



POLSKO-JAPOŃSKA WYŻSZA SZKOŁA  
TECHNIK KOMPUTEROWYCH

## Information Technology

### Computer Networks

System and computer network programming

### Artificial Intelligence

Intelligent systems of data processing

# Distributed speaker diarization system.

**Piotr Sukiennik**

s7526

**Type of thesis:**

Engineer's thesis

**Supervisors:**

Witold Kosiński, Professor

Paulina Adamska, M. Sc.

Warsaw, January 2013

*Distributed speaker diarization system*  
*Piotr Sukiennik*

## Table of contents

Preface.....	4
Chapter 1.....	5
Introduction.....	5
Speaker Diarization.....	6
Chapter 2: Tools and Technologies Overview.....	7
JAVA.....	7
Spring Framework.....	7
Apache Maven.....	7
Apache HTTP Server.....	8
Apache Tomcat.....	8
HAproxy.....	8
Chapter 3: Architecture.....	9
System architecture.....	9
Request processing flow.....	10
Chapter 4: Frontend.....	11
Request flow.....	12
Technologies.....	12
Spring MVC.....	12
jQuery.....	12
Modules.....	13
Frontend Module.....	13
Data submit Module.....	14
Chapter 5: Backend.....	15
Request flow.....	15
Technologies.....	15
Memcached.....	15
Modules.....	16
Backend Logic Module.....	16
Balancer Module.....	16
Converting Module.....	16
Technologies Used.....	17
FFmpeg.....	17
Processing Module.....	17
Tools used.....	18
MFCC .....	18
Speaker Diarization.....	18
MFCC parameters .....	18
Classification Module.....	19
Clustering.....	19
EM.....	19
kMeans.....	20
xMeans.....	20
Technologies Used.....	20
Weka.....	20
Splitting Module.....	21
Chapter 6: Future updates .....	22

Chapter 7: Conclusion.....	23
Example Interface Interaction.....	24
References.....	30

## Preface

The original intention of my work, and the first version of this engineering thesis was to make a voice authentication system. After many tests using various classification and features extraction methods it occurred to me that such system is too error prone and provides not enough reliability for usage in authentication.

I've changed draft of my work to speaker diarization. Using experience I've gained working with text independent speech feature extraction and classification I was able to create working system that provided greater reliability and was less error prone due to different nature of the problem.

# Chapter 1

## ***Introduction***

Goal of this project was to build an accessible and scalable application solving problem of speaker diarization.

The user of the application selects an audio file containing speech fragments, specifies number of actors occurring in an audio file and uploads it to the servers. The input file has to be converted for homogenous feature extraction, therefore users can upload files of various audio formats.

The output of a web application are audio files containing audio fragments of classified actors found in input audio file and a time-line of occurrences. User is notified with an email when the request processing is completed and can download these files.

System is build in a form of web application making it accessible for users across the globe. Due to the performance stress that is put on a hardware during modules step, the system has been divided to modules making it horizontally scalable. Such separation of responsibilities in a web application also greatly reduces maintenance cost of updates. There is only one use case of the web application that leads from uploading file to the server to obtaining segmented audio files that correspond to speakers detected in the recording. Such files are then downloadable by the user. Because speaker segmentation process is complex, single request processing could take some time. Therefore email-notification system has been introduced. The user will obtain email from the server containing the link to the processing result, once the request has been completed by the application.

Keeping the request processing and computation on the server side allows to implement various solutions. Web applications advantage is it's portability. User does not have to download any part of the application and uses just the interface, that can be accessed from anywhere. Furthermore any updates that are done to the application itself can be transparent for the users and does not require any intervention. Taking advantage of the web application's accessibility, mobile interface application could be developed that could upload recorded data (recorded meetings or conferences) and receive split results.

The web application is not released to general public at the moment. The concept name of it is "whowhen" and it is assumed that it is accessible under URL "<http://whowhen.com>" for convenience.

## **Speaker Diarization**

Speaker diarization (called also speaker segmentation) is a problem in computer science of assigning speech fragments of a speech audio document to speakers.

Speaker diarization is most commonly used in speaker segmenting of telephone conversations, recorded conferences and radio transmissions. The increasing size of audio archives that is hard to index and search through makes subject of Speaker Diarization important, because it provides additional information about the structure of an audio file, in form of a searchable textual format, making archives more accessible [1].

There are many methods of speaker diarization depending of a prior information about an input audio document and nature of the problem. The segmentation itself can be made dependently on textual information that is spoken by speaker, joining the problem of speaker diarization and voice recognition. The problem could also be solved involving higher level analysis of the spoken words, analysis of accent and the way separate phonemes are being spoken.

The conspect of this project covers speaker diarization without speech content analysis. This is a common approach and is covered in many publications [3-6]. The nature of the problem that is dealt with in this thesis is that there is no prior information about an input audio file, apart from number of speakers that are present in a recording.

## Chapter 2: Tools and Technologies Overview

In this chapter technologies that are used in project are described and it's choices explained.

### **JAVA**

Java is an object oriented programming language, which syntax is derived from early programming languages C and C++. The features that make java one of the most commonly used programming languages in client – server applications are that its high level (introduction of garbage collector), multiple free of charge programming frameworks and it's intention to fulfill WORA<sup>1</sup> slogan.

Source code written in Java is compiled to classes that are run on Java Virtual Machine. JVM has multiple implementations and can be run on multiple architectures, making software written in Java accessible and portable.

### **Spring Framework<sup>2</sup>**

Spring framework is an open source framework written in Java. Spring is published under Apache license making it free of charge for both commercial and non-commercial use. It is reliable and used widely in many projects, because it greatly reduces development time and complexity.

Main feature behind Spring framework is Inversion of Control. IoC is a technique in an object oriented programming where objects used in application are assembled and coupled during application run time. In Spring Framework till version 3.0 Inversion of Control has been done by constructing XML descriptor with object properties and dependencies. Since version 3.0 objects and it's dependencies can be also constructed using annotations.

### **Apache Maven<sup>3</sup>**

Apache Maven is a project management tool. Before it has been introduced, when project used a library, the developer had to download the jar from the project site and place it in the correct project directory. Apache Maven simplifies that problem, resolving the dependencies for the developer. Developer has to only specify what library to use and in which version it is using special notation. The project structure and it's component is configured in an XML file called POM (project object model). Using Maven tool the project can be build, tested and deployed, basing on that configuration. The functions that are offered by the maven tool are enclosed in the project life-cycle.

The project is divided into modules that correspond to project's dependencies. Each application that is developed on a separate Tomcat server instance is a maven module, that share required dependencies with other modules. Using maven simplifies development, maintenance and updates of the project.

---

1 Write Once Run Anywhere

2 <http://www.springsource.org/>

3 <http://maven.apache.org/>

## **Apache HTTP Server<sup>4</sup>**

Apache HTTP Server is the most popular web server that offers great performance and multiple features. It supports common script programming languages like Ruby, Python and Pearl. It offers many modules that introduce more complex features like URL rewriting, authentication, SSL support proxy and others.

In the project the Apache HTTP Server has been used to share the files that are result of request processing. Rewriting module has been used to make URL's leading to audio files more intuitive for users. Configuration also included registering a virtual host under the sub domain <http://audio.whowhen.com>

## **Apache Tomcat<sup>5</sup>**

Apache Tomcat is the container for web applications developed in Java Servlet technology. The server deploys applications that are packed up by Maven into WAR (web archive) files. Apache Tomcat does not support EAR's (enterprise archive) but offers many desired functions like forming a cluster, that is capable of sharing object states between Tomcat instances. Tomcat cluster can share session state and balance the load of HTTP requests.

Both back-end and front-end modules that are implemented as part of the project thesis are to be deployed on separate Tomcat instances. During the development and early stages of the project, all applications can be run on the same Tomcat instance, without application distribution, but it is not recommended because developing multiple applications on one Tomcat instance is error-prone and whole application can crash with crash of the JVM.

## **HAproxy<sup>6</sup>**

*HAproxy* is an open-source application that offers reliable and fast load-balancing functions working as a proxy-server. It offers multiple algorithms for balancing the load across servers that can be configured accordingly to the problem. Configuring *HAproxy* server increases application's availability.

In the project *HAproxy* server is used to balance the load of the incoming front-end HTTP requests across the configured servers and to provide access to the split audio files. *HAproxy* sends HTTP HEAD requests to the URL that is requested by the incoming request to check what is the response from the server. Doing that it is aware of what URL's are served by which server instances. It then selects a proper server using the load-balancing algorithm and sets the cookie to the request that corresponds to the server name. That functionality is used in the project to detect on which server the split audio-files are kept in order to forward request accordingly.

---

4 <http://httpd.apache.org/>

5 <http://tomcat.apache.org/>

6 <http://haproxy.1wt.eu/>

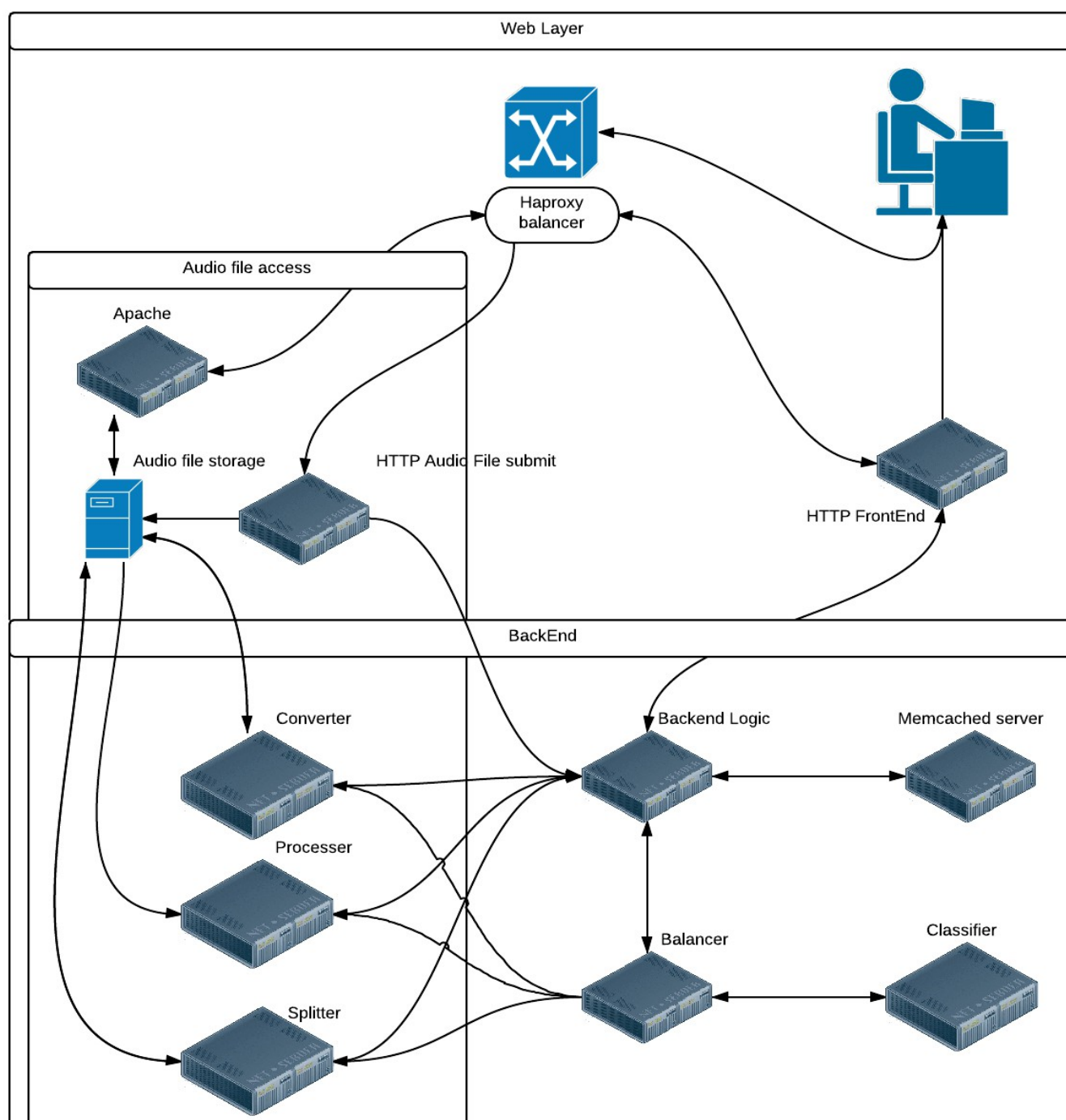


## Chapter 3: Architecture

In this chapter system architecture is described as well as the request flow in an application.

### System architecture

The web application is divided to maven submodules. That division allows distribution of web application to multiple Tomcat container instances. The balancer module, that is aware of all applications running on servers routes requests randomly to servers, providing equal load balance. Picture below displays components of web application.



*Distributed speaker diarization system  
Piotr Sukiennik*

## ***Request processing flow***

When client submits file to an application, the post request is routed to `datasubmit` module that saves file on a hard drive and client is redirected to the request processing page, where user can monitor the process.

Submitted file is routed by a *HAproxy* balancer to a server that further sends request to backend logic module for further request processing.

Backend logic module sends further request basing on the tasks completion notifications received from other modules. It also records the progress of request processing and saves that information and received request completion results on a *memcached* server. Further communication between modules is asynchronous.

The backend logic sends conversion requests to a balancer, that routes it to conversion module, that has direct access to submitted audio file. When conversion is completed, notification is being send to backend logic server with an information about converted file.

The backend logic sends further request through balancer to processing module, that extracts features from converted audio file that are most effective for speaker diarization. After processing has been completed, a notification with extracted data is send to backend logic server.

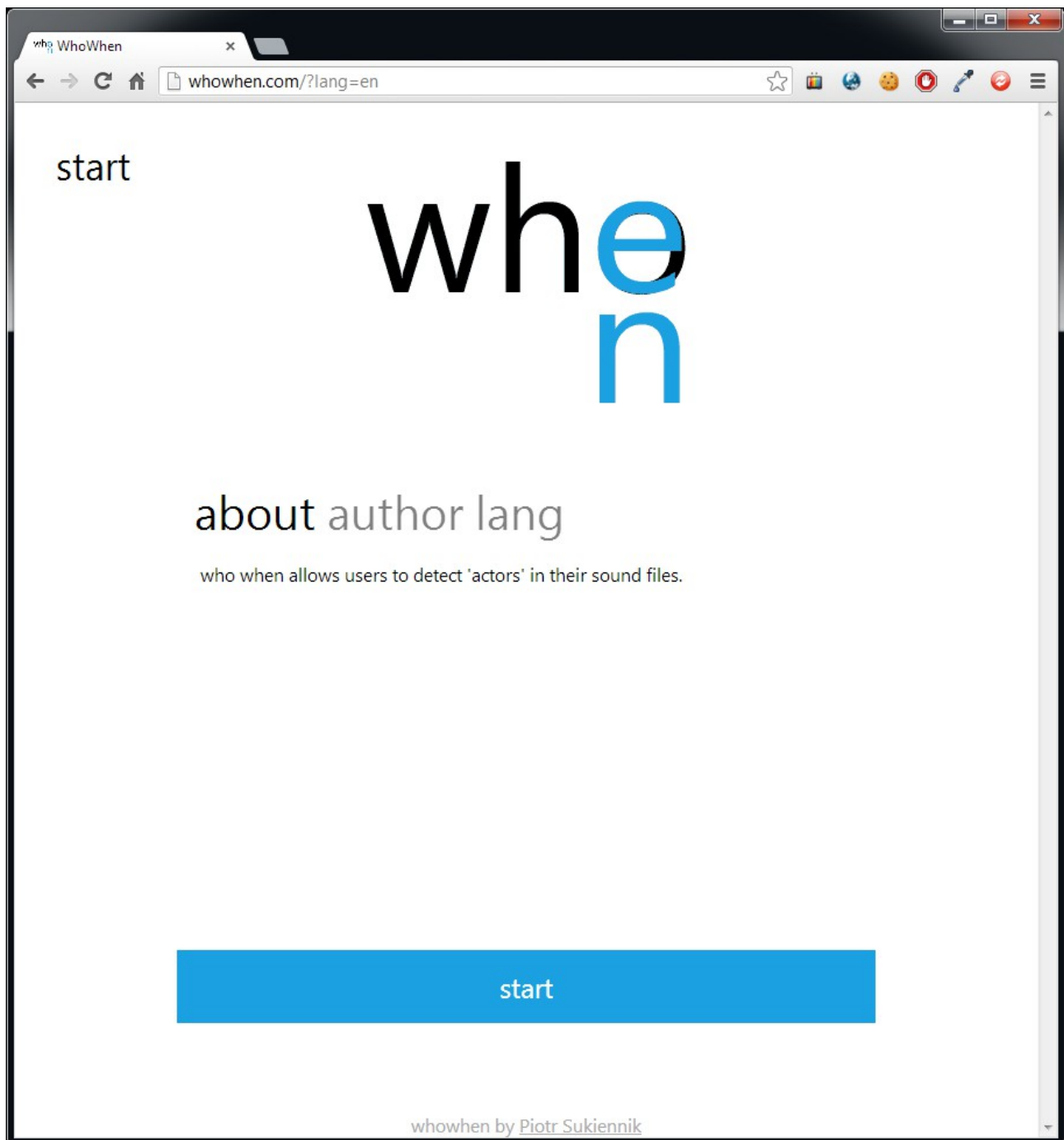
Backend logic module then sends received data from processing module to classification module, that labels single feature vectors.

The result of classification is then send back to backend logic and is routed to splitter module that splits audio file to corresponding speakers found in audio file. When audio file has been split and backend logic server receives proper notification, it sends an email notification back to a client with an URL to request page with processing output. Client can then review the output of the labeling, and can save split audio files containing found speaker speech.

The responsibilities and way the separate modules work are described in detail in further chapters.

## Chapter 4: Frontend

The frontend modules provide an interface for user to submit input files and specify number of speakers in submitted audio file and email for further completion notification. To make it easier to explain, let's assume that URL to an application is "<http://whowhen.com>".



Picture 1: the front page of the application

## ***Request flow***

The after the client enters the website, the request is routed through a *HAproxy* balancer to a frontend module. The role of an frontend module is to present an interface to submit an audio file, specify mail and number of speakers in an audio file, and to present information about submitted task progress.

Using Spring message dispatcher all text that is shown to user are kept in properties files. Therefore introducing support to a new language is an easy tasks, and requires only translation of one file, renaming it to represent the corresponding language suffix and placing it in specific directory.

## ***Technologies***

### **Spring MVC<sup>7</sup>**

Spring MVC is an extension of Core Spring Framework.

Request is being routed through an internal Spring dispatcher servlet, to an appropriate handler that is bound to a specified URL. Spring MVC framework makes it easy to implement new functions that are handled under URL's with parameters matching certain conditions. It also enforces good values passed as parameters to handlers, preventing malicious attempts.

The configuration of a web application is made using Java annotations, that represent meta data of handlers, that are plain java functions. The handlers are functions of objects called Controllers that can also be bound to URL prefixes or suffixes. Controllers objects are created by spring annotation configuration that scans specified packages for classes that are described with *@Controller* annotation, and registers handlers of those objects to URLs in dispatcher servlet.

Spring MVC is one of the most commonly used MVC frameworks in business environment. There are many Java MVC frameworks that represent different functions. Spring MVC has been chosen because it offers unique options that improve robustness and productivity.

### **jQuery<sup>8</sup>**

Jquery is a lightweight JavaScript programming library that provides set of tools making manipulation of HTML DOM elements easier. JQuery provides multiple functions for dynamic changes of page content and styles, animation and sending AJAX requests.

In the project jQuery is used for animations, scrolling, changing content of DOM elements and sending AJAX<sup>9</sup> requests. The effect of sliding tabs is obtained by the *jqMetro*<sup>10</sup> library.

---

<sup>7</sup> <http://www.springsource.org/>

<sup>8</sup> <http://jquery.com/>

<sup>9</sup> Asynchronous JavaScript and XML

<sup>10</sup> <http://manorey.net/mohblog>

## Modules

### Frontend Module

The frontend module is written in Spring MVC architecture and consists of three handlers.

1. The first handler's responsibility is to return the view of the main page, containing a Spring form with data to be collected for request processing. The user is required to specify an audio file that is located on his hard drive and matches supported audio formats. User also specifies number of speakers that occur in audio recording and email, to which a request completion notification is sent.

Handler responds to GET request to:

*http://whowhen.com*

2. The second handler's responsibility is to return view of the request processing progress page. It represents an interface, notifying user what is the state of the request that he has submitted. When client has this web page open on its web browser, jQuery AJAX requests are sent to the third handler, that returns information about request processing progress. When processing reaches state of completion, a new tab is shown for client, to review the classification results and allows him to download splitted files.

The splitted files are accessible through an apache server that is set on audio. Subdomain. Regexp patterns are used in apache configuration so the url structure to speakers are intuitive.

Files are accessible under pattern:

*http://audio.whowhen.com/{REQUEST\_ID}/clusterX.wav*

Handler responds to GET request to:

*http://whowhen.com/{REQUEST\_ID}/request*

3. The third handler's responsibility is to return the progress of the request. The information is received by AJAX request that are send from client's web browser with generated request processing page view. The handler sends progress requests to backend logic server that responds with a proper information of request state in form of serialized JSON data.

Handler responds to GET request to:

*http://whowhen.com/{REQUEST\_ID}/progress*

Where REQUEST\_ID is an id generated by the data submit module after it receives the form and X is the number of speaker found in the audio document.

## Data submit Module

Data submit module is also implemented with a use of Spring MVC framework. It's kept as a separate module because with a use of *HProxy* balancing submitted audio files are saved equally on physical hard drives, connected to backend modules.

The data submit module consists of only one handler, and receives form data posted from main web page.

The data submit module:

- Receives audio file, and information about client's email and number of speakers in submitted audio file, that client submitted using form.
- Generates an id (UUID<sup>11</sup>) that is further bound to the request.
- Stores the audio file in new folder which name is an id.
- Sends a request to the backend logic server with information about received file location, number of speakers, request id and clients email.

---

<sup>11</sup> Universally unique identifier

## Chapter 5: Backend

Backend services are web applications run in a Tomcat container. Java package of interfaces is shared across all backend modules and services are implemented using Spring Framework.

### ***Request flow***

When backend logic module application receives new request it saves Java POJO<sup>12</sup> object containing information about request progress on a *memcached* server, to be further overwritten as new step completion notifications are received.

### ***Technologies***

#### **Memcached<sup>13</sup>**

Memcached is an open source project that offers functionality of memory caching. *Memcached* servers can be distributed in a way that it's transparent for developer of an application. It is commonly used with web applications to reduce database load and store the processed results of queries, so further requests are able to receive cached object. It offers storing both Java objects and Strings. *Memcached* application could be simplified to a big hash-table that stores key-value pairs in memory. Objects stored on the server are cached for certain specified interval of maximum 30 days.

Memcached offers access to cached objects both by telnet interface and API<sup>14</sup>. In project, the *spymemcached*<sup>15</sup> Java client library is used for communication, object storage and retrieval.

It is used by many corporations, speeding up web applications that require frequent access to data stored in databases. The most known users of *Memcached* applications are: *Youtube*, *Twitter* and *Wikipedia*.

---

12 Plain Old Java Object

13 <http://memcached.org>

14 Application Programming Interface

15 <http://code.google.com/p/spymemcached/>

## Modules

### Backend Logic Module

Backend logic module serves role as a mediator between frontend and backend modules. It's implemented in spring framework technology and implements shared service, making it accessible to other modules.

It's functions are:

Responding to progress requests. Application responds to request sending POJO object that is stored on a *Memcached* server. Object stores information about current progress of request processing. That object is overwritten each time backend logic server gets notification about completion of module request.

Sending request to backend application through a balancer application. Backend logic module responds to requests and notifications received both from frontend modules and backend modules. Once request completion notification is received from backend module, backend logic module saves that response on the *Memcached* server and composes next logic step request based on communication history cached on *Memcached* server and sends request to appropriate module through balancer module. Requests and responses are cached for one day on *Memcached* server.

### Balancer Module

Balancer module serves as a mediator, balancing the load of the backend requests. It is aware of all applications that are running across the backend servers and backend logic server uses it to send requests to other servers.

The application servers designed to serve incoming request are chosen randomly from all possible application instances, what guarantees equal load distribution in an environment of multiple concurrent requests.

The backend balancer application is a web application developed in Java and offers implementation of functionality of all the backend services. The implementation is simply redirecting the request to the handler that handles logic.

### Converting Module

Converting module is used to convert uploaded files to format acceptable for further processing and feature extraction. The implementation covers thread scheduler and request handler that submits new tasks (threads) to the scheduler. Scheduler is set up to accept 4 running threads at once.

JAVE is a conversion library implemented the Java that supports converting multiple audio file formats to WAV<sup>16</sup>. JAVE simply wraps FFmpeg program and supports a Java interface for purpose of managing the communication with running process and an application.

The conversion is homogenous for all incoming formats. The output of a conversion is an audio file of 16 bit sample size and 8000 hz sampling rate. It is quality that is adequate for human speech and further allows recognition of speech recorded on a telephone line, what would be a common usage for developed application.

---

<sup>16</sup> Wave form audio format



## **Technologies Used**

### **FFmpeg<sup>17</sup>**

FFMPEG is a free software providing tools for conversion and editing multimedia files. FFMPEG supports conversion of multiple audio file types, making it flexible for use in a web application, not constraining user input audio type.

In the project modified version of JAVE<sup>18</sup> is used. JAVE wraps FFMPEG, providing Java interface for conversion of multimedia files. It communicates with FFMPEG via it's standard input, running separate process for conversion.

### **Processing Module**

Processing module is a Java application that provides feature extraction. The implementation covers a thread scheduler that accepts 4 concurrent running threads and request handler that builds a processing thread and submits it to the scheduler.

Running feature extraction thread firstly extracts the short time power spectrum information from the audio file using MFCC method. Such obtained vectors are then used to calculate their arithmetic mean values. Five feature vectors are used to calculate the mean vector, that decreases the output vectors count. Then the obtained vectors are used to calculate the delta cepstrum information. That process doubles feature vectors length.

Introducing data about delta cepstrum led to major increase in accuracy of the detected segments in audio document by the clustering algorithms.

---

<sup>17</sup> <http://ffmpeg.org/>

<sup>18</sup> <http://www.sauronsoftware.it/projects/jave/>

## **Tools used**

### **MFCC**

Mel Frequency Cepstral Coefficients is a method of extracting the parametrized information from the audio signal. The transformation is being done firstly converting the audio signal using FFT<sup>19</sup> to the time-frequency domain [7-9]. Such obtained data then is put through the filter banks that extract the most speech-dependent features from the energy spectrum, mapping it onto Mel scale. Lastly the output is transformed using DCT<sup>20</sup>, that discards the higher coefficients of a signal representation, compressing the information in output feature vector.

Mapping the energy spectrum of an input audio signal onto logarithmic scale approximates the way human ears perceive sound. Such operation is performed to extract more speech dependent features from the audio signal, leading to better results in classification [10].

In the project MFCC features are extracted from the audio document using the Comirva<sup>21</sup> library.

### **Speaker Diarization**

Many publications cover using MFCC in speaker diarization and discuss its parametrization. It is the most common choice in both voice recognition and speaker validation. Many of those configurations have been tested in project leading to the final result of that, giving most reliable results, that confirm results obtained by the researchers [11,12].

### **MFCC parameters**

Parametrization of the MFCC was one of the biggest problems during the development of the application. The settings include boundary frequencies for which the result will be obtained and Mel-filters will be placed, number of filters in the filter bank and number of the coefficients. Many papers discuss different values for the parameters of the algorithms for speaker authentication and segmentation.

During the development, the results of the segmentation were most accurate using following parameters:

Number of the coefficients: 12

Minimum frequency: 300 Hz

Maximum frequency: 3400 Hz

Number of filters: 24

---

<sup>19</sup> Fast Fourier Transform

<sup>20</sup> Discrete Cosine Transform

<sup>21</sup> <http://www.cp.jku.at/people/schedl/Research/Development/CoMIRVA/webpage/CoMIRVA.html>

## Classification Module

Once features are extracted, the list of double arrays is sent to the classification application. Goal of the classification application is to label each double array accordingly to data it represents.

As other modules, the application is implemented in Java and composes of thread scheduler allowing 4 threads to be running concurrently and a request handler that builds a thread which role is to label the obtained data. Such thread is then submitted to the scheduler.

Firstly each vector is put through the classifier that recognizes vectors that correspond to periods of silence in an audio signal. This classifier simply gathers absolute sums of the vector components and checks if the output sum exceeds the required threshold needed for the instance to be classified as a one representing moment of speech. The threshold has been experimentally set to the value of 30. Summing vector components that are result of MFCC processing allows to discard information that is not important for speech detection, and classifies it as silence. The moments of silence are also sounds that got filtered by log-scale Mel filter bank.

When the data is labeled accordingly as silence and not silence, the vectors that got through filtering classifier are marked as those that contain speech. Such vectors are then gathered and labeled on another level, looking for occurrences of different speakers.

To recognize the speakers in the extracted features vectors representing speech, the data needs to be properly clustered. That purpose is served by kNN algorithm that clusters the data basing on the euclidean distance of each instances to the cluster center, that is mean of all vectors that are classified as that instance.

After clustering is performed by the kNN algorithm, the labeled data is smoothed, so that when there are multiple classifications of one speaker in a group and one mismatch between those, it is assumed to be an error and fixed.

## Clustering

Clustering is a problem of unsupervised learning and it's task is to find similarities or structure in a set of unlabeled data that could lead to grouping it in clusters that represent some level of similarity. Clustering has multiple applications in image compression, web mining, marketing and many other tasks where large sets of unlabeled data is handled.

Due to the fact that information about the number of speakers in an audio document that is being uploaded to the web server is not known in every situation, the clustering algorithm that could decide on the most fit number of clusters was desired. EM and xMeans algorithms were the first choice because those methods offer automatic detection of most fit number of clusters (speaker segments).

## EM

Expectation Maximization is a clustering algorithm that assigns probability distribution to every instance using Gaussian Mixture Models [13] method. The advantage of EM algorithm is that it can decide on the number of clusters to generate with use of cross validation. The GMM EM algorithm is one of the most popular solutions to speaker segmentation problem and is covered in multiple publications [1-7]. However it occurred to be a poor choice for the web application in a distributed environment. The clusterer has been build using only half of the data obtained from the input audio

file and this step took about 1100 msec per one minute of the audio document. That occurred to be about 5 time more than time required by kMeans algorithm. Furthermore the automatic cluster detection was not successful, leading to lack of accuracy of found clusters. There has been noticed a tendency to cluster similar sounds into multiple small clusters and one containing rest of data. That has been checked with multiple configuration of extracted features, leading to similar unsatisfying results. High computation cost of EM algorithm and small accuracy led to abandonment of this clustering method.

## **kMeans**

The usage of kNN algorithm puts a constraint and requires user of the application to know how many speakers are present in the audio recording he submits to be processed by the application.

Such apriori information is used by an kMeans clustering method to split an  $n$  observation data-set into  $k$  clusters, for each data-point selecting the cluster with the closest mean. To calculate the distance between the data-point and the cluster mean, the distance function (i.e. Euclidean Distance) is used.

Kmeans algorithm showed best efficiency and performance, when building clusterer and using it to classify the data occurred in an average of about 205 msec per minute of an audio document.

The poor accuracy of algorithms providing automatic cluster count prediction led to using kMeans algorithm. The prior information about an uploaded audio document about the number of speakers present is a drawback but results of the speaker segmentation are more reliable and satisfying.

## **xMeans**

xMeans is an extended version of kMeans algorithm with a part that improves structure of found clusters by splitting it and comparing the BIC<sup>22</sup>-values of the new clusters. BIC is used to found an most probable number of clusters, at the same time avoiding the problem of over-fitting. In case of xMeans it was common that in set of instances of features extracted from audio document with one speaker present, many clusters were found.

## ***Technologies Used***

### **Weka**

Weka is a Java library offering multiple implementations of most used machine learning software. It offers easy to use GUI and data visualization, configuration of tasks in style of makefile, algorithms for pre-processing and modeling and many more. In the project Weka 3 Java library is used.

---

22 Bayesian information criterion

## Splitting Module

The splitting module is the last backend application that receives output from classification application in form of labeled vector indexes. Basing on that information, overall number of vectors and the duration of an audio file it is estimated how long the one classification lasts in the audio file and where it is placed.

The intervals of group classifications are then grouped and saved on a hard drive for each level of classification (silence/ not silence, speakers diarization) in form of an audio file. Once files are saved, it is possible for user to access them using configured Apache Server.

After split files are saved on a server instance, the notification is sent to the Backend Logic module, that concludes request processing.

## Chapter 6: Future updates

The system's architecture has been designed in such way that implementing new functions of the application is easy and does not require a lot of modification of existing code.

The most obvious and easy to upgrade module would be classification module. Classifier trained to classify data instances corresponding to speech, music, speech and music and other could be introduced between silence detection and speaker segmentation. Speaker segmentation could then be performed only on the data that corresponds to the speech-fragments.

Furthermore, extracting more data from the input audio documents containing speech, like phonemes, accent-analysis and textual content, could lead to more accurate segmentation. With an expansion of the feature vector it is possible that different clustering algorithm could be applied that supports automatic cluster count detection, what would further improve usability.

Another improvement could involve the request processing mechanism to be more failure proof. If for example the converting module fails to convert the input file due to the internal, application-independent problem, the processing is stopped and user would have to reinitialize procedure. Storing request identifiers with the form data submitted with a request and it's request progress in a database would allow to introduce mechanism of reinitialization of the backend logic steps. The list of the currently processed request could be reiterated by a scheduled module and if request processing takes more time than it should, a request processing could be restarted from the last step.

## Chapter 7: Conclusion

The distribution of the application to the task-dependent modules allowed system to be more flexible in terms of updates and maintainability. The replacement of one application module requires the pause of functions of this module, not whole application. That is also important in case of code management and current version control. The combination of Maven module management and Spring remoting support made development of a distributed application an easily maintainable task.

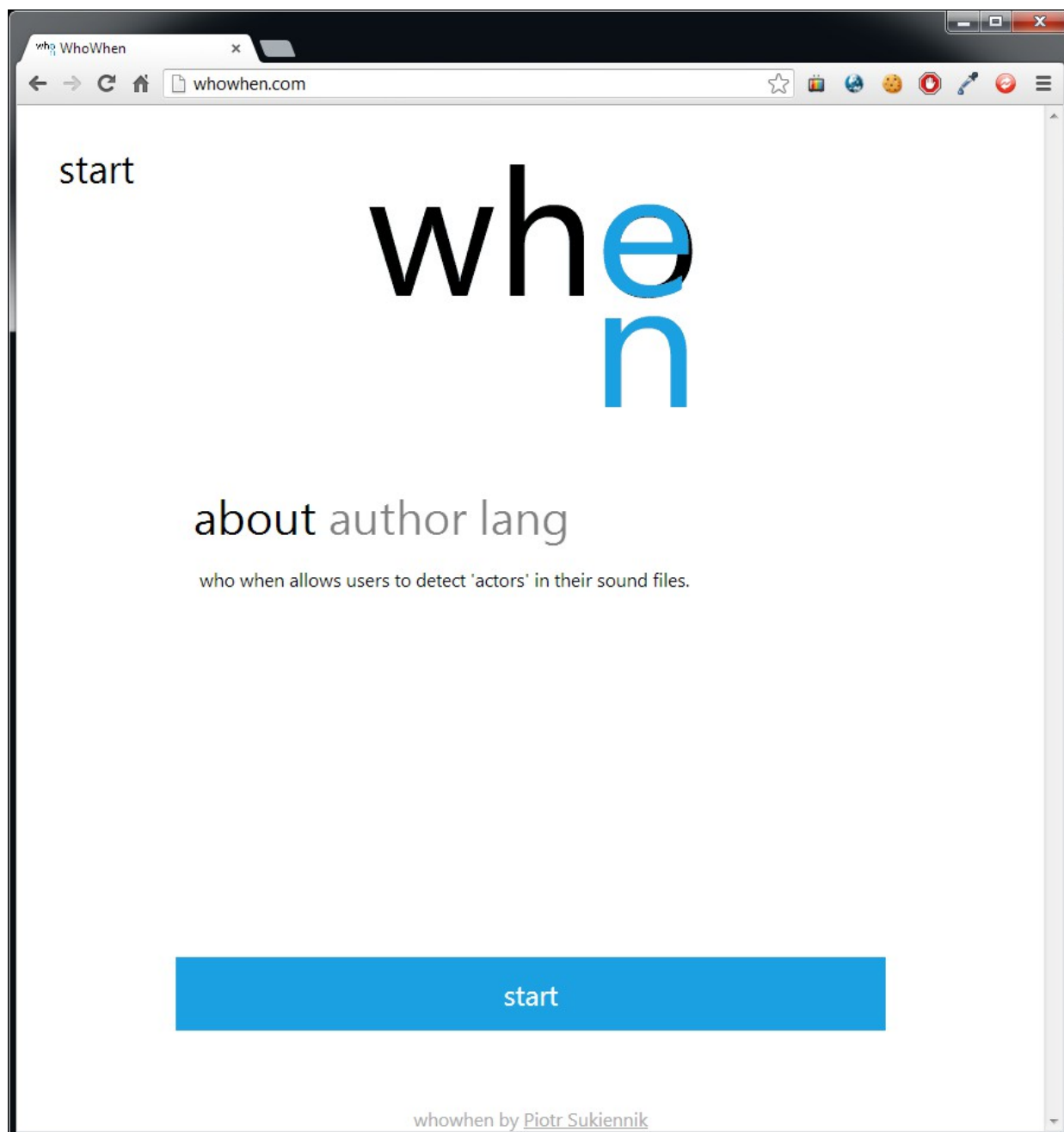
The feature extraction from the audio documents using MFCC and successful results obtained from that method proved that it is a reliable technique for information extraction from speech fragments. The major increase of accuracy obtained from using additional data obtained with delta cepstral coefficients, showed importance of this step.

This thesis showed that speaker diarization can be performed using simple clustering algorithms like kMeans that are computationally efficient, making them good choice for use in web application environment. It has been showed, that using GMM EM clustering algorithm did not lead to successful results, and was computationally demanding, that excluded it for usage in a system of multiple concurrent requests. That extraction of more complex and high-level features may lead to the increase of accuracy of speaker segmentation.

The outcome of the project showed that creating an speaker diarization system that is accessible for everyone through web interface is possible. Speaker diarization is a still open problem in computer science and certainly new research will provide appliances to new tools and more accurate solutions will be implemented.

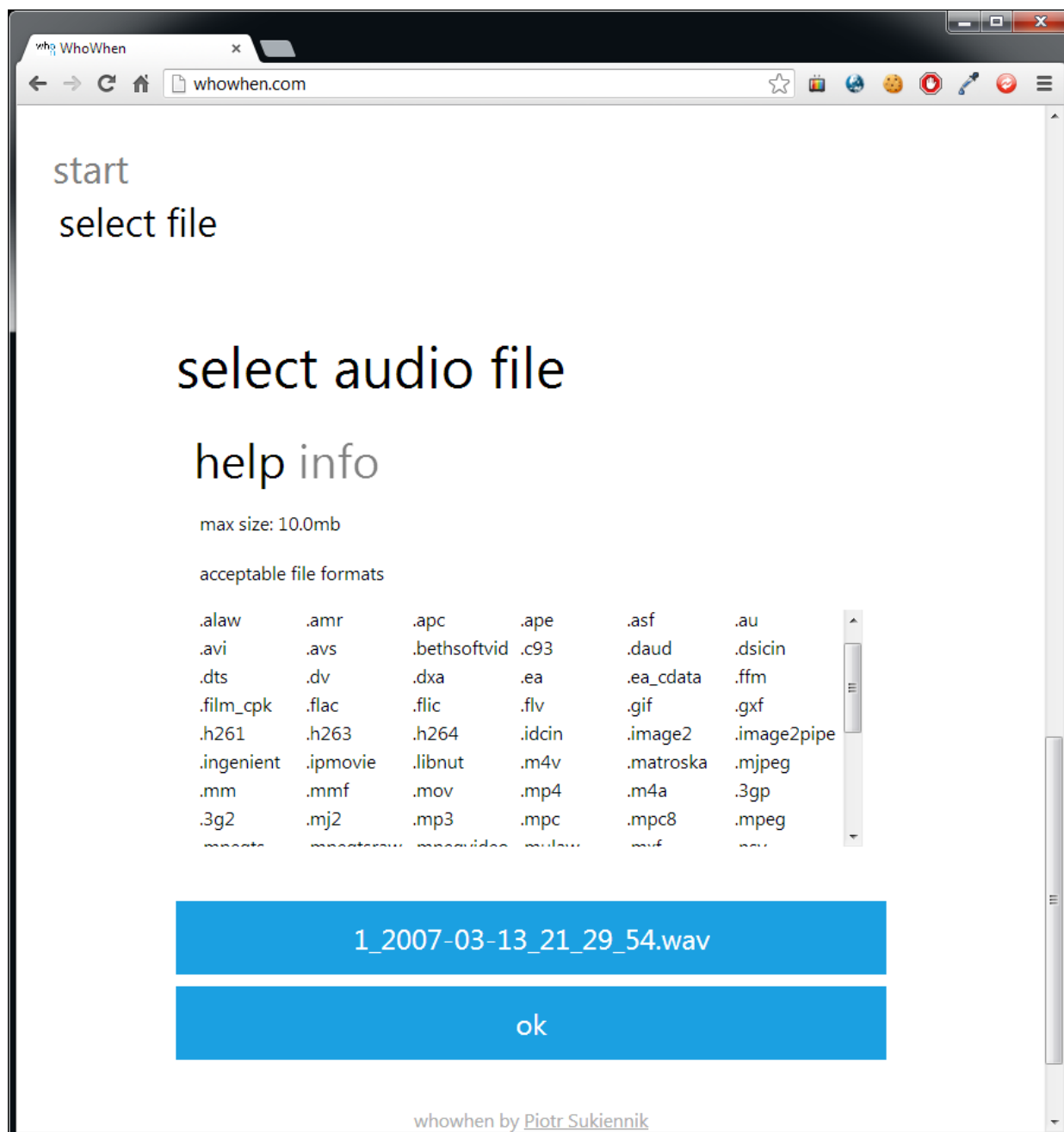
## Example Interface Interaction

1. User enters the main, index page.

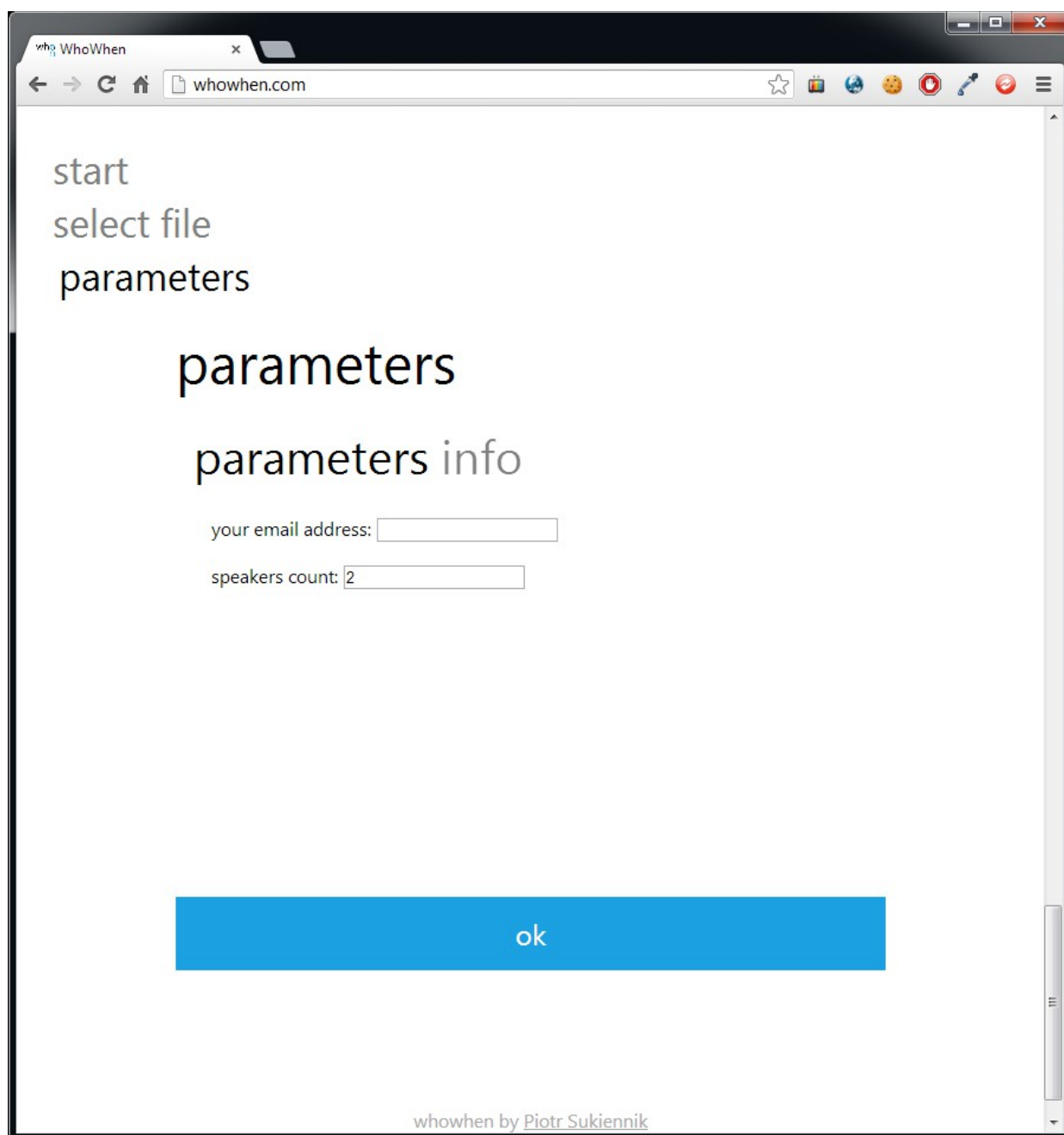




2. User selects the audio-file to be processed by the application. Input form allows only formats that are accessible for conversion.



3. User inputs his email and number of speakers in the audio-file.



The screenshot shows a web browser window with the address bar displaying 'whowhen.com'. The page content includes a sidebar with links: 'start', 'select file', and 'parameters'. The main area is titled 'parameters' and 'parameters info'. It contains two input fields: 'your email address:' followed by an empty text box, and 'speakers count:' followed by a text box containing the number '2'. A large blue button labeled 'ok' is positioned below the input fields. At the bottom of the page, it says 'whowhen by Piotr Sukiennik'.

start  
select file  
parameters

## parameters

### parameters info

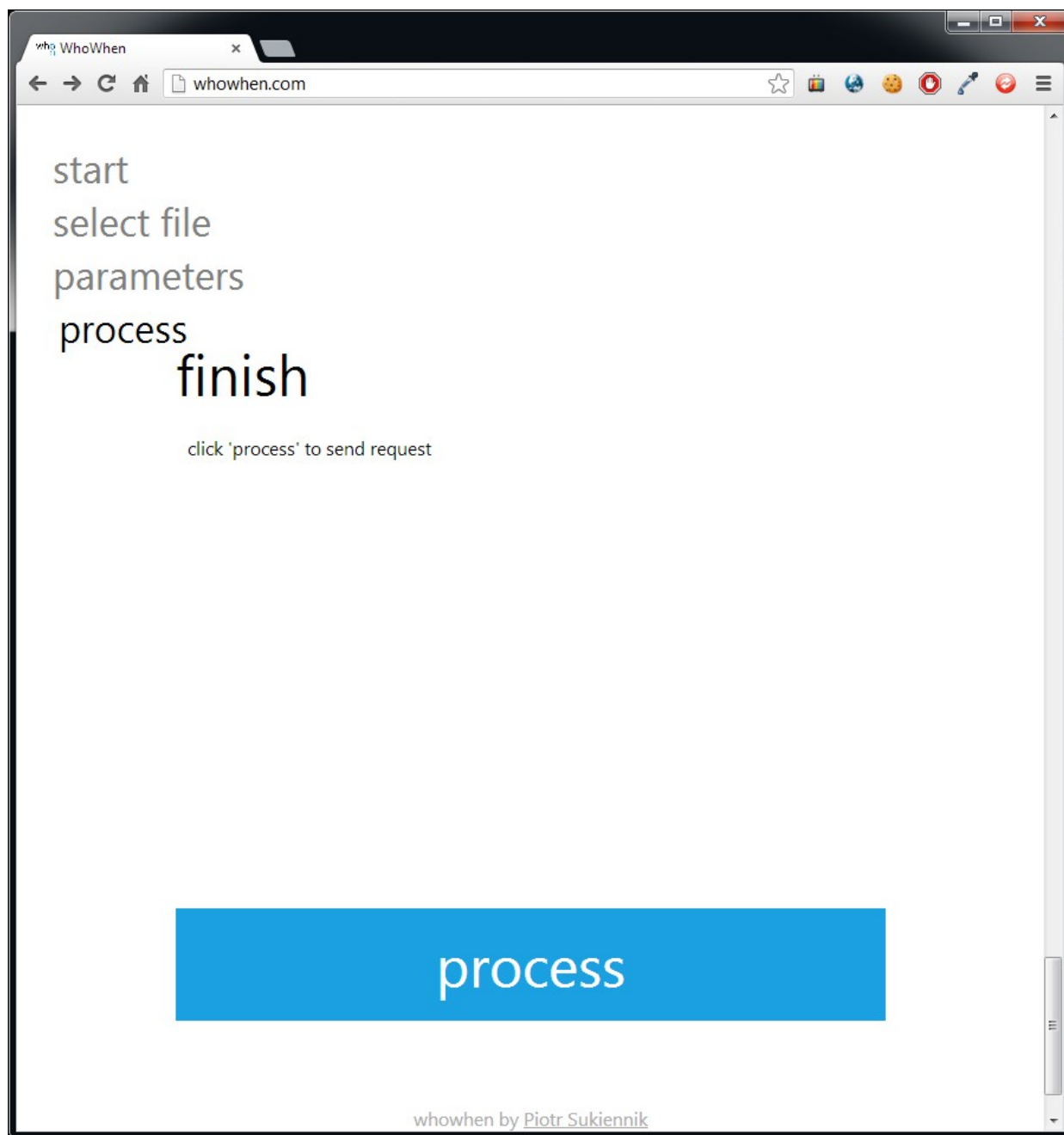
your email address:

speakers count: 2

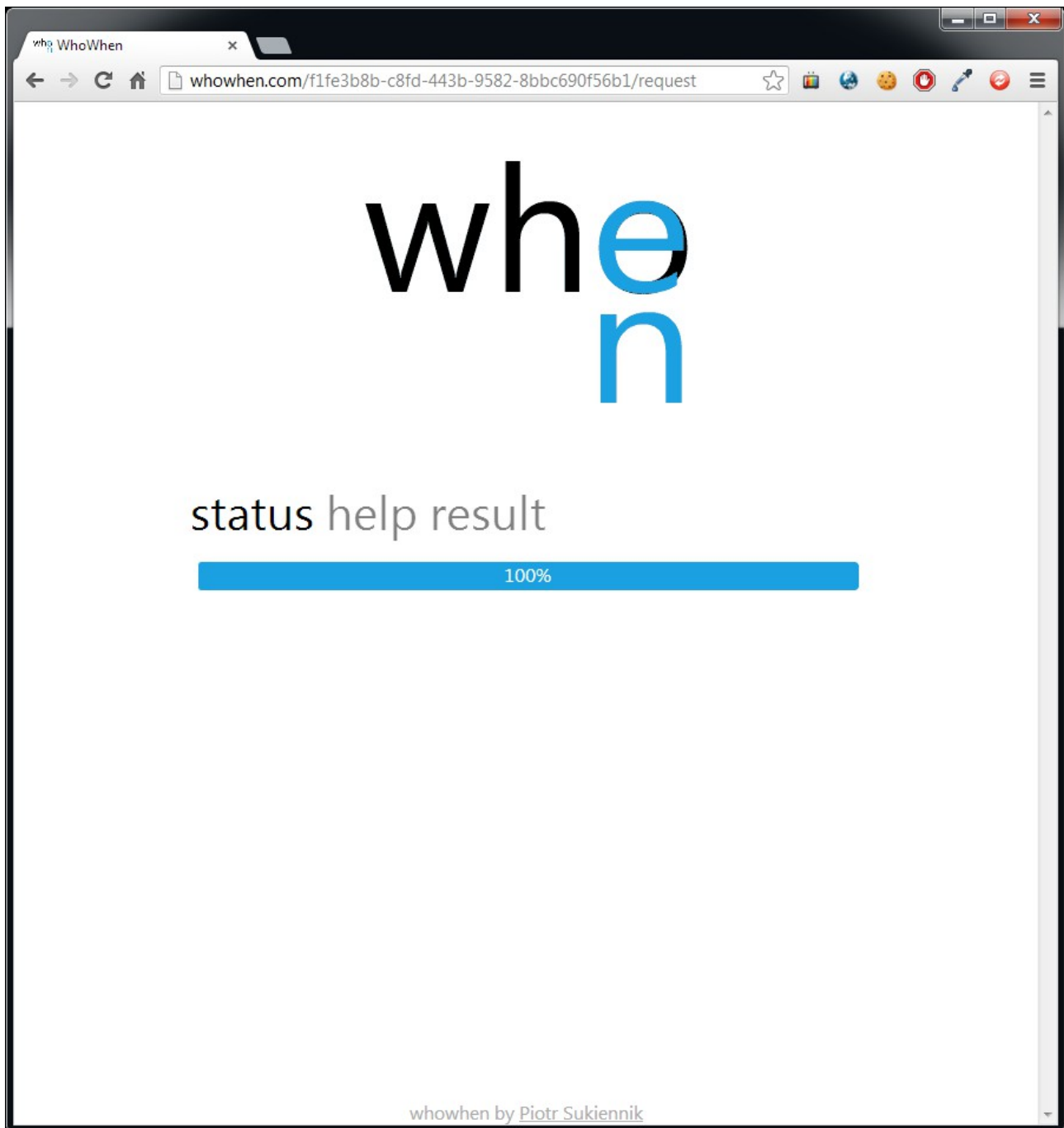
ok

whowhen by Piotr Sukiennik

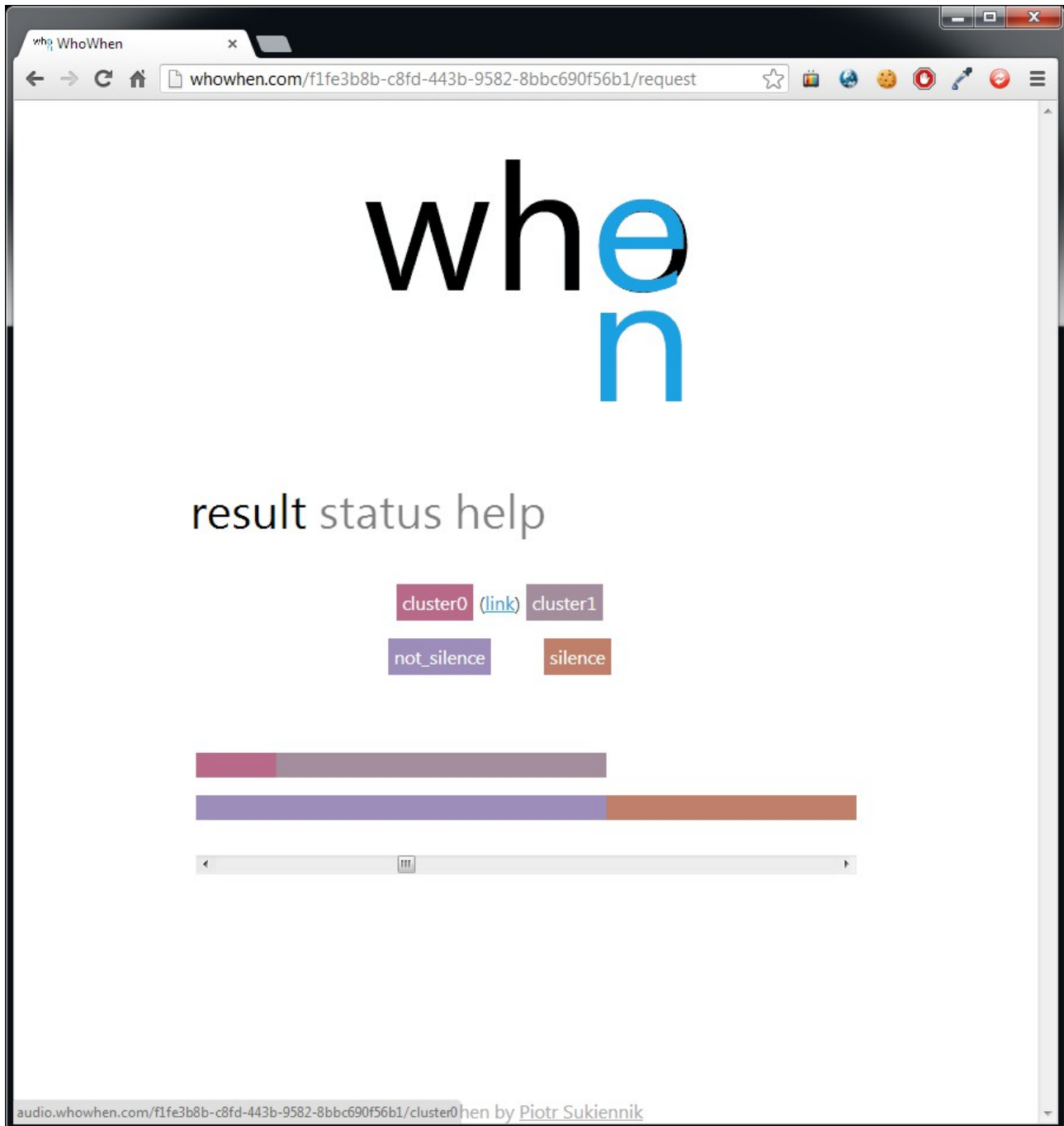
4. User is taken to the last step of the form.



5. Once user clicks process, it is redirected to the request processing page, where he can monitor the request. Every update of the progress is shown on the progress bar (depending on the current step of the request processing).



6. Once the results has been obtained, it can be seen on the “results” tab. The user can preview on a simple time-line how audio fragments are classified in the split audio-files. User can access audio files using links that are revealed on hover over the segment labels.



## References

- [1] Anguera, X; Bozonnet, Simon; Evans, Nicholas W D; Fredouille, Corinne; Friedland, O; Vinyals, O "Speaker diarization: A review of recent research", *IEEE Transactions On Audio, Speech, and Language Processing*, Volume 20, N°2, February 2012
- [2] F.Bimbot, J. Bonastre, C. Fredouille, G. Gravier, I. Magrin-Chagnolleau, S. Meignier, T. Merlin, J. Ortega-García, D. Petrovska-Delacrétaz, and D. A. Reynolds "A tutorial on text-independent speaker verification." *EURASIP J. Appl. Signal Process.*, pages 430-451, 2004
- [3] A. Reynolds, T. F. Quatieri and R. B. Dunn "Speaker verification using adapted Gaussian mixture models", *Digital Signal Process.*, vol. 10, pages 19 -41, 2000
- [4] A. Reynolds "Speaker identification and verification using gaussian mixture speaker models", *Speech Communication*, vol. 17, pages 91 -108, 1995
- [5] A. Reynolds, R. C. Rose "Robust Text-Independent Speaker Identification Using Gaussian Mixture Speaker Models", *IEEE Transactions On Speech And Audio Processing*, vol. 3, 1995
- [6] H. Ning, M. Liu, H. Tang, T. Huang "A Spectral Clustering Approach to Speaker Diarization"
- [7] J. W. Picone, "Signal Modeling Techniques in Speech Recognition", *Proceedings of the IEEE*, vol. 81, No. 9, pages 1215-1247, 1993
- [8] W. Han "An efficient MFCC extraction method in speech recognition", *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium*, 2006
- [9] V. Tiwari "MFCC and its applications in speaker recognition", *International Journal on Emerging Technologies*, pages 19-22, 2010
- [10] Wong, E "Comparison of linear prediction cepstrum coefficients and mel-frequency cepstrum coefficients for language identification" *Intelligent Multimedia, Video and Speech Processing*, 2001. *Proceedings of 2001 International Symposium*, pages 95-98, 2001
- [11] Md. R. Hasan, M. Jamil, Md. G. Rabbani Md. S. Rahman "Speaker identification using mel frequency cepstral coefficients", *3rd International Conference on Electrical & Computer Engineering ICECE 2004*, pages 28-30, 2004
- [12] T. Ganchev, N. Fakotakis, G. Kokkinakis "Comparative Evaluation of Various MFCC Implementations on the Speaker Verification Task", in *Proc. of the SPECOM-2005*, pages 191-194, 2005
- [13] E. Rasmussen "The Infinite Gaussian Mixture Model", in *Advances in Neural Information Processing System*, pages 554-560