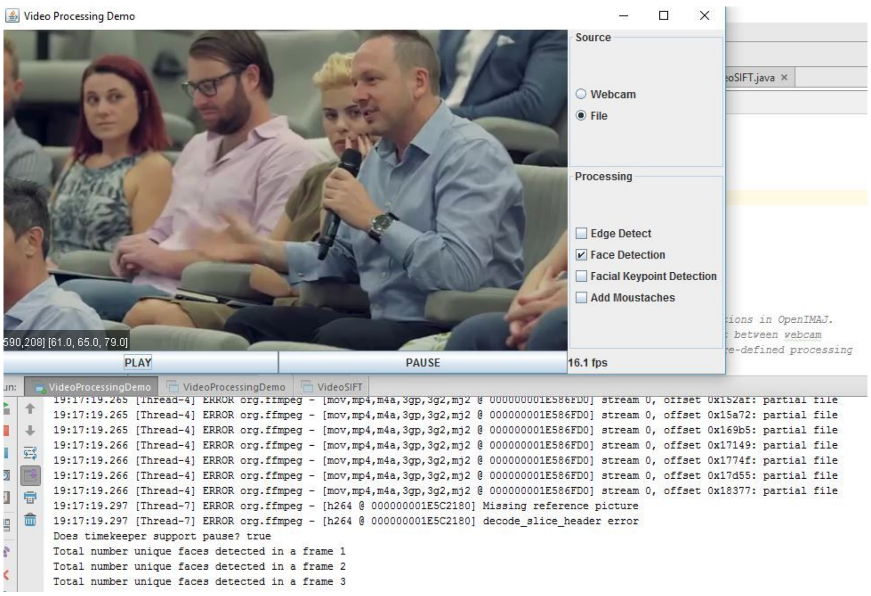
.

6. <u>DOCUMENTATION:</u>

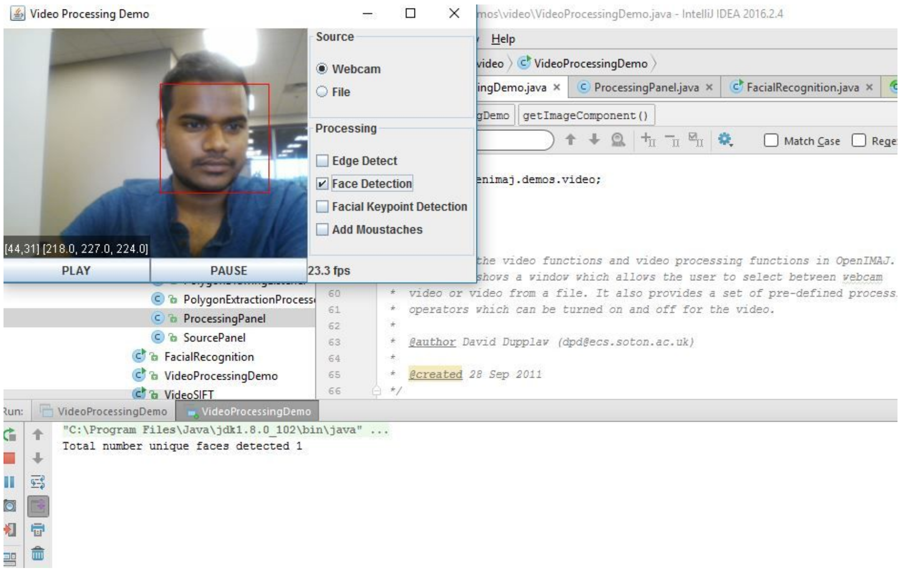
<u>PHASE1:</u>

RESULTS USING OPENIMAJ:

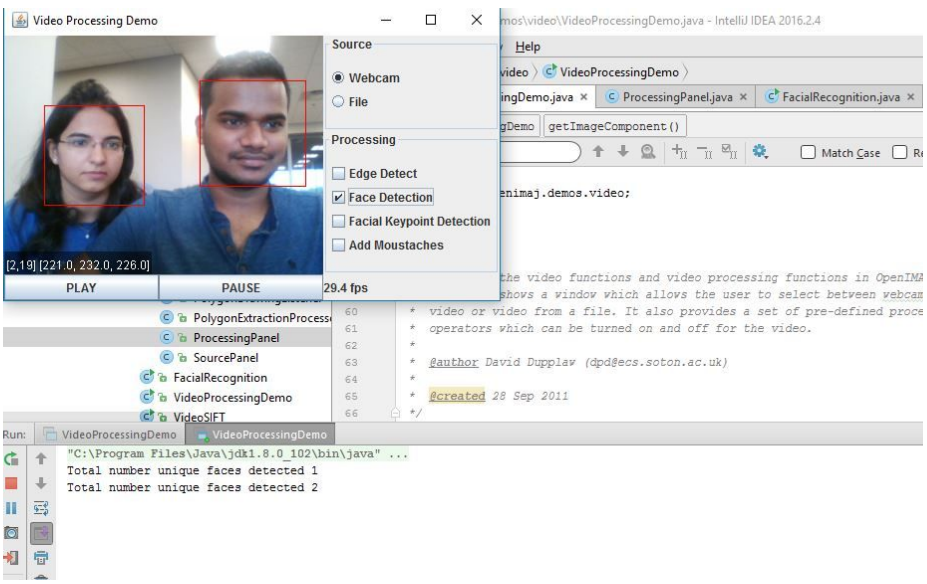
1. Static Classroom Video input - 3 faces detecting showing number of students count:



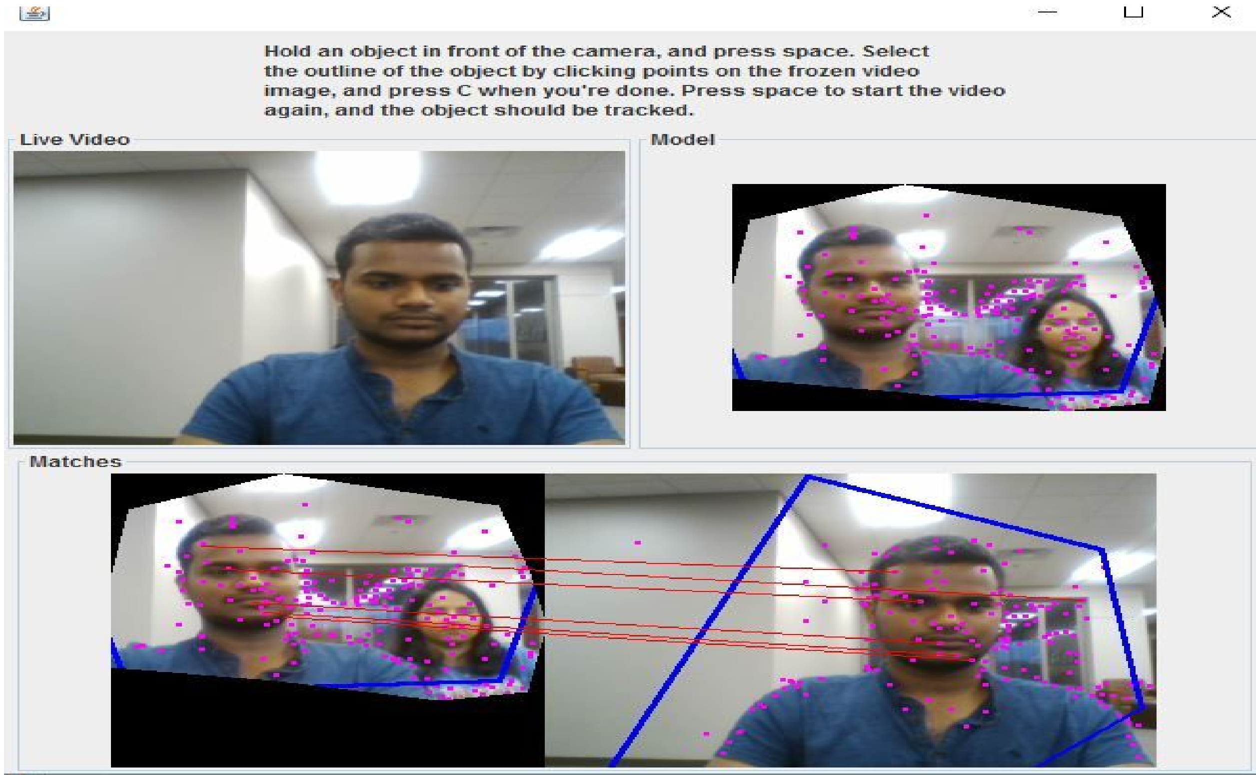
2. Live input using WebCam showing One face detected:



3. Live WebCam input showing Two faces detected:

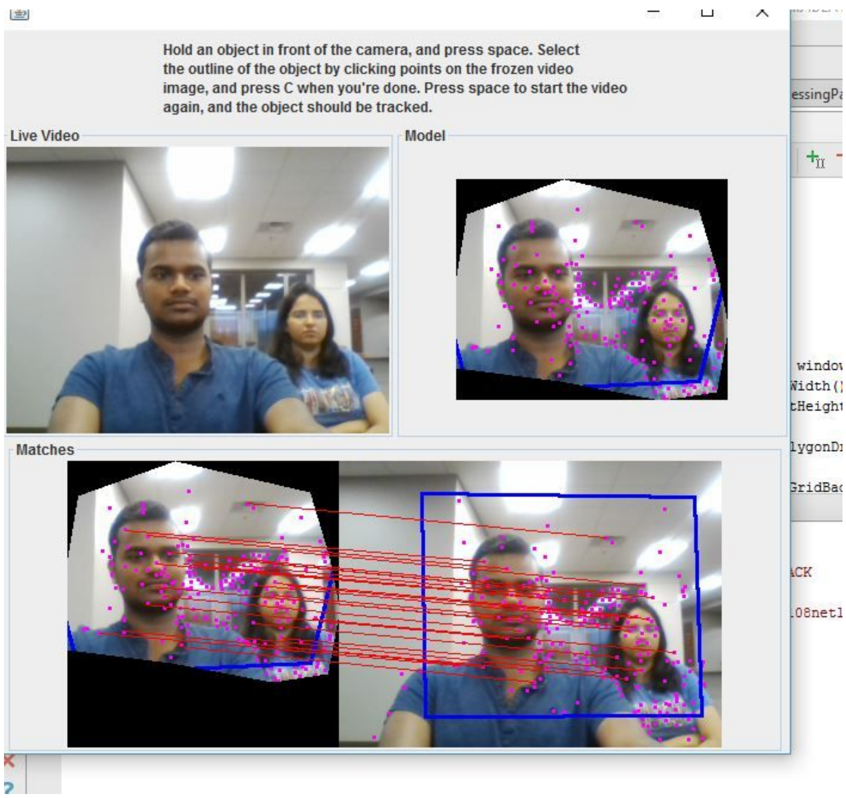


4. Live video input showing Face Recognition (Recognizing students facial features and matching them using SIFT). One match when one student p



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.



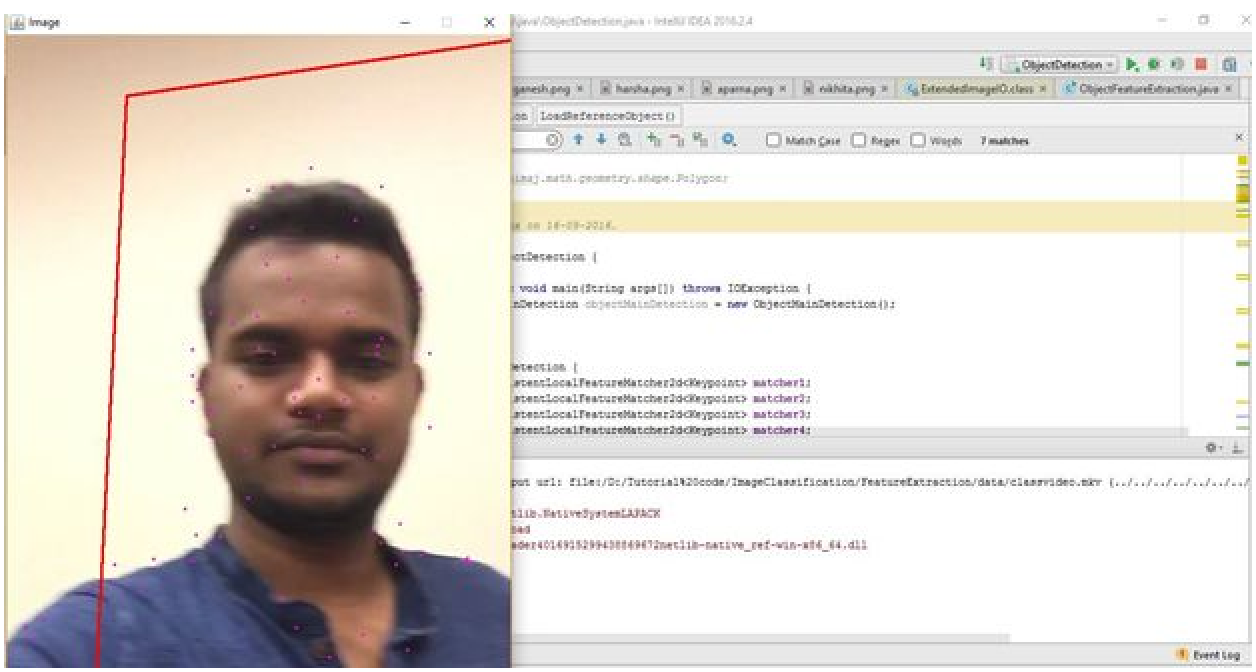


Fig: Classification of individual student in a video data from the model generated using the image key features

Phase2 :

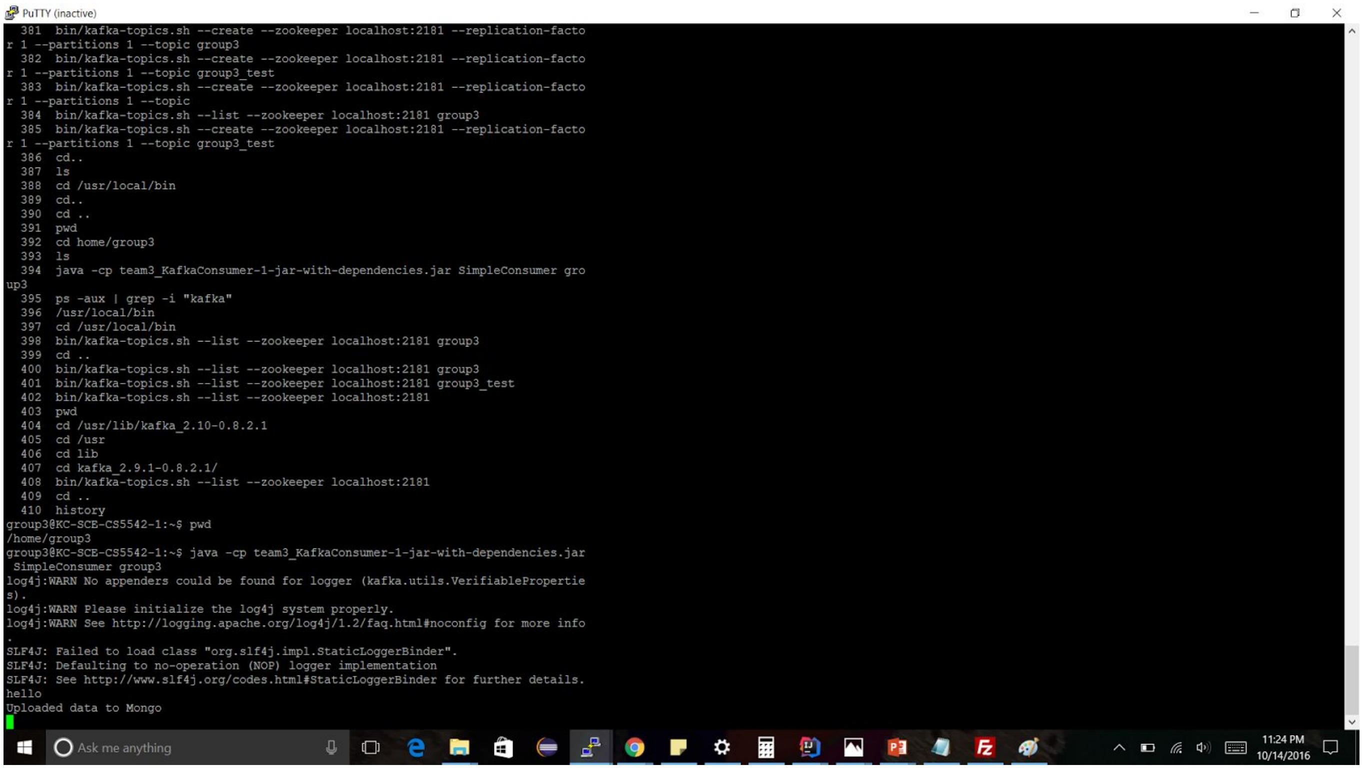


Fig: The data sent to the Kafka using RestApi

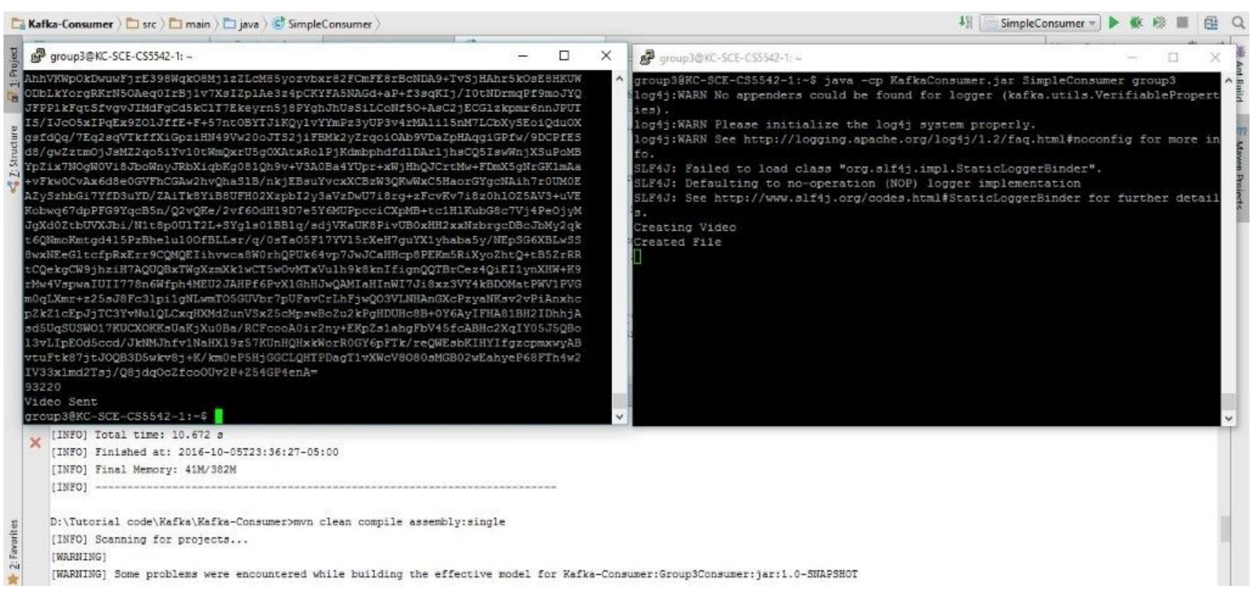


Fig: Video sent to the Storm using Kafka

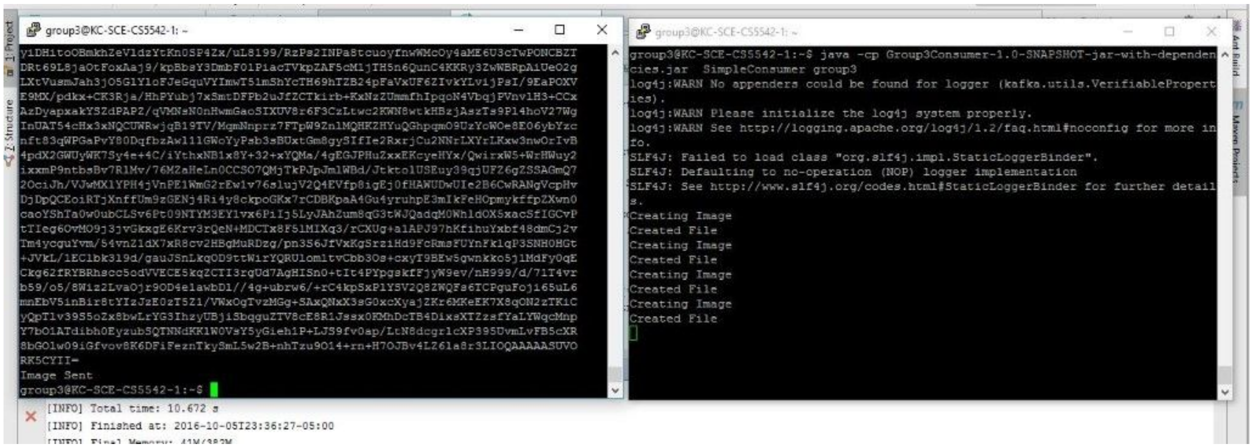


Fig: Image sent to the Storm using Kafka

Phase 3:

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

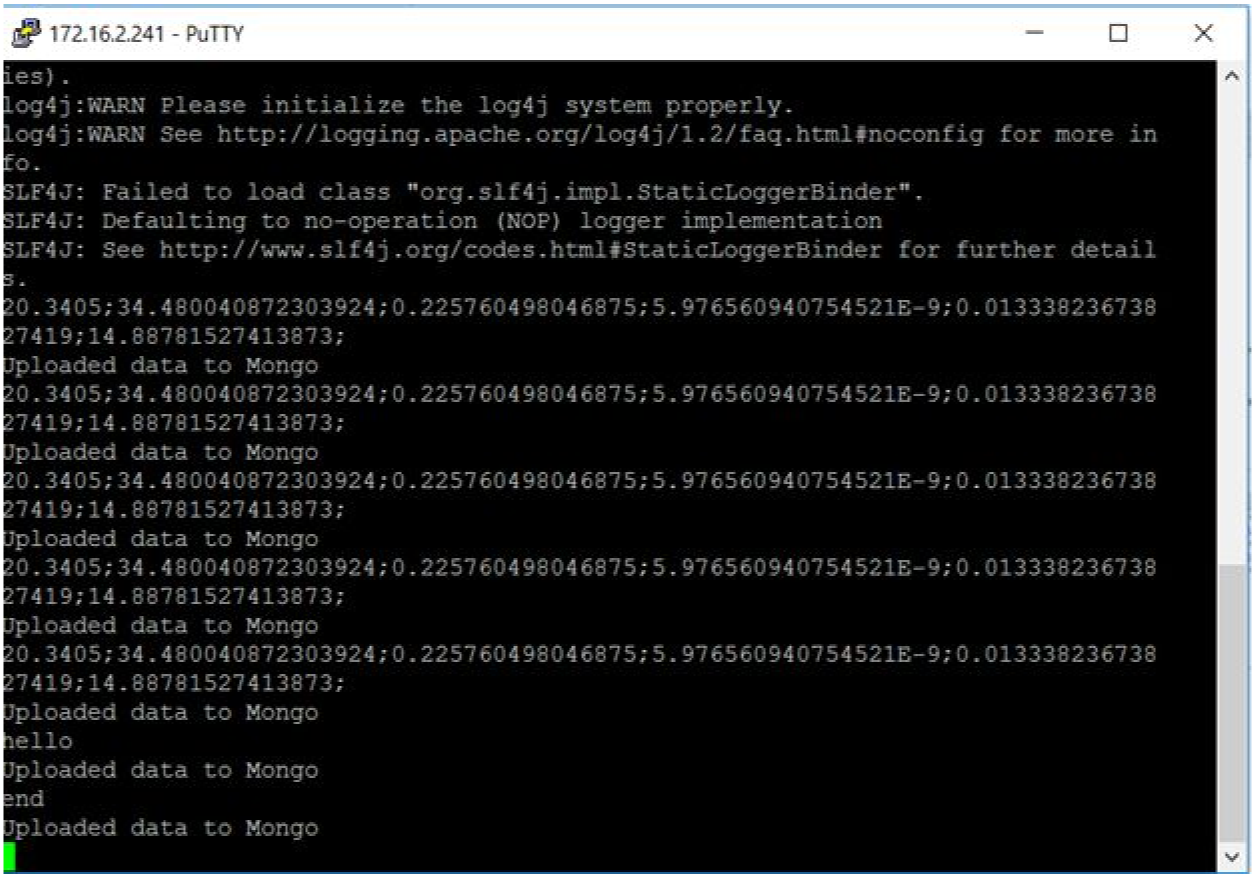


Fig: The figure depicts that the data has been successfully uploaded to the MongoDB

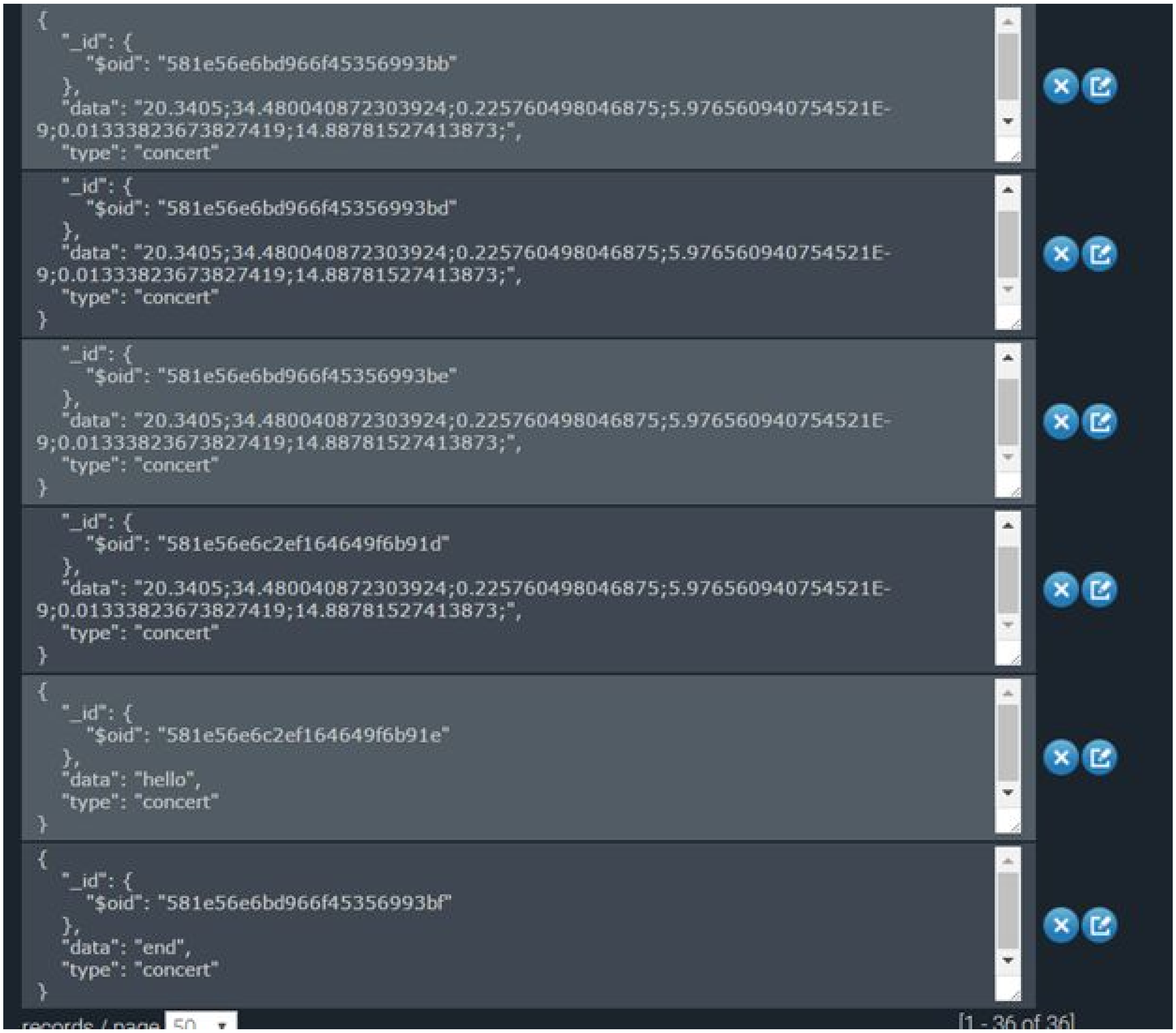
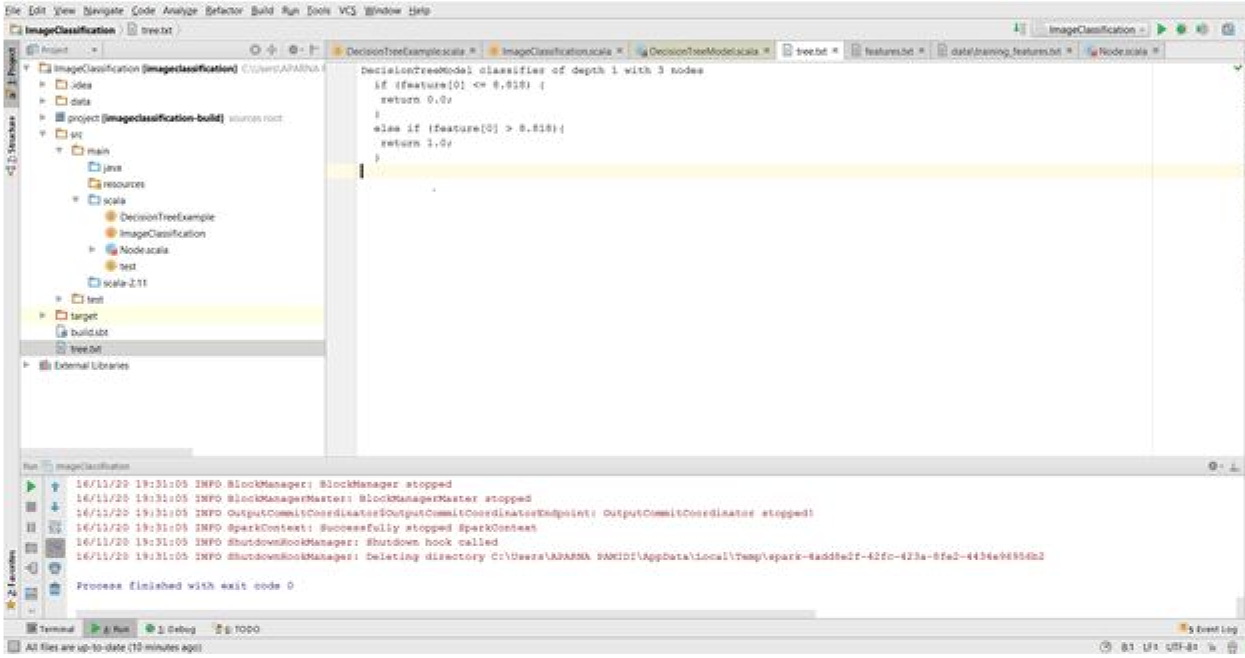


Fig: Screenshot of data being stored in tuples format in mongoDB



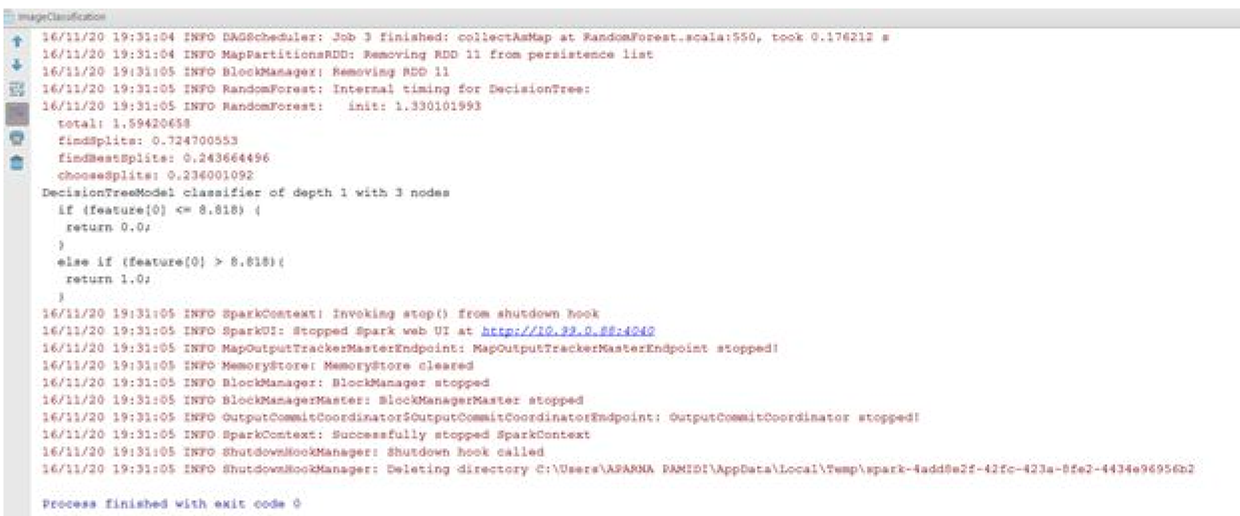


Fig: The figure depicts that generation of decision tree from the input features file

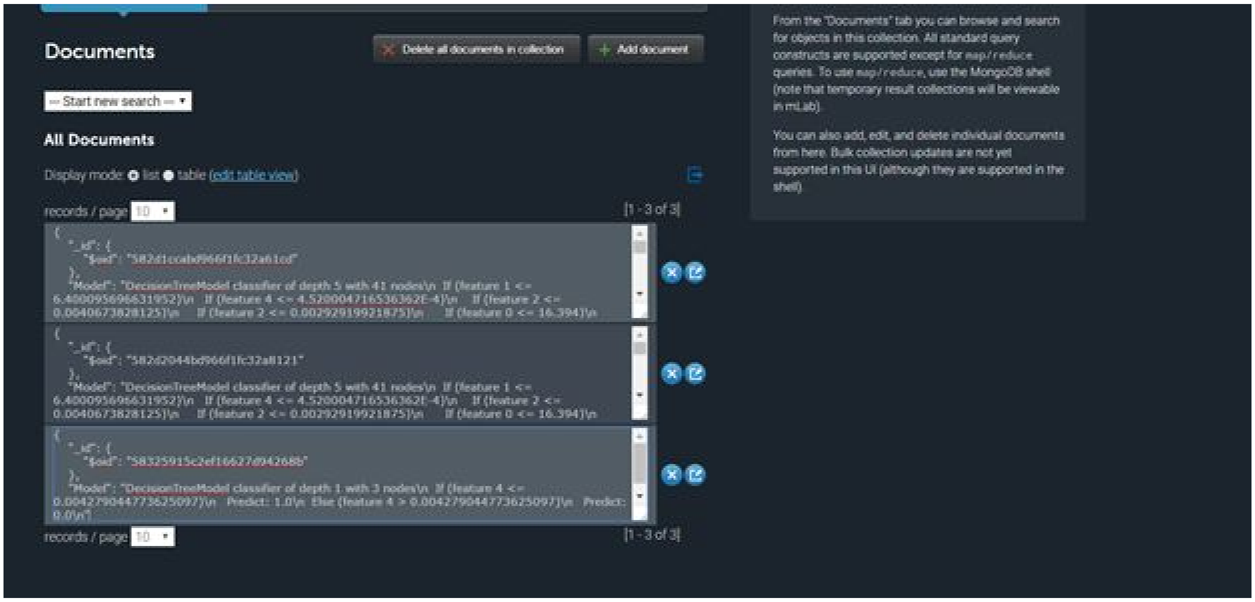


Fig: The figure depicts the decision tree stored in MongoDB

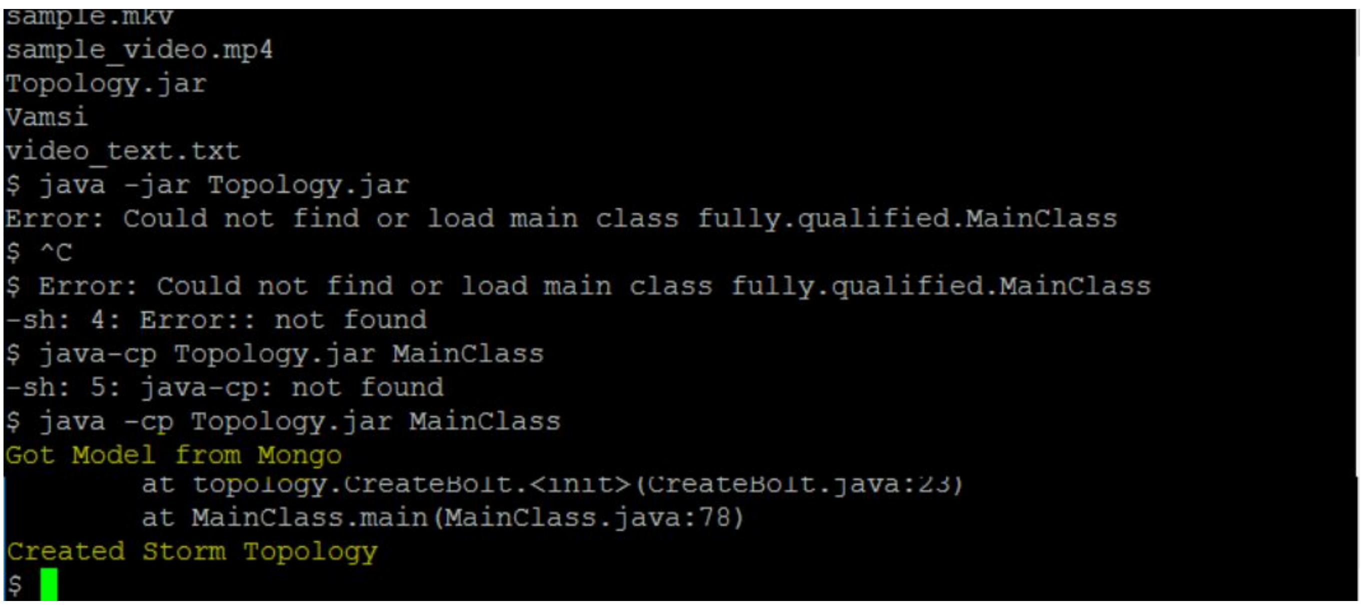
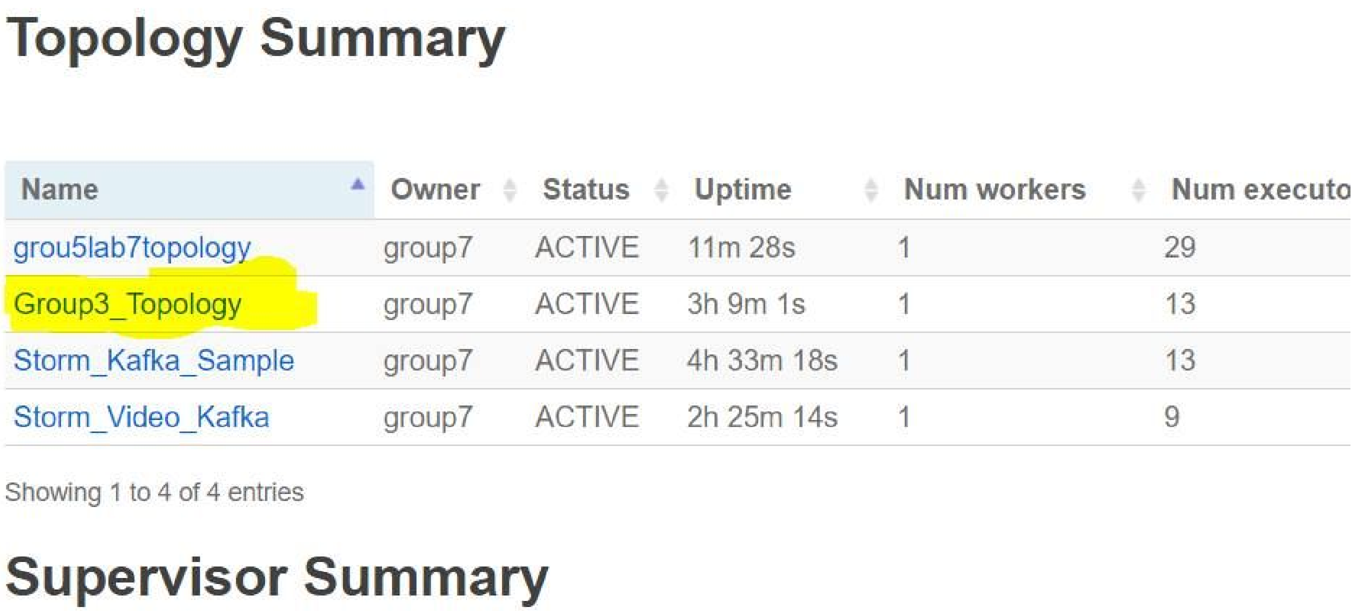


Fig: Kafka consumer received the model from the MongoDB and generated topology



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

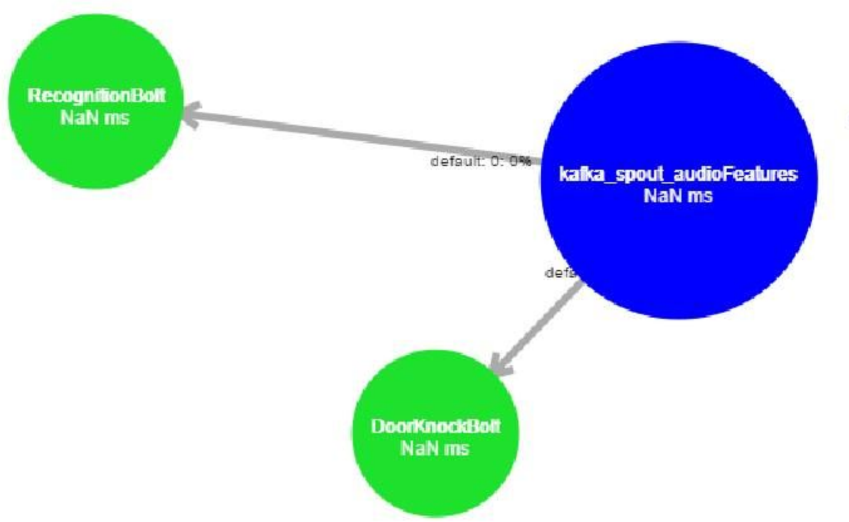


Fig: Figure depicts the creation of spout and bolts generation

7. 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.

  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