MIMIC HUMAN SPEECH IN BAHASA INDONESIA USING SPEECH RECOGNITION AND SPEECH SYNTHESIS

By

Valens Prabagita Ivan Susilo

A Thesis
Submitted to the Faculty of Computing
President University
in Partial Fulfilment of the Requirements
for the Degree of Bachelor of Science
in Information Technology

Cikarang, Bekasi, Indonesia

October 2018

MIMIC HUMAN SPEECH IN BAHASA INDONESIA USING

SPEECH RECOGNITION AND SPEECH SYNTHESIS

By

Valens Prabagita Ivan Susilo

Approved:

Dr. Tjong Wan Sen, S.T., M.T. Thesis Advisor	Drs. Nur Hadisukmana, M.Sc. Program Head of Information Technology

Ir. Rila Mandala, M.Eng., Ph.D.
Dean of Faculty of Computing

ABSTRACT

Everyday people use Speech recognition and speech synthesis unconsciously. The technologies help them with their activities. With each technology can produce any kinds software related to speech. Combine both of technologies can produce many more. One of the combinations is mimic human speech. This research will discuss about Convolutional Neural Network, Speech Recognition, and Speech Synthesis. The purpose of this research is to develop application to collect, train, and mimic speech in Bahasa Indonesia. User can participate record their speech. The collected speech will be train to be used in the application to recognize the speech. After the collected speech is trained, User can mimic their speech by identify or recognize the speech and generate or synthesis the speech. The application to collect and mimic speech develops in website application.

Keywords: Convolutional Neural Network, Speech Recognition, Speech Synthesis

ACKNOWLEDGMENTS

The author gratefully acknowledges the amazing supports, guidance and advices from:

1. Bebe, Annisa Feryannie Widya.
2. My beloved family, Ayah, Ibu, Mas Risky and Dek Lia.
3. B35TFR13NDZ0NE, Nisa, Marsel, Bora, Fira, Rai, Pindy, Oscar, and Michel.
4. Sangar Team, Mas Andreas, Mba Diana, Nisa, Alfian, and Aci
5. DV 14, Vovo, Gojal, Ariyel, Rino, Willy, Azmi, Damara, Atisah, Risda, Sonya, Dini, and Arizha.
6. My cool thesis lecturer, Mr. Wan Sen.
7. My second family, Van Lith XXII.
8. And you, yes you.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

FIGURE Page

FIGURE Page

CHAPTER I

INTRODUCTION

1.1 Background

“Ok Google, play some music”. “Siri, what should I eat for lunch?”. Everyday people use their virtual assistance to boost their activities. People very like to use it because they just asked to their device and then in seconds, the wish is granted. It seems like, people are talking to the computer. The truth is, speech recognition takes big role with the help of machine learning. Google Assistance, Apple Siri, Microsoft Cortana, Amazon Alexa, and others have thousands of speech data to be analysed with the machine learning and they easily add data by collecting people speech from the assistance with permission.

If speech recognition is the process to get data by analysed speech, the opposite of speech recognition is speech synthesis, the process to produce artificial speech. Therefore, speech recognition is known as speech-to-text and speech synthesis is known as text-to-speech. “Hey Cortana, read my email” command make virtual assistance generate speech from the email text. With each technology can produce any kinds software related to speech. Combine both of technologies can produce many more. One of the combinations is mimic human speech.

1.2 Problem Statement

This research aims to develop application which can be used to collect speech data, train the collected data and mimic speech in Bahasa Indonesia. The application to collect and mimic speech develops in website application. The application can recognize the speech and generate speech from text.

1.3 Research Objective

This research sees an opportunity to implement speech recognition and speech synthesis to create a mimic speech.

1.4 Scope and Limitation

This research focuses on developing an application which will be able to:

1. Perform collecting data.
2. Perform train the collected data.
3. Perform speech recognition.
4. Perform speech synthesis.

The limitations of this application are as following:

1. There are 10 selected phonemes to be used in the application, a, i, t, na, ma, mu, di, ri, and ku.
2. Recorded speech in 1 second, with sample rate 16000 and mono sound.
3. Speech recognition data is taken from recorded speech and in human speech in Bahasa Indonesia.
4. Speech synthesis data is taken from saved speech, result from speech recognition.
5. Application is developed as website application.

6. Collecting, testing, and running the application is used in the same hardware.

1.5 Thesis Methodology

Rapid Application Development (RAD) methodology will be used in the development of this application. The RAD method, which was first developed by James Martin, is a Software Development Life Cycle method that gains its popularity in recent years due to its suitability to manage web application projects. The features of RAD were designed to overcome most of the shortcomings found in traditional waterfall model. Some of these features are: fast prototyping and capability to deal with change in requirements.

The RAD model implemented in this thesis will consists of four major phases:

1. Requirement Planning Phase

   This is the where system planning and analyses are done. System requirements are established, including the view range of the camera, the numbering schema for parking lots, and the general category of cars to be detected. Algorithms are proposed to solve the problem along with the overall outline of the program.

2. User Design Phase

   During this phase, the model of system's processes is the main focus. Models for input, output, process and user interface is built, and represented in different parts that include diagrams visualization. The system design will refer to the plan created in previous stage. The design will be continuously discussed, reviewed, and updated until the best version is found.

3. Development Phase

   This phase is where the ideas and plans are executed. The application is developed according to the predefined features standard. All the components of image processing and object detection are put together into one program to perform the work from beginning to the final output. Unit testing will also be done here. It focuses on application development, including: coding, unit integration, and testing.

4. Cut Over Phase

   In this final phase there will be some test to evaluate the program's ability to determine which area of parking lot is empty. There will be certain test cases of parking areas that will produce different images to be processed. An evaluation will be done to see how far the application is capable to detect the area. Bugs fixing will also be done in this step. Another part of cutover phase is to create installation and operating manuals to allow people operate the program in their environment.

1.6 Thesis Outline

The thesis consists of seven chapters, which are as follow:

1. Chapter I: Introduction

This chapter introduce the research background, problem, and objective. It also explains the research scope and limitation, method to achieve the objective.

2. Chapter II: Literature Study

This chapter contain the literature study that related to the research background.

3. Chapter III: System Analysis

This chapter explains the analysis of the application – both in its function and behaviour, in order to fulfil the prescribed requirements.

4. Chapter IV: System Design

This chapter explains the system design of interfaces and class diagram based on the previous chapter that will be used in the next chapter.

5. Chapter V: System Development

This chapter explains the system development of interfaces and code details on the application.

6. Chapter VI: System Testing

This chapter ensures the application system runs well by evaluating all the features, and making sure the system fulfils its function requirements.

7. Chapter VII: Conclusion and Future Work

This chapter sums up this research and also suggestion for future research work.

CHAPTER II

LITERATURE STUDY

2.1 Machine Learning

Machine learning is a form of AI that enables a system to learn from data rather than through explicit programming [1]. Others state that machine learning usually refers to the changes in systems that perform tasks associated with artificial intelligence (AI) [2].

Basically, machine learning consists of 2 words, Machine and Learning. As a noun, there are many definitions related to machine [3] and one of the definitions, machine is a computer. Meanwhile, learning can be interpreted as the activity of obtaining knowledge or knowledge obtained by study [4]. Combine both words, machine learning can be interpreted as a computer that does an activity of obtaining knowledge.

In general, there are 2 types of machine learning Supervised Learning and Unsupervised Learning.

2.1.1 Supervised Learning

In supervised machine learning, a system is trained with data that has been labelled. The labels categorise each data point into one or more groups, such as 'apples' or 'oranges'. The system learns how this data – known as training data – is structured, and uses this to predict the categories of new – or 'test' – data [5]. It can be considered the learning is guided by a teacher. One has a dataset which acts as a teacher and its role is to train the model or the machine. Once the model gets trained it can start making prediction or decision when new data is given to it [6].

It can be concluded that supervised machine learning is the learning way where the machine is guided by labelled data and it needs to learn how to classify the model data to fits the labelled data.

2.1.2 Unsupervised Learning

Unsupervised learning is learning without labels. It aims to detect the characteristics that make data points more or less similar to each other, for example by creating clusters and assigning data to these clusters [5]. Suppose images of apples, bananas and mangoes are presented to the model, so what it does, based on some patterns and relationships it creates clusters and divides the dataset into those clusters. Now if a new data is fed to the model, it adds it to one of the created clusters [6].

It can be concluded that unlike supervised learning, unsupervised learning is the learning way where the machine is the figure out data by cluster the data.

2.2 Neural Network

A neural network is an approach to machine learning in which small computational units are connected in a way that is inspired by connections in the brain [5].

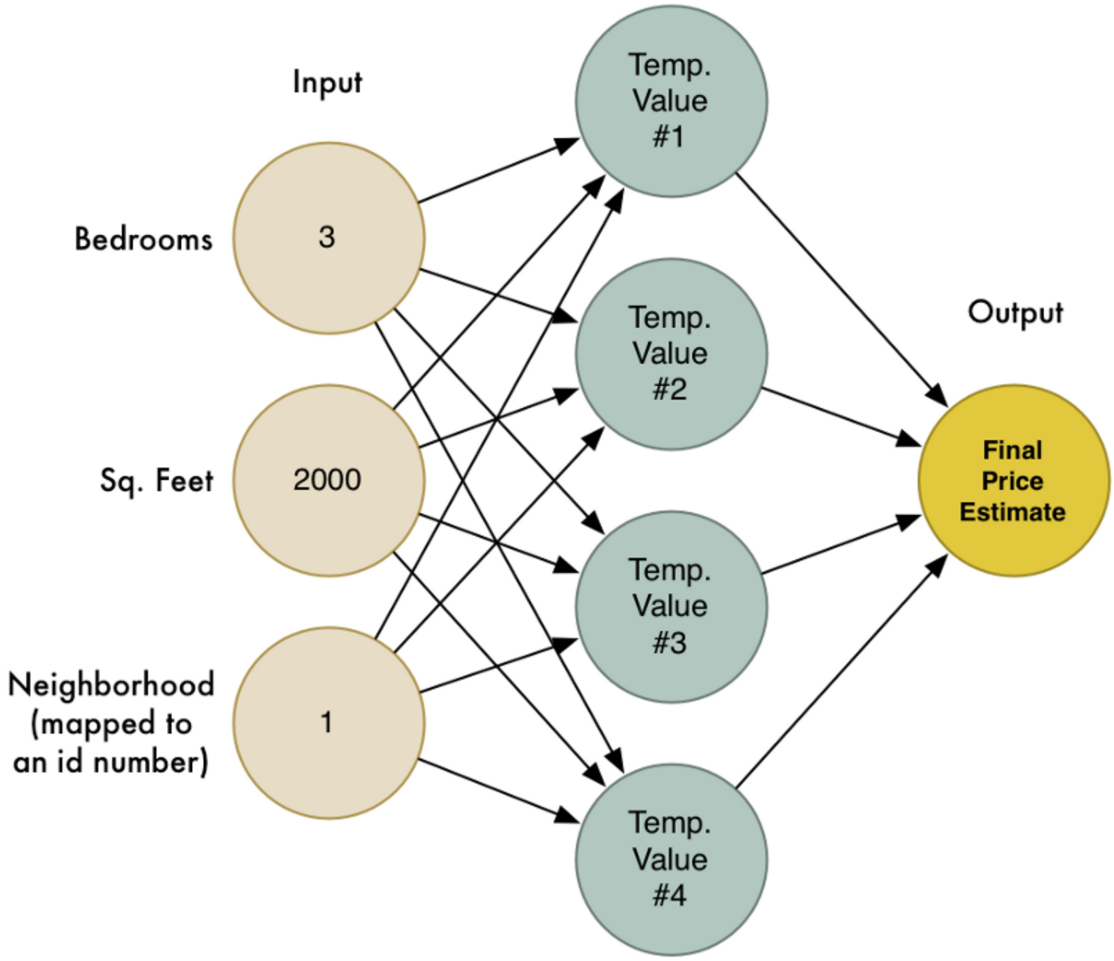


Figure 2.1 Neural network to find estimated price from specific input [7].

Every neural network model is basically a three-layered system, which are Input layer, Hidden Layer and Output Layer [8]. Input layer, where the inputs of the problem are received, hidden layers, where the relationship between the inputs & outputs are determined & represented by synaptic weights, & an output layer which emits the outputs of the problem [9].

…Example and explanation of Neural Network…

2.3 Phonemic

…Explanation of phonemic…

A phoneme is the basic unit of phonology. It is the smallest unit of sound that may cause a change of meaning within a language, but that doesn't have meaning by itself. For example, in the words' "bake" and "brake," only one phoneme has been altered, but a change in meaning has been triggered. The phoneme /r/ has no meaning on its own, but by appearing in the word it has completely changed the word's meaning [10].

In the other hand, grapheme is individual letters and groups of letters that represent single phonemes, like the "s" and the "oo" in "spoon". Understanding how letters are used to encode speech sounds in written language is crucial in learning to decode unfamiliar words [11].

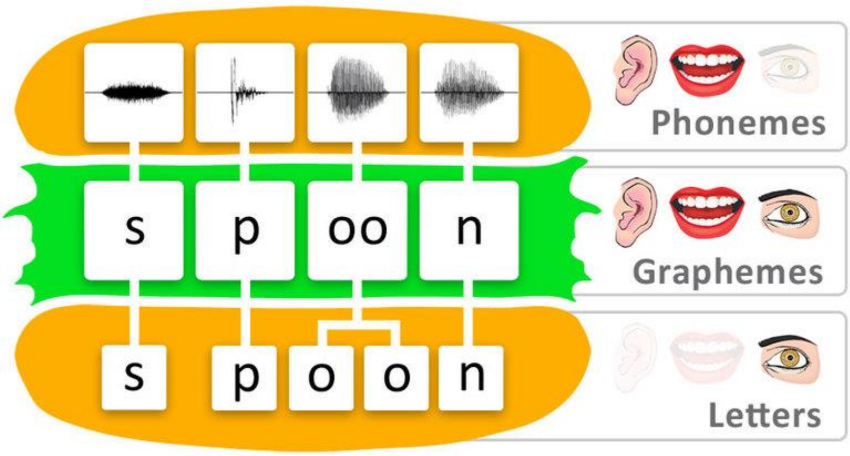


Figure 2.2 Phonemes, graphemes, and letters in the word "spoon" [11].

In Bahasa Indonesia phoneme is distributed into *vokal*, *diftong*, *konsonan*, *gugus konsonan* [12].

…Explanation of silabel…

2.4 Speech Recognition

The process of automatically recognizing spoken words of speaker based on information in speech signal is called Speech Recognition [21]. Other definition of speech recognition, also known as Automatic Speech Recognition (ASR), or computer speech recognition, is the process of converting a speech signal to a sequence of words, by means of an algorithm implemented as a computer program [22].

…Google Convolutional Neural Network for Small-footprint Keyword Spotting…

2.5 Pre-processing Speech

As sound is transmitted as waves, and computer understand numbers, it's necessary to pre-processing speech so that computer understand and can be feed as input into neural network.

…Additional explanation to convert to CNN…

2.5.1 Sound Sampling

Sound sampling is taking a reading thousands of times a second and recording a number representing the height of the sound wave at that point in time. Basically, all an uncompressed .wav audio file [23].

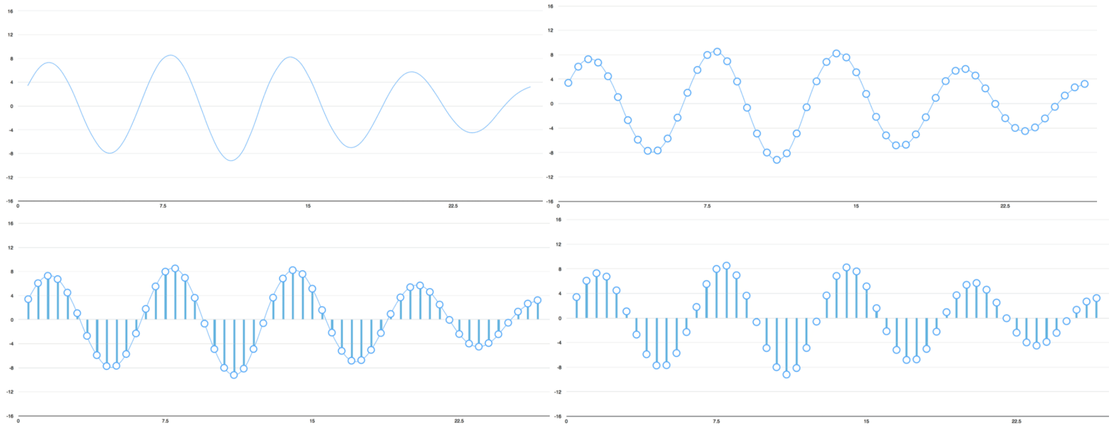


Figure 2.3 Sound sampling process [23].

```
[-1274, -1252, -1160, -986, -792, -692, -614, -429, -286, -134, -57, -41, -169, -456, -450, -541, -761, -1067, -1231, -1047, -952, -645, -489, -448
, -397, -212, 193, 114, -17, -110, 128, 261, 198, 390, 461, 772, 948, 1451, 1974, 2624, 3793, 4968, 5939, 6057, 6581, 7302, 7640, 7223, 6119, 5461,
 4820, 4353, 3611, 2740, 2004, 1349, 1178, 1085, 901, 301, -262, -499, -488, -707, -1406, -1997, -2377, -2494, -2605, -2675, -2627, -2500, -2148, -
1648, -970, -364, 13, 260, 494, 788, 1011, 938, 717, 507, 323, 324, 325, 350, 103, -113, 64, 176, 93, -249, -461, -606, -909, -1159, -1307, -1544]
```

Figure 2.4 100 samples in numbers from “Hello” sound with sample rate of 16kHz (16 samples per seconds) [23].

Nyquist sampling theorem [24] provides a prescription for the nominal sampling interval required to avoid aliasing. The sampling frequency should be at least twice the highest frequency contained in the signal. In the case, where one has f_c = 3 Hz, and so the Nyquist theorem tells that the sampling frequency, f_s, must be at least 6 Hz [25].

2.5.2 MFCC

…Explanation of MFCC…

2.6 Speech Synthesis

Speech synthesis is the artificial production of human speech [26]. The Synthesized speech can be created by concatenating pieces of recorded speech that are stored in a database [27]. From phoneme, text can be analysed by its phonemes and then with the phonemes concatenating speech can be done.

2.7 Related Work

The following are most related work to the research. Have relation to mimic speech, both speech recognition and speech synthesis.

2.7.1 Lyrebird

Lyrebird is website application, *https://lyrebird.ai*, that has 3 products: Custom Voice, Vocal Avatar, and Vocal Avatar API [28]. Custom voice is a product to create speech based on real people's speech, it can control the intonation, expression, and the emotion of the speech. Vocal avatar is a product to create own digital speech by read some English sentences, and then generate any sentences with own digital speech. Vocal avatar API is a product to provide API to use user's own vocal avatar.

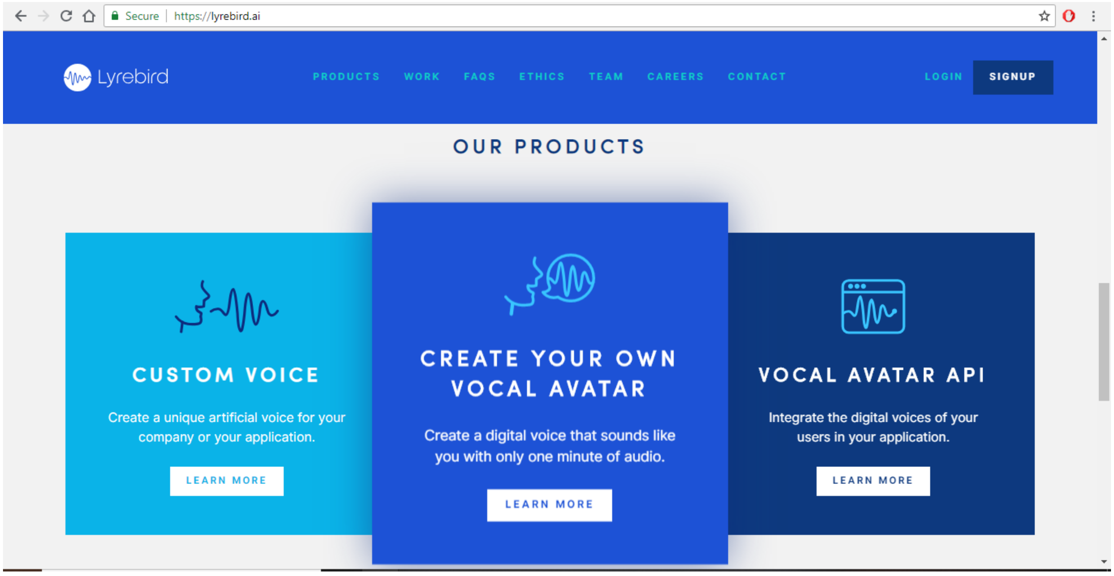


Figure 2.5 Screenshot of Lyrebird in the website [28].

2.7.2 Google Translate

Google Translate is one of Google products that is an application to translate languages. Google Translate can be access freely through *https://translate.google.com/*. It has many features [29] and one of them is Talk feature. Talk has function to input text from speech and generate speech from text in any languages.

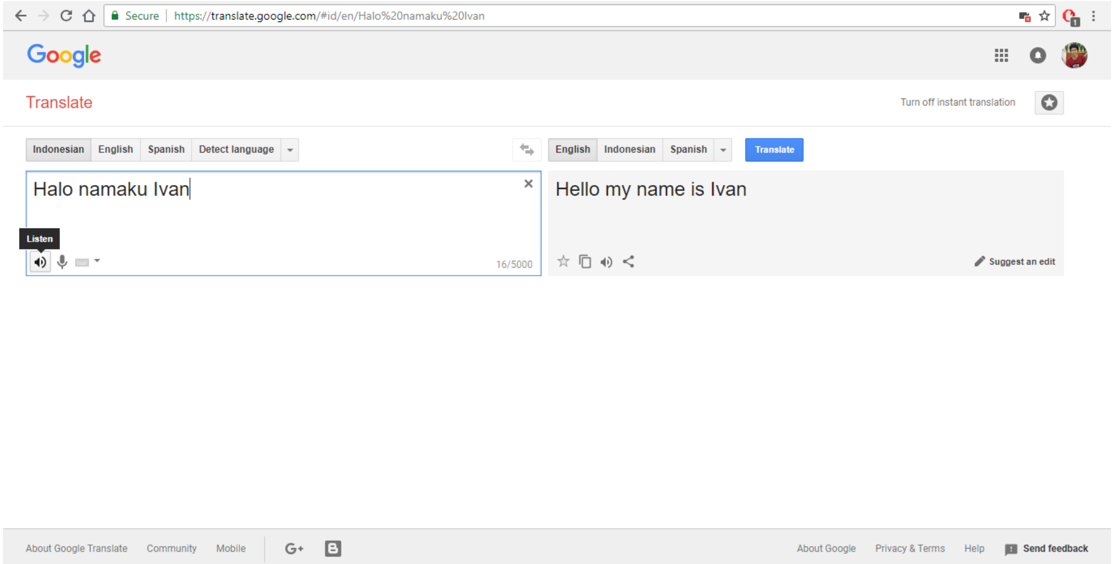


Figure 2.6 Screenshot of Google Translate in website [29].

From related work, it can be concluded that Lyrebird can mimic speech with its vocal avatar, but the speech is in English. In the other hand, Google Translate could mimic speech into any languages, but the speech vocal is from the Google Translate itself.

CHAPTER III

SYSTEM ANALYSIS

3.1 System Overview

…Update according to final app…

This research is intended to implement speech recognition and speech synthesis into this research. This application will be trained to recognize the speech before used by user. After enough training, this application will identify speech from the user based on sentences that will be displayed. Then, with speech synthesis user can generate speech from identified speech that will become mimic speech. The objective of this research is to create a web-based application for mimic speech by identify user speech and then generated them.

3.2 Functional Analysis

… Update according to final app…

There are several functions from this application listed in the Table 3.1.

Table 3.1 Functionality Table.

No	Function Description
1	Allow user to identify user's speech.
2	Allow user to select which speech data that will be used.
3	Allow user to generated speech.

3.3 Software and System Requirements

This research and application development should be supported by the following list requirement in order to write the research, build and run the application well.

1. Laptop / Personal Computer

   Laptop or Personal Computer is used as the tool where operating system is run. In this research, ASUS A455LN is used with Windows 10 as the OS.

2. Browser
3. Microsoft Office Word

   Microsoft Office Word is used to write the research documentation. In this research, Microsoft Office Word 2016 is used.

4. Node.js, JavaScript Run-Time Environment.

   Node.js is an open source server environment – Node.js is free – Node.js runs on various platform (Windows, Linux, Unix, Mac OS X, etc) – Node.js uses JavaScript on the server [30]. In this research, Node.js v10.14.1 is used.

5. NoSQL document-oriented database

   Document databases pair each key with a complex data structure known as a document. Documents can contain many different key-value pairs, or key-array pairs, or even nested documents [31]. In this research, MongoDb Community Server 4.0.2 as database server and MongoDb Compass 1.15.4 as MongoDb UI.

6. Integrated Development Environment (IDE)

IDE is used as application development environment. In this research, Visual Studio Code 1.30.0 is used.

7. Git

Git is used as version control. The repository is placed on local and cloud as preventive action. In this research, Git 2.18.0.windows.1 and GitLab as cloud repository is used.

3.4 System Architecture

… Update according to final app…

This sub-chapter discusses about the use-case diagram and narrative for this application in both point of view, the actors and the system.

3.4.1 Use-Case Diagram

Use-Case Diagram defines the functionality of a system and explains it in user point of view. The actors in this research is the application user. The diagram will be shown in Figure 3.1.

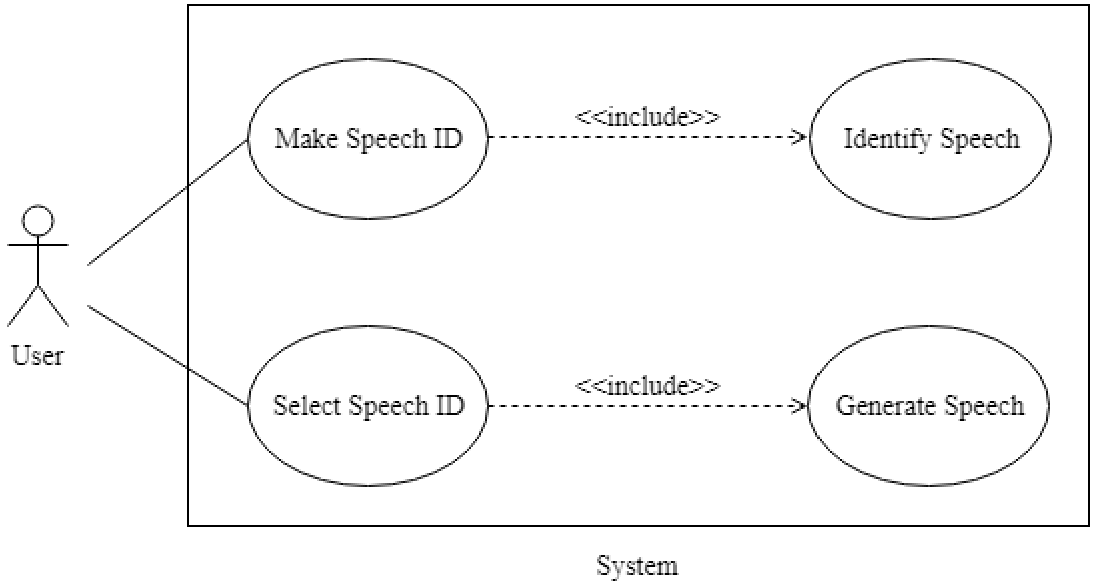


Figure 3.1 Use-Case Diagram.

3.4.2 Use-Case Narrative

Use-Case Narrative explains the interaction between the actors and the system. It describes the detail of use-cases such as name, description, pre-condition, post-condition, business rules, and the course of events that happened in the system. The Use-Case Narrative is shown in Table 3.2 and Table 3.5.

Table 3.2 Use-Case Narrative – Make Speech ID.

User Case Name	**Make Speech ID**
Use Case ID	UC01
Priority	High
Primary Business Actor	User
Primary System Actor	System
Another Participating Actor	None
Description	This use-case describes the event when user opens this application or in the home screen.

Precondition	None	
Trigger	User opens this application or user in the home screen.	
Typical Course of Event	**Actor Action**	**System Response**
	Choose Make Speech ID.	Start Make Speech ID activity.
Alternate Course	None	
Post Condition	Identify Speech screen is shown.	
Business Rule	None	
Implementation Constraint and Specifications	None	

Table 3.3 Use-Case Narrative – Identify Speech.

User Case Name	**Make Speech ID**	
Use Case ID	UC02	
Priority	High	
Primary Business Actor	User	
Primary System Actor	System	
Another Participating Actor	None	
Description	This use-case describes the event when Make Speech ID activity start.	
Precondition	User is from home screen.	
Trigger	User click Make Speech ID button in the home screen.	
Typical Course of Event	**Actor Action**	**System Response**
	Do Identify Speech.	Process Speech to Speech Data.
Alternate Course	**Actor Action**	**System Response**
	Finish Identify Speech.	Back to home screen.
Post Condition	User do identify speech again or Home screen is shown.	
Business Rule	None	
Implementation Constraint and Specifications	One speech ID for 1 speech data.	

Table 3.4 Use-Case Narrative – Select Speech ID.

User Case Name	**Select Speech ID**	
Use Case ID	UC03	
Priority	High	
Primary Business Actor	User	
Primary System Actor	System	
Another Participating Actor	None	
Description	This use-case describes the event when user opens this application or in the home screen.	
Precondition	None	
Trigger	User opens this application or user in the home screen.	
Typical Course of Event	**Actor Action**	**System Response**
	Select Speech ID.	Provide Speech ID.
Alternate Course	None	
Post Condition	Generate Speech screen is shown.	
Business Rule	None	
Implementation Constraint and Specifications	None	

Table 3.5 Use-Case Narrative – Generate Speech.

User Case Name	**Select Speech ID**	
Use Case ID	UC04	
Priority	High	
Primary Business Actor	User	
Primary System Actor	System	
Another Participating Actor	None	
Description	This use-case describes the event when Select Speech ID activity start.	
Precondition	User is from home screen.	
Trigger	User click Select Speech ID button in the home screen.	
Typical Course of Event	**Actor Action**	**System Response**
	Selected Speech ID.	Start Generate Speech Activity.
	Generate Speech.	Speech generated based on speech ID data.

Alternate Course	**Actor Action**	**System Response**
	Finish Generate Speech.	Back to home screen.
Post Condition	User do generate speech again or Home screen is shown.	
Business Rule	None	
Implementation Constraint and Specifications	None	

3.4.3 Activity Diagram

Activity Diagram presents a flowchart to represent the flow from one activity to another activity. As user open the application the home screen is shown. User could decide by choose to Make Speech ID or Select Speech ID. As long as user don’t decide it stays in the home screen. If Make Speech ID is selected, Make Speech ID activity is started which is Identify Speech. Finish the activity will bring user back to home screen. If Select Speech ID is selected, Select Speech ID activity is started which is Generated Speech. Finish the activity will bring user back to home screen. The figure of the activity diagram can be seen in Figure 3.2.

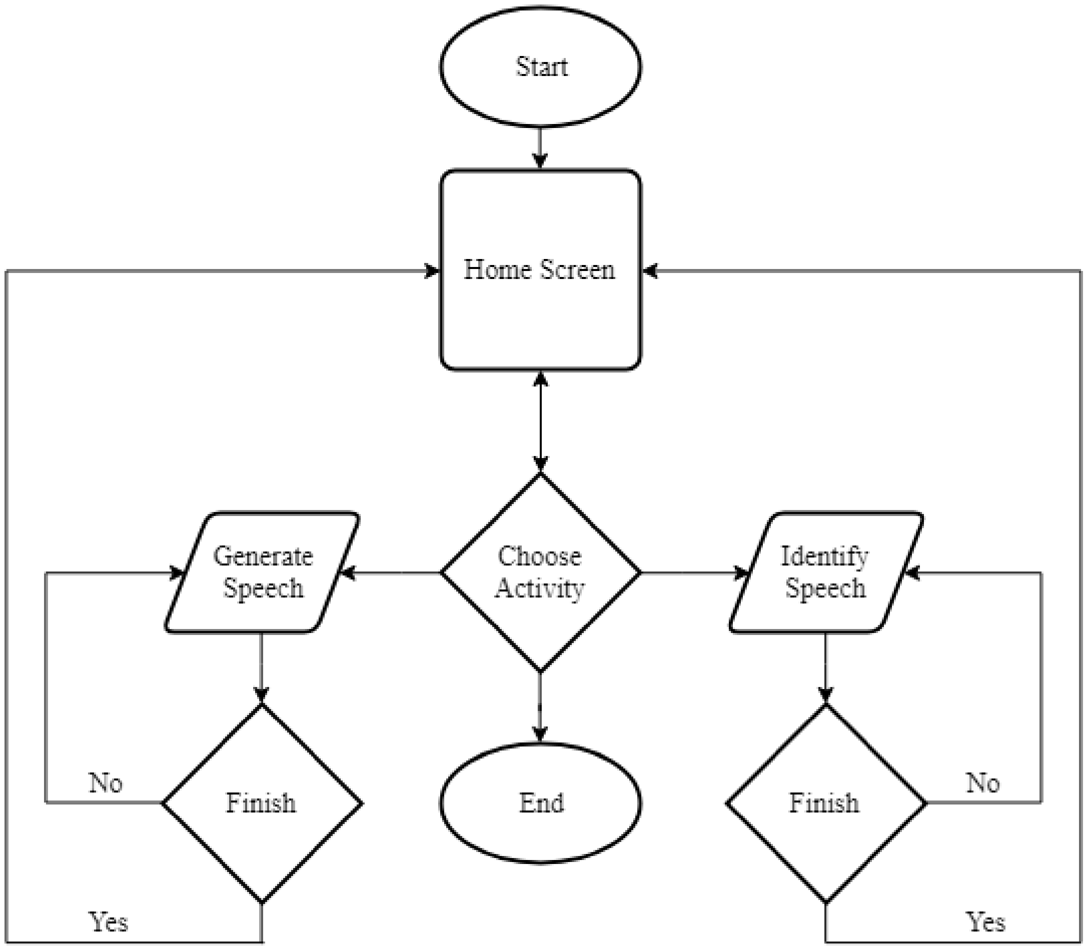


Figure 3.2 Activity Diagram.

CHAPTER IV

SYSTEM DESIGN

4.1 User Interface Design

…Update according to final app…

The User Interface (UI) design of this mobile application is divided into several features which are home screen, identify speech screen, and generate speech screen. The detail of every feature will be explained further below.

4.1.1 Home Screen

Figure 4.1 shows the design layout for home screen of the application. When user opens the application or finish Make Speech ID or Select Speech ID, it will show the home screen that consist of 2 buttons such as Make Speech ID and Select Speech ID. The description of the design layout is shown in Table 4.1.

Figure 4.1 Home Screen.

Table 4.1 Home Screen Description.

No	Description
1	Make Speech ID
2	Select Speech ID

4.1.2 Identify Speech Screen

Figure 4.2 shows the design layout for identify speech screen of the application. When user choose or click Make Speech ID, it will show the identify speech screen that consist of 4 element such as Random Sentences, Record Button, Identify Button, and Finish Button. The description of the design layout is shown in Table 4.2.

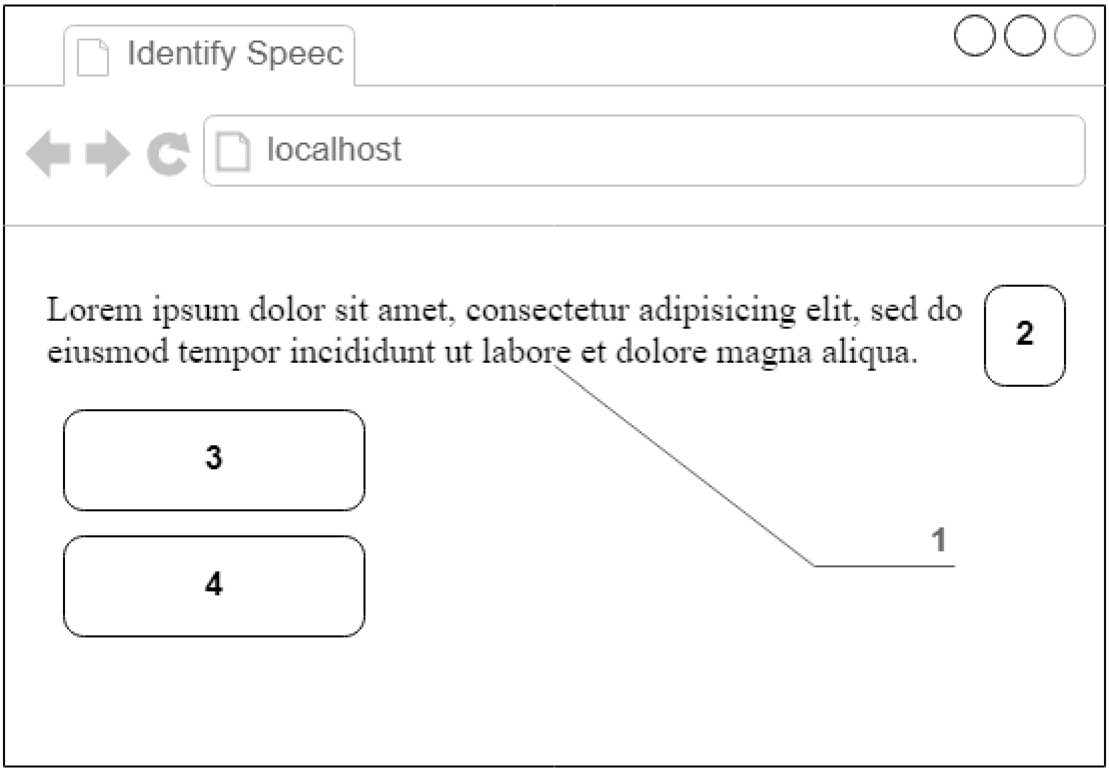


Figure 4.2 Identify Speech Screen.

Table 4.2 Identify Speech Screen Description.

No	**Description**
1	Random Sentences
2	Record Button
3	Identify Button
4	Finish Button

4.1.3 Generate Speech Screen

Figure 4.3 shows the design layout for generate speech screen of the application. When user choose or click Select Speech ID, it will show the generate speech screen that consist of 4 element such as ID Selector, Textbox Form, Generate and Play Button, and Finish Button. The description of the design layout is shown in Table 4.3.

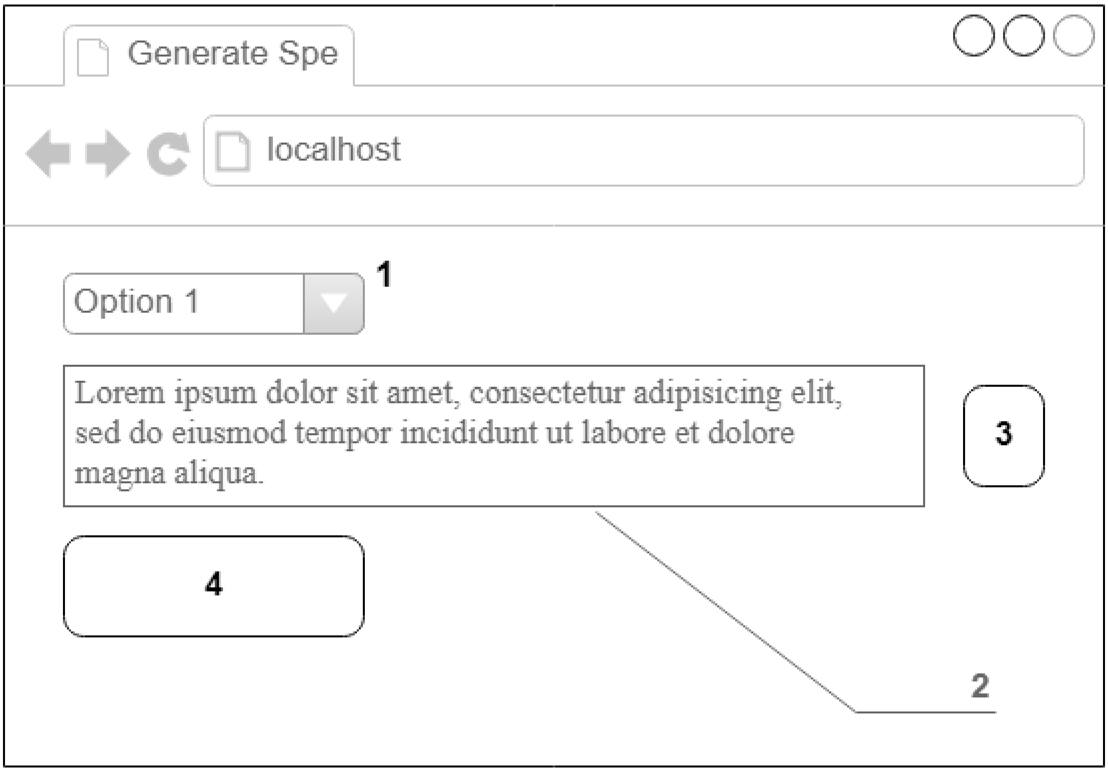


Figure 4.3 Generate Speech Screen.

Table 4.3 Generate Speech Screen Description.

No	Description
1	ID Selector
2	Textbox Form
3	Generate and Play Button
4	Finish Button

4.2 Class Diagram

…Update according to final app…

The class diagram is the structure of the system used toward this research. In this application there are 2 main category libraries, front-end and back-end. The detail of every libraries will be explained further below.

4.2.1 Front-end Library

Front-end library is used as the application UI code. The front-end class diagram details are shown in Figure 4.4 Approximately there are 3 front-end classes as listed below.

1. Home

   `Home` class is class used to render Home screen. It consisting of 2 main methods, `identifyButton` and `generateButton`.

   The `identifyButton` is method to start Make Speech ID activity which open Identify Speech screen. The `generateButton` is method to start Select Speech ID activity which open Generate Speech screen.

2. IdentifySpeech

   `IdentifySpeech` class is Make Speech ID activity. It is class used to render Identify Speech screen. It consisting of 4 main methods, `randomSentences`, `recordButton`, `identifyButton`, and `finishButton`.

   The `randomSentences` is method to randomize sentence that used to sentence user need to be said. The `recordButton` is method to input user recorded speech. The `identifyButton` is method to start identify user recorded speech and start `randomSentences` method again. The `finishButton` is method to finish Make Speech ID activity which open Home screen.

3. GenerateSpeech

`GenerateSpeech` class is Select Speech ID activity. It is class used to render Generate Speech screen. It consisting of 3 main methods, `selectID`, `generateButton`, and `finishButton`.

The `selectID` is method to select and define the speech ID from listed speech ID. The `generateButton` is method to generate and play speech based on inputted speech ID and text. The `finishButton` is method to finish Select Speech ID activity which open Home screen.

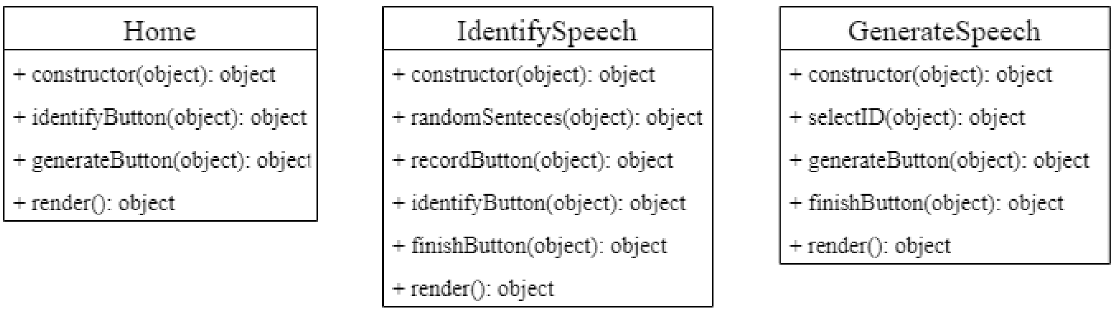


Figure 4.4 Front-end Class Diagram.

4.2.2 Back-end Library

Back-end library is used as speech recognition process, speech synthesis process, create-read-update-delete (CRUD) database process and database collections. The back-end class diagram details are shown in Figure 4.5 Approximately there are 4 back-end classes as listed below.

1. Speech Recognition

   `SpeechRecognition` class is class contains speech recognition algorithm. It containing `recognize` method to recognize speech from user and train speech.

2. Speech Synthesis

   `SpeechSynthesis` class is class contains speech synthesis algorithm. It containing `generate` method to classify inputted text then generate the speech from classified text.

3. MongoDb

   `MongoDb` class is class contains MongoDb process to established connection and data processing. It containing CRUD methods which is create, read, update, and delete data to MongoDb database.

4. MongoDb Collections

   MongoDb collections is class collection that is used as data type. It containing 4 main collections, `SpeechCollection`, `TextCollection`, `SpeechDataCollection`, and `SpeechTrainedCollection`.

The `SpeechCollection` is collection used to store all speech data classify by grapheme and gender, and store speech file location and trained speech file location. It consisting 4 main variables, `grapheme`, `gender`, `speechFile`, and `trainedFile`. The `TextCollection` is collection used to store text data classify by word and grapheme. It consisting 2 main variables, `word` and `grapheme`.

The `SpeechDataCollection` is collection used to store speech data classify by speechID and gender, and store identified speech. It consisting 3 main variables, `speechID`, `gender`, and `speech`. The `SpeechTrainedCollection` is collection used to store trained speech data classify by grapheme and gender, and store identified speech. It consisting 3 main variable, `grapheme`, `gender`, and `data`.

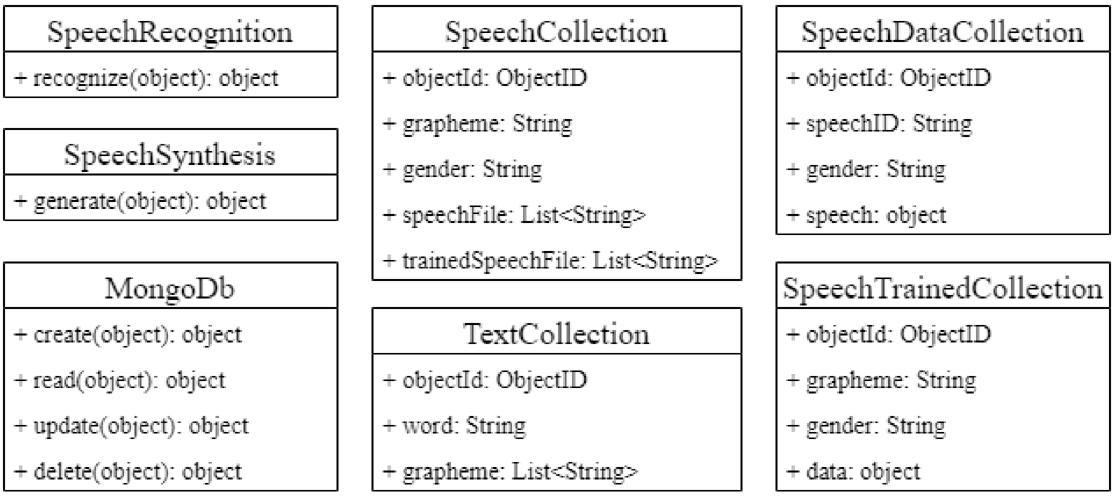


Figure 4.5 Bank-end Class Diagram.

CHAPTER V

SYSTEM DEVELOPMENT

5.1 User Interface Development

The UI is applied on Collect and Mimic application. There are two core pages in Collect application and three core pages in Mimic Application.

5.1.1 Collect Application

Two core pages in Collect application are Home page and Phonemes page.

5.1.1.1 Home Page

Home page is the home page of the Collect application. It shows information about how the collecting process works and information regarding to the collecting process. There are also 2 buttons, Start and Mimic Home button.

Start button will direct user to Phonemes page to start collecting. Mimic Home button will direct user to Mimic application Home page.

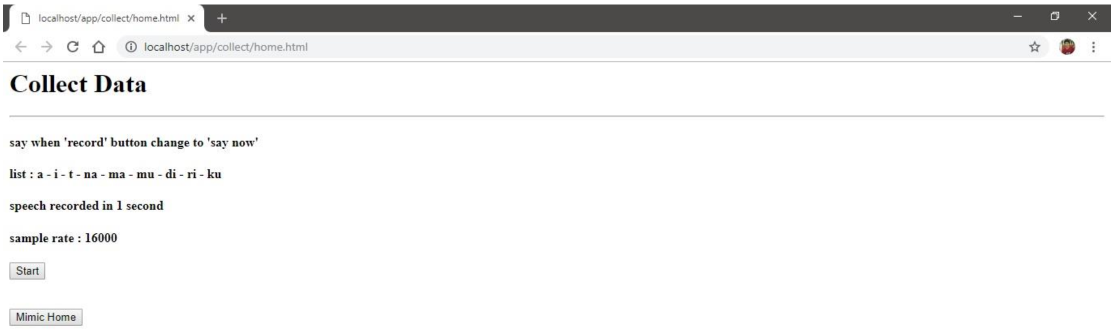


Figure 5.1 Collect Home page.

5.1.1.2 Phonemes Page

Phonemes page is the main page of the Collect application. It shows information about which phoneme should be recorded and replayed the record. There are also 2 buttons, Record and Next button and an Audio element. Any response from the server is also printed in the console.

Record button will start the record process. When the record process is starting, Record button will be renamed to ‘Say Now’ which tell user that the record process is starting and indicate user to say the relevant phoneme. When the record process is finish, the recorded blob is uploaded to the server and Record button will be renamed to ‘Record Again’ which tell user that the record process is finish and user can do the next record. Next button will direct user to the next Phonemes page. In this application there are 10 Phonemes. On the last phoneme, Next button will direct user back to Home page.

Audio element will be shown when the record process is finish with an audio from the record. User can hear the replay of the record from the Audio element.

Figure 5.2 Collect Phonemes page with ‘a’ phoneme.

Figure 5.3 Collect Phonemes page after record process is finish.

5.1.2 Mimic Application

Three core pages in Mimic application are Home page, Identify page, and Generate page.

5.1.2.1 Home Page

Home page is the home page of the Mimic application. It shows information about how the mimic process works and information regarding to the identify and generate process. There are also 3 buttons, Identify, Generate, and Collect Home button.

Identify button will direct user to Identify page to start identifying speech. Generate button will direct user to Generate page to start generating speech. Collect Home button will direct user to Collect application Home page.

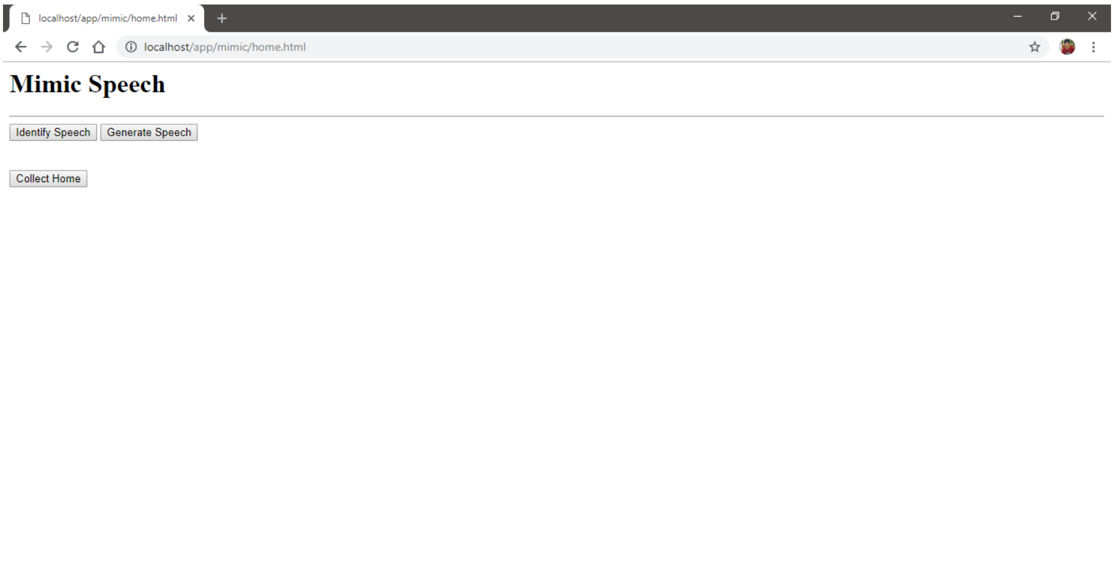


Figure 5.4 Mimic Home page.

5.1.2.2 Identify Page

Identify page is the one of the main pages of the Mimic application. It shows information about which phoneme is recognizable and form of input and audio that will be used to identify the speech and update it to database. There are also 3 buttons, Record, Identify Speech and Finish button, and an Input text and an Audio element. Any response from the server is also printed in the console.

Record button will start the record process. When the record process is starting, Record button will be renamed to 'Say Now' which tell user that the record process is starting and indicate user to say the phoneme. When the record process is finish, the recorded blob is uploaded to the server Record button will be renamed to 'Record Again' which tell user that the record process is finish and user can do the next record.

Identify Speech button will take the value of Input element and file source of Audio element, then, send them to the server to be identify. The indication of identify process in the server is success or error is there is alert on the browser that show any information regarding to identify process in the server. If there is no value of Input element or file source of Audio element, Identify Speech button will alert user that there is not any value. Finish button will direct user back to the Home page.

Audio element will be shown when the record process is finish with an audio from the record. User can hear the replay of the record from the Audio element. Input element is used as input of user speech ID value.

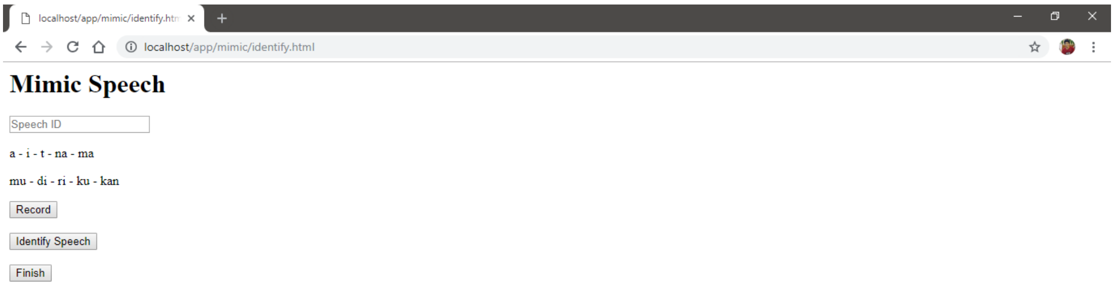


Figure 5.5 Mimic Identify page.

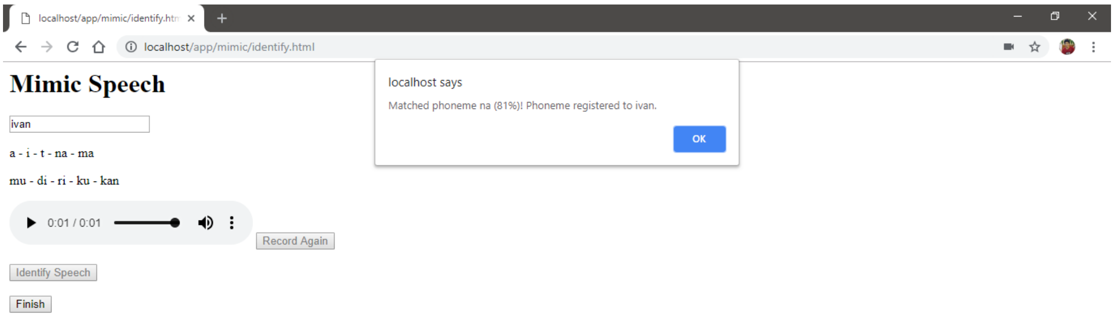


Figure 5.6 Mimic Identify page alert user indicating identify process in the server is finish.

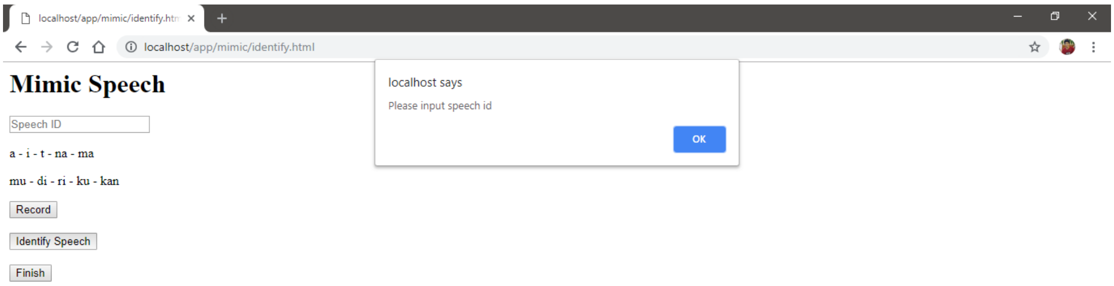


Figure 5.7 Mimic Identify page alert user indicating there is no value in Input element.

5.1.2.3 Generate Page

Generate page is the one of the main pages of the Mimic application. It shows form of data list and text area that will be used to generate the speech. When the page is loading, it also loads speech data from the database and then store it on data list. If there is no speech data found, it will alert user and direct user back to Home page. There are also 2 buttons, Generate Speech and Finish button, and a Datalist element, a Textarea element and an Audio element. Any response from the server is also printed in the console.

Generate Speech button will take the value of Datalist element and Textarea element, then, send them to the server to be generate. The indication of generate process in the server is error is there is alert on the browser that show any error information regarding to generate process in the server. The indication of generate process in the server is success is the Audio element is shown with an audio from the generated speech. If there's no value of Datalist element or Textarea element, Identify Speech button will alert user that there is not any value. Finish button will direct user back to the Home page.

Audio element will be shown when the generate process is success with an audio from the generated speech. User can hear the generated speech from the Audio element. Datalist element is shows all speech data from database and it is used as input of user speech ID value. Textarea element is used as input of words that user want to generate.

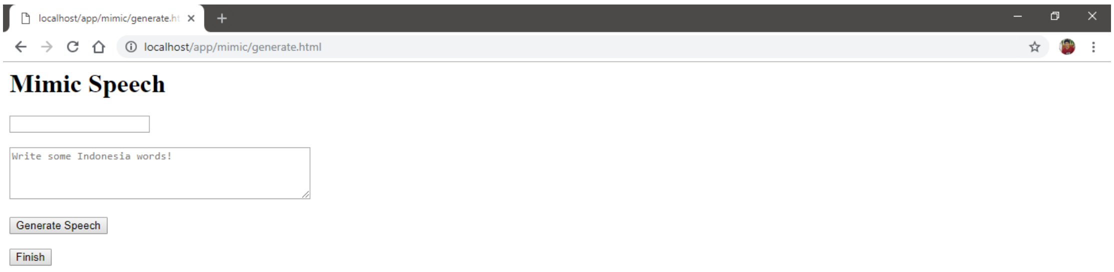


Figure 5.8 Mimic Generate page.

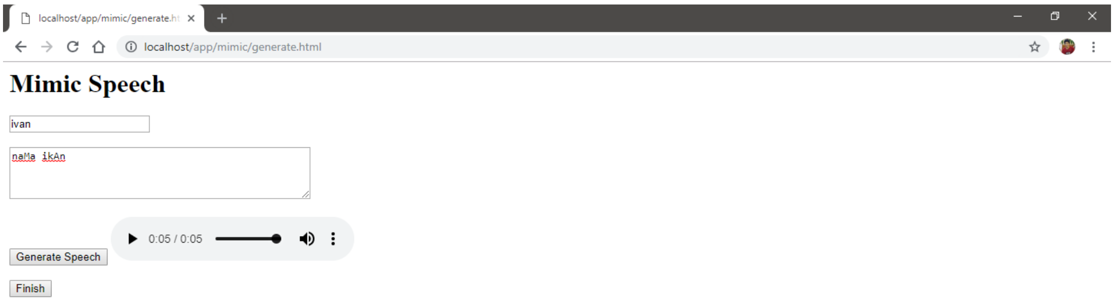


Figure 5.9 Mimic Generate page play the generate speech after generate process success.

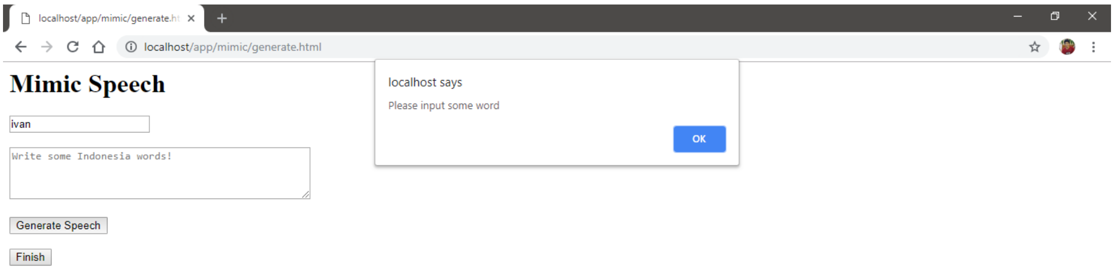


Figure 5.10 Mimic Generate page alert user indicating there is no value in Textarea element.

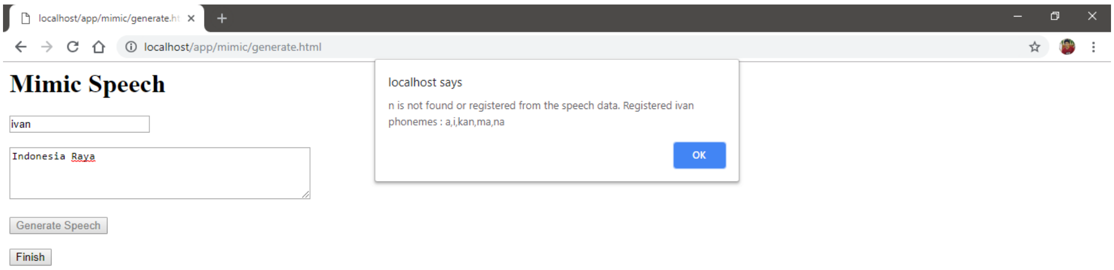


Figure 5.11 Mimic Generate page alert user indicating there is error in the generate process.

5.2 Application Details

The application details explain on server code, router code, database code, Train application code and library package that is used in the application. The figures that show the code also included comment that help explanation for corresponding line.

5.2.1 Server Code

The server code is used to create http server in the computer. It contains `serverHandler` and create server command itself.

`serverHandler` handle all request to the server each response corresponding to each request. There are 3 type main router `collect`, `identify`, `generate` to handle any request corresponding to them. When the request match it returns 200 alongside with the response corresponding to the router. When the request doesn't match any of those, it checks if the file URL as `filename` is existed in working directory or not. When the `filename` not found it return 404 Not Found. When found it read the `file` and return it with 200 so that it can be displayed by the browser. It returns 500 if error occurred when reading the `file`. used to create http server in the computer core pages in Mimic application are Home page, Identify page, and Generate page.

server.js

```javascript

/**
 * `serverHandler` handle all request to the server each response corresponding to each request.
 * There are 3 type main router `collect`, `identify`, `generate` to handle any request corresponding to them.
 * When the request match it return 200 alongside with the response corresponding to the router.
 * When the request doesn't match any of those, it check if the file url as `filename` is existed in working
 * directory or not. When the `filename` not found it return 404 Not Found. When found it read the `file` and return it with 200
 * so that it can be displayed by the browser. It return 500 if error occured when reading the `file`.
 */
async function serverHandler(request, response) {
  const uri = url.parse(request.url).pathname;
  const filename = path.join(process.cwd(), uri);

  const collect = await require('./router/collect')(address, port, filename, request);
  if (typeof collect !== 'undefined') {
    response.writeHead(200);
    response.write(collect);
    response.end();
    return;
  }

  const identify = await require('./router/identify')(address, port, filename, request);
  if (typeof identify !== 'undefined') {
    response.writeHead(200);
    response.write(identify);
    response.end();
    return;
  }

  const generate = await require('./router/generate')(address, port, filename, request);
  if (typeof generate !== 'undefined') {
    response.writeHead(200);
    response.write(generate);
    response.end();
    response.write(generate);
    response.end();
    return;
  }

  fs.exists(filename, function (exists) {
    if (!exists) {
      response.writeHead(404);
      response.write('404 Not Found: ' + filename + '\n');
      response.end();
      return;
    }

    fs.readFile(filename, 'binary', function (err, file) {
      if (err) {
        response.writeHead(500);
        response.write(err + '\n');
        response.end();
        return;
      }

      response.writeHead(200);
      response.write(file, 'binary');
      response.end();
    });
  });
}

```

Figure 5.12 serverHandler on Server code.

Create server command accept `serverHandler` as the parameter. It will listen on defined variable `localhost` and `port`. When the server established, it prints information in console.

server.js

```javascript

let app;

// create the server and listen with `serverHandler`.
app = server.createServer(serverHandler);
app = app.listen(port, address, function () {
  // print information regarding to the app.
  console.log('Could take time to load tensorflow before the page could be open.');
  console.log('Collect: ', 'http://' + address + ':' + port + '/app/collect/home.html');
  console.log('Mimic: ', 'http://' + address + ':' + port + '/app/mimic/home.html');
  console.log('Readme: ', 'http://' + address + ':' + port + '/readme.md');
});

```

Figure 5.13 Create server command on Server code.

5.2.2 Router Code

The router is used to handle categorized request from the server. It split into 3 categorized files, `collect`, `identify`, and `generate`.

Collect `router` handle all collect app request, each response corresponding to each request. There are a lot of request categorized by collect app but all of them is basically `uploadCollect`. When the request doesn't match anything, it returns nothing. `uploadCollect` upload file in the `request` based on defined directory and then convert its path location to URL. It returns the URL as `fileURL`.

collect.js

```javascript

/**
 * `uploadCollect` upload file in the `request` based on defined directory and
 * then convert its path location to url.
 *
 * Return the url as `fileURL`.
 */
function uploadCollect(address, port, label, request) {
  const dirData = '\\data\\';
  const dirCollect = '\\collect\\';
  const dirLabel = '\\' + label + '\\';

  const form = new formidable.IncomingForm();
  form.uploadDir = process.cwd() + dirData + dirCollect + dirLabel;
  form.keepExtensions = true;
  form.maxFieldsSize = 10 * 1024 * 1024;
  form.maxFields = 1000;
  form.multiples = false;

  return new Promise(function (resolve) {
    form.parse(request, function (err, fields, files) {
      const file = util.inspect(files);

      const fileName = file.split('path:')[1].split('\',')[0].split(dirData)[1].split(dirCollect)[1].split(dirLabel)[1].toString().replace(/\\/g, '').replace
      (/\//g, '');
      const fileURL = 'http://' + address + ':' + port + '/data/collect/' + label + '/' + fileName;

      console.log('fileURL: ', fileURL);
      resolve(JSON.stringify({ fileURL: fileURL }));
    });
  });
}

```

Figure 5.14 uploadCollect on Collect Router code.

collect.js

```javascript

/**
 * `router` handle all collect app request, each response corresponding to each request.
 * There are a lot of request categorized by collect app but all of them is basically `uploadCollect`.
 * When the request doesn't match anything it return nothing.
 */
async function router(address, port, filename, request) {
  let response;

  if (request.method === 'POST') {
    if (filename.toString().indexOf('\\uploadCollectA') != -1) response = await uploadCollect(address, port, 'a', request);
    else if (filename.toString().indexOf('\\uploadCollectI') != -1) response = await uploadCollect(address, port, 'i', request);
    else if (filename.toString().indexOf('\\uploadCollectT') != -1) response = await uploadCollect(address, port, 't', request);
    else if (filename.toString().indexOf('\\uploadCollectNa') != -1) response = await uploadCollect(address, port, 'na', request);
    else if (filename.toString().indexOf('\\uploadCollectMa') != -1) response = await uploadCollect(address, port, 'ma', request);
    else if (filename.toString().indexOf('\\uploadCollectMu') != -1) response = await uploadCollect(address, port, 'mu', request);
    else if (filename.toString().indexOf('\\uploadCollectDi') != -1) response = await uploadCollect(address, port, 'di', request);
    else if (filename.toString().indexOf('\\uploadCollectRi') != -1) response = await uploadCollect(address, port, 'ri', request);
    else if (filename.toString().indexOf('\\uploadCollectKu') != -1) response = await uploadCollect(address, port, 'ku', request);
    else if (filename.toString().indexOf('\\uploadCollectKan') != -1) response = await uploadCollect(address, port, 'kan', request);
  }

  return response;
}

```

Figure 5.15 router on Collect Router code.

Identify `router` handle all mimic app identify section request, each response corresponding to each request. There are 2 type requests categorized by mimic app identify section, `uploadSpeech` and `identifySpeech`. When the request doesn't match anything, it returns nothing. `uploadSpeech` upload file in the `request` based on defined directory and then convert its path location to URL. It returns the URL and path location as `fileURL` and `filePath`. `identifySpeech` load latest saved trained model from MongoDB `models` collection. Use it to identify the speech file from `filePath` by `extract` then feed it to `model`. Matched speech will register the phoneme to corresponding `name` and update it on MongoDB. Error connection will return message. Not found model will return message. Unsatisfied identification, results below 0.75 (75%) from `model` will return message. None of those is occurred will update the MongoDB `speechDatas` collection on its `name` and identified phoneme `filePath` and return message. The returned message whether just information or even error is return as `status`. `extract` is support function decode wav based on `filePath`, frame it, extract by FFT and MFCC and convert it to TensorFlow input shape. It returns 4d tensor.

identify.js

```
/**
 * `extract` decode wav based on `filePath`, frame it, extract by FFT and MFCC and
 * convert it to tensorflow input shape.
 *
 * Return 4d tensor.
 */
function extract(filePath) {
  const sampleRate = 16000;
  const frameSize = 25 / 1000 * sampleRate; // 400
  const frameShift = 10 / 1000 * sampleRate; // 160
  const fftSize = 256; // based on the next power of 2 from 400 -> 512 as the input.
  const melCount = 40;
  const lowHz = 300;
  const highHz = 8000;

  const mfcc = require('node-mfcc/src/mfcc').construct(fftSize, melCount, lowHz, highHz, sampleRate);

  // decode wav.
  const sample = wav.decode(fs.readFileSync(filePath)).channelData[0];

  // framing the sample.
  const frameSample = [];
  for (i = 0; i <= sample.length - frameSize; i = i + frameShift) {
    const frame = new Float32Array(fftSize * 2);
    for (j = 0; j < frame.length; j++) {
      if (j < frameSize) frame[j] = sample[i + j];
      else frame[j] = 0;
    }
    frameSample.push(frame);
  }

  // extract by FFT and MFCC.
  const extractedSample = [];
```

identify.js

```javascript

  // extract by FFT and MFCC.
  const extractedSample = [];
  for (i = 0; i < frameSample.length; i++) {
    const fftSample = fft(frameSample[i]);
    extractedSample.push(new Float32Array(mfcc(fftSample)));
  }

  // define tensorflow input shape.
  const time = extractedSample.length;
  const freq = extractedSample[0].length;
  const data = new Float32Array(time * freq);
  const shape = [1, time, freq, 1];

  // convert extracted wav to tensorflow input data.
  for (i = 0; i < time; i++) {
    const melFreq = extractedSample[i];
    const offset = i * freq;

    data.set(melFreq, offset);
  }

  // return 4d tensor.
  return tf.tensor4d(data, shape);
}

```

Figure 5.16 extract on Identify Router code.

identify.js

```javascript

/**
 * `uploadSpeech` upload file in the `request` based on defined directory and
 * then convert its path location to url.
 *
 * Return the url and path location as `fileURL` and `filePath`.
 */
function uploadSpeech(address, port, request) {
  const dirData = '\\data\\';
  const dirSpeech = '\\speech\\';

  const form = new formidable.IncomingForm();
  form.uploadDir = process.cwd() + dirData + dirSpeech;
  form.keepExtensions = true;
  form.maxFieldsSize = 10 * 1024 * 1024;
  form.maxFields = 1000;
  form.multiples = false;

  return new Promise(function (resolve) {
    form.parse(request, function (err, fields, files) {
      const file = util.inspect(files);

      const fileName = file.split('path:')[1].split('\',')[0].split(dirData)[1].split(dirSpeech)[1].toString().replace(/\\/g, '').replace(/\//g, '');
      const fileURL = 'http://' + address + ':' + port + '/data/speech/' + fileName;
      const filePath = files.file.path;

      console.log('fileURL: ', fileURL);
      console.log('filePath: ', filePath);
      resolve(JSON.stringify({ fileURL: fileURL, filePath: filePath }));
    });
  });
}

```

Figure 5.17 uploadSpeech on Identify Router code.

identify.js

```javascript

/**
 * `identifySpeech` load lastest saved trained model from MongoDB `models` collection.
 * Use it to identify the speech file from `filePath` by `extract` then feed it to `model`.
 * Matched speech will register the phoneme to corresponding `name` and update it on MongoDB.
 *
 * Error connection will return message. Not found model will return message. Unsatisfied identification,
 * results below 0.75 (75%) from `model` will return message. None of those is occured will update the MongoDB
 * `speechDatas` collection on its `name` and identified phoneme `filePath` and return message. The returned
 * message wheter just information or even error is return as `status`.
 */
function identifySpeech(request) {
  const form = new formidable.IncomingForm();

  return new Promise(function (resolve) {
    form.parse(request, async function (err, fields, files) {
      // take `name` and `filePath` data from POST request.
      const name = fields.name;
      const filePath = fields.filePath;

      let status;
      let resultDB = {};

      // load model location from MongoDB.
      try {
        model.models.db = await connection.connect();
        resultDB = await model.models.findOne({}).sort({ createAt: -1 }).select('location -_id').exec();
        await connection.disconnect();

        // return if nothing is found.
        if (typeof resultDB.location === 'undefined') {
          status = 'No train model found.';
          console.log('Status:', status);
        if (typeof resultDB.location === 'undefined') {
          status = 'No train model found.';
          console.log('Status:', status);

          resolve(JSON.stringify({ status: status }));
        }
      } catch (err) { // return if connection error is occured.
        status = err.errmsg;
        console.log('Status:', status);

        resolve(JSON.stringify({ status: status }));
      }

      // load the model as `tfModel` from model location and extract speech as `data` from `filePath`.
      const locationURL = resultDB.location;
      const tfModel = await tf.loadModel(locationURL + '/model.json');
      const data = extract(filePath);

      // identify the speech and process data from it.
      const prediction = tfModel.predict(data).dataSync();
      let predictionLabel = 0;
      for (i = 0; i < prediction.length; i++) {
        if (prediction[predictionLabel] <= prediction[i]) predictionLabel = i;
      }
      const label = labels[predictionLabel]
      const labelData = prediction[predictionLabel]

      // return if result unsatisfied, below 75%.
      if (labelData < 0.75) {
        status = 'No phonemes matched.';
        console.log('Status:', status);

        resolve(JSON.stringify({ status: status }));
      } else {
```

```
      status = 'No phonemes matched.';
      console.log('Status:', status);

      resolve(JSON.stringify({ status: status }));
    } else {
      // find if `name` is already registered in MongoDB or not.
      try {
        model.speechDatas.db = await connection.connect();
        resultDB = await model.speechDatas.findOne({ name: name }).select('phonemes -_id').exec();

        // create the document if not found, update if found before update to MongoDB.
        let updateData = {};
        if (resultDB === null) {
          updateData.name = name;
          updateData.phonemes = {};
          updateData.phonemes[label] = filePath
        }
        else {
          updateData = resultDB;
          updateData.phonemes[label] = filePath
        }
        await model.speechDatas.updateOne({ name: name }, updateData, { upsert: true });

        await connection.disconnect();
      } catch (err) { // return if connection error is occured.
        status = err.errmsg;
        console.log('Status:', status);

        resolve(JSON.stringify({ status: status }));
      }

      status = 'Matched phoneme ' + label + ' (' + (labelData.toFixed(2) / 1 * 100) + '%)! Phoneme registered to ' + name + '.';
      console.log('Status:', status);
      status = 'Matched phoneme ' + label + ' (' + (labelData.toFixed(2) / 1 * 100) + '%)! Phoneme registered to ' + name + '.';
      console.log('Status:', status);

      // return success identify speech information.
      resolve(JSON.stringify({ status: status }));
    }
  });
 });
}
```

Figure 5.18 identifySpeech on Identify Router code.

identify.js

```
/**
 * `router` handle all mimic app identify section request, each response corresponding to each request.
 * There are 2 type request categorized by mimic app identify section, `uploadSpeech` and `identifySpeech`.
 * When the request doesn't match anything it return nothing.
 */
async function router(address, port, filename, request) {
  let response;

  if (request.method === 'POST') {
    if (filename.toString().indexOf('\\uploadSpeech') != -1) response = await uploadSpeech(address, port, request);
    if (filename.toString().indexOf('\\identifySpeech') != -1) response = await identifySpeech(request);
  }

  return response;
}
```

Figure 5.19 router on Identify Router code.

generate `router` handle all mimic app generate section request, each response corresponding to each request. There are 2 type requests categorized by mimic app identify section, `loadSpeech` and `generateSpeech`. When the request doesn't match anything, it returns nothing. `loadSpeech` load all registered name from MongoDB `speechDatas` collection. It returns the array of name as `data` and `error` message if nothing is found or error. `generateSpeech` load registered phonemes from MongoDB `speechDatas` collection based on `name`. Extract `words` as `extractWord`. Decode all coresponding `extractWord` based on the database result. them and encode to wav file and take its `fileURL` location. It returns the URL as `fileURL` and `error` message if connection error or unregistered phoneme is found.

generate.js

```javascript

/**
 * `loadSpeech` load all registered name from MongoDB `speechDatas` collection.
 *
 * Return the array of name as `data` and `error` message if nothing is found or
 * connection error.
 */
async function loadSpeech() {
  let resultDB = [];
  let error;

  try {
    model.speechDatas.db = await connection.connect();
    resultDB = await model.speechDatas.find({}).select('name -_id').exec();
    await connection.disconnect();

    if (resultDB.length === 0) {
      error = 'No speech data found.';
      console.log('Error:', error);
    }
  } catch (err) {
    error = err.errmsg;
    console.log('Error:', error);
  }

  const data = resultDB.map((data) => data.name);

  console.log('speechData:', data);
  return JSON.stringify({ speechData: data, error: error });
}

```

Figure 5.20 loadSpeech on Generate Router code.

generate.js

```javascript

/**
 * `generateSpeech` load registered phonemes from MongoDB `speechDatas` collection based on `name`.
 * Extract `words` as `extractWord`. Decode all coresponding `extractWord` based on the database result.
 * Combine them and encode to wav file and take its `fileURL` location.
 *
 * Return the url as `fileURL` and `error` message if connection error or unregistered phoneme is found.
 */
function generateSpeech(address, port, request) {
  const form = new formidable.IncomingForm();

  return new Promise(function (resolve) {
    form.parse(request, async function (err, fields, files) {
      // take `name` and `words` data from POST request.
      const name = fields.name;
      const words = fields.words.toLowerCase();

      let fileURL;
      let error;
      let resultDB = {};
      let buffer;

      // load registered phonemes from MongoDB.
      try {
        model.speechDatas.db = await connection.connect();
        resultDB = await model.speechDatas.findOne({ name: name }).select('phonemes -_id').exec();
        await connection.disconnect();
      } catch (err) { // return if connection error is occured.
        error = err.errmsg;
        console.log('Error:', error);

        resolve(JSON.stringify({ fileURL: fileURL, error: error }));
      }

        resolve(JSON.stringify({ fileURL: fileURL, error: error }));
      }

      // take just the phonemes from `resultDB`.
      const phonemes = [];
      for (phoneme in resultDB.phonemes) phonemes.push(phoneme);

      // extract the `words`.
      const extractWord = [];
      for (i = 0; i < words.length; i++) {
        if (words[i] === ' ') { // check if it is space.
          extractWord.push(' ');
        } else if (i + 3 <= words.length && phonemes.includes(words.substring(i, i + 3))) { // check if it the next 3 char is in `phonemes`.
          extractWord.push(words.substring(i, i + 3));
          i = i + 2;
        } else if (i + 2 <= words.length && phonemes.includes(words.substring(i, i + 2))) { // check if it the next 2 char is in `phonemes`.
          extractWord.push(words.substring(i, i + 2));
          i = i + 1;
        } else if (phonemes.includes(words.substring(i, i + 1))) { // check if it the next char is in `phonemes`.
          extractWord.push(words.substring(i, i + 1));
        } else { // return if unregistered phoneme is found.
          error = words[i] + ' is not found or registered from the speech data. ' +
            'Registered ' + name + ' phonemes : ' + phonemes;
          console.log('Error:', error);

          resolve(JSON.stringify({ fileURL: fileURL, error: error }));
        }
      }

      // combine each `extractWord` into 1 `channelData`.
      const sampleRate = 16000;
      const channelData = [new Float32Array(0)];
      // iterate each `extractWord`.
      for (i = 0; i < extractWord.length; i++) {
```

```javascript
    // iterate each `extractWord`.
    for (i = 0; i < extractWord.length; i++) {
      const prevAmpli = channelData[0];
      const offset = channelData[0].length;

      let nextAmpli;
      if (extractWord[i] === ' ') { // if space it fill the `nextAmpli` with 0 `sampleRate` times.
        nextAmpli = new Float32Array(sampleRate).map(() => 0);
      } else { // else it fill from the decode result based on `resultDB` and corresponding `extractWord`.
        buffer = fs.readFileSync(resultDB.phonemes[extractWord[i]]);
        nextAmpli = wav.decode(buffer).channelData[0].slice(0, sampleRate);
      }

      // arrange the ampli and redefined the `channelData`.
      const ampli = new Float32Array(sampleRate + offset);
      ampli.set(prevAmpli);
      ampli.set(nextAmpli, offset);
      channelData[0] = ampli;
    }

    // define upload directory.
    const dirData = '\\data\\';
    const dirGenerate = '\\generate\\';
    const fileName = 'upload_' + cryptoRandomString(32) + '.wav';
    const file = process.cwd() + dirData + dirGenerate + fileName;

    // encode wav based on `channelData` and `sampleRate`
    buffer = wav.encode(channelData, { sampleRate: sampleRate });
    fs.writeFileSync(file, Buffer.from(buffer));

    // return the file url as `fileURL`.
    fileURL = 'http://' + address + ':' + port + '/data/generate/' + fileName;
    console.log('fileURL:', fileURL);
    // return the file url as `fileURL`.
    fileURL = 'http://' + address + ':' + port + '/data/generate/' + fileName;
    console.log('fileURL:', fileURL);
    resolve(JSON.stringify({ fileURL: fileURL }));
  });
 });
}
```

Figure 5.21 generateSpeech on Generate Router code.

```javascript
generate.js ×

/**
 * `router` handle all mimic app generate section request, each response corresponding to each request.
 * There are 2 type request categorized by mimic app identify section, `loadSpeech` and `generateSpeech`.
 * When the request doesn't match anything it return nothing.
 */
async function router(address, port, filename, request) {
  let response;

  if (request.method === 'GET') {
    if (filename.toString().indexOf('\\loadSpeech') != -1) response = await loadSpeech();
  } else if (request.method === 'POST') {
    if (filename.toString().indexOf('\\generateSpeech') != -1) response = await generateSpeech(address, port, request);
  }

  return response;
}
```

Figure 5.22 router on Generate Router code.

5.2.3 Database Code

The database code is used to create connection to MongoDB and create MongoDB collection model scheme. It contains `connection` and `model`.

`connection` define 2 functions `connect` and `disconnect`. `connect` establish connection to MongoDB mimic_speech database on localhost. Established connection is saved in `connection`. `disconnect` close established `connection` to MongoDB.

connection.js

```javascript

/**
 * `connect` establish connection to mongodb mimic_speech database on localhost.
 *
 * Establised connection is saved in `connection`.
 */
async function connect() {
  connection = await mongoose.connect('mongodb://localhost/mimic_speech', { useNewUrlParser: true });
  return connection;
};

/**
 * `disconnect` close established `connection` to mongodb.
 */
async function disconnect() {
  await connection.disconnect();
}

```

Figure 5.23 connection on Database code.

`model` define 2 schemes variable `speechDatas` and `models`. `speechDatas` define scheme with `name` as string and `phonemes` as object. `models` define scheme with `location` as string and `createAt` as date with current date being default value when create or update collection when `createAt` is not defined.

model.js

```javascript

// define `speechDatas` schema for its mongodb collection.
const speechDatas = mongoose.model('speechdatas', mongoose.Schema({
  name: String,
  phonemes: Object,
}));

// define `models` schema for its mongodb collection.
const models = mongoose.model('models', mongoose.Schema({
  location: String,
  createAt: {
    type: Date,
    default: Date.now
  }
}));

```

Figure 5.24 model on Database code.

5.2.4 Train Application Code

Train application code is used to train collected data and save the model on n iteration and last iteration so that later can be used to identify the speech. It contains `extractWav`, `loadData`, `nextBatch`, `saveModelDB` as support code and `model` and `train` as the main code.

`extractWav` decode wav based on `file` and `label` location, frame it and extract by FFT and MFCC. It returns JSON of extracted sample as `data` and `label`.

extractWav.js

```javascript

/**
 * `extractWav` decode wav based on `file` and `label` location, frame it and extract by FFT and MFCC.
 *
 * Return JSON of extracted sample as `data` and `label`.
 */
function extractWav(file, label) {
  // decode wav.
  const sample = wav.decode(fs.readFileSync(process.cwd() + '/data/collect/' + label + '/' + file)).channelData[0];

  // framing the sample.
  const frameSample = [];
  for (i = 0; i <= sample.length - frameSize; i = i + frameShift) {
    const frame = new Float32Array(fftSize * 2);
    for (j = 0; j < frame.length; j++) {
      if (j < frameSize) frame[j] = sample[i + j];
      else frame[j] = 0;
    }
    frameSample.push(frame);
  }

  // extract by FFT and MFCC.
  const extractedSample = [];
  for (i = 0; i < frameSample.length; i++) {
    const fftSample = fft(frameSample[i]);
    extractedSample.push(new Float32Array(mfcc(fftSample)));
  }

  return { data: extractedSample, label };
}

```

Figure 5.25 extractWav on Train application code.

`loadData` is used to load all collected data, then, extract them and split them into train data, validation data, and test data. http server in the computer. It contains `extractFiles` and load and split command itself. `extractFiles` extract wav from loaded files as `load`, assigned to corresponding `data`. It returns array of extracted waves. Every iteration when extracting the loaded files is printed. If inconsistent data is met, the process stops and exit.

loadData.js

```javascript

/**
 * `extractFiles` extract wav from loaded files as `load`, assigned to corresponding `data`.
 *
 * Return array of extracted waves. Every iteration when extracting the loaded files is printed.
 * If incosistent data is met, the process stop and exit.
 */
function extractFiles(load, data, type) {
  const data = [];
  let lastSpectogram;

  // shuffle `load` to avoid overfitting model.
  load = shuffle(load);

  for (k = 0; k < load.length; k++) {
    const spectogram = extractWav(load[k].file, load[k].label);
    if (k !== 0 && lastSpectogram !== 'undefined'
      && lastSpectogram.data.length !== spectogram.data.length) {
      console.log(load[k - 1].file + ' (label : ' + load[k - 1].label + ')' +
        ' is different with data [time] ' +
        load[k].file + ' (label : ' + load[k].label + '). ' +
        'Please check again. Default time is 1 second. Process exiting.');
      process.exit();
    } else {
      data.push(spectogram);
    }

    lastSpectogram = spectogram;
    console.log(new Date().toISOString() + ' | Step ' + (k + 1) + ' | Extracting ' + type + ' Wav: ' + load[k].file);
  }

  return data;
}

```

Figure 5.26 extractFiles on loadData Train application code.

loadData.js

```javascript

// iterate each `labels` to load files. Every iteration when loading the files is printed.
for (i = 0; i < labels.length; i++) {
  console.log(new Date().toISOString() + ' | Step ' + (i + 1) + ' | Load files from label: ' + labels[i]);

  // shuffle `files` to avoid overfitting model.
  const files = shuffle(fs.readdirSync(process.cwd() + '/data/collect/' + labels[i]));

  // 80% train data, 10% validation data, 10% test data.
  const validation = Math.floor(10 / 100 * files.length);
  const test = Math.floor(10 / 100 * files.length);
  const train = files.length - validation - test;
  for (j = 0; j < files.length; j++) {
    const data = {
      file: files[j],
      label: labels[i]
    };

    if (j < train) loadTrain.push(data);
    else if (j < train + validation) loadValidation.push(data);
    else if (j < train + validation + test) loadTest.push(data);
  }
}

const datatrain = extractFiles(loadTrain, datatrain, 'Train');
const datavalidation = extractFiles(loadValidation, datavalidation, 'Validation');
const datatest = extractFiles(loadTest, datatest, 'Test');

// put `labels`, `datatrain`, `datavalidation`, `datatest` to JSON and export it.
const dataset = { labels, datatrain, datavalidation, datatest }
```

Figure 5.27 load and split command on loadData Train application code.

`nextBatch` is used to get the next batch data and then convert it to train model input or output. It contains `getData` and `getLabel`. `getData` convert corresponding `dataset` data based on `iteration`, `batch`, `time`, `freq` to TensorFlow input. It defines TensorFlow input shape from `batch`, `time`, and `freq`. It defines TensorFlow input data convert it by rearrange the data inside `dataset` based on `batch`, `iteration`, `time`, and `freq`. Then it returns as 4d tensor. `getLabel` convert corresponding `dataset` label based on `iteration`, `batch` to TensorFlow output. It defines TensorFlow output shape from `batch`, and `labels.length`. It defines TensorFlow output data convert it by define array from `labels` based on `batch`, `iteration`, and `dataset`. Then it returns as 2d tensor.

nextBatch.js

```javascript

/**
 * `getLabel` convert corresponding `dataset` label based on `iteration`,
 * `batch` to tensorflow output.
 *
 * Define tensorflow output shape from `batch`, and `labels.length`.
 * Define tensorflow output data convert it by define array from `labels` based on
 * `batch`, `iteration`, and `dataset`. Then it return as 2d tensor.
 */
function getLabel(labels, dataset, iteration, batch) {
  const data = [];
  const shape = [batch, labels.length];

  for (j = 0; j < batch; j++) {
    let label;

    if (iteration + j >= dataset.length) label = dataset[(iteration + j) % dataset.length].label;
    else label = dataset[iteration + j].label;

    for (k = 0; k < labels.length; k++) {
      if (label === labels[k]) data.push(1);
      else data.push(0);
    }
  }

  return tf.tensor2d(data, shape);
}

```

Figure 5.28 getData on nextBatch Train application code.

nextBatch.js

```javascript

/**
 * `getLabel` convert corresponding `dataset` label based on `iteration`,
 * `batch` to tensorflow output.
 *
 * Define tensorflow output shape from `batch`, and `labels.length`.
 * Define tensorflow output data convert it by define array from `labels` based on
 * `batch`, `iteration`, and `dataset`. Then it return as 2d tensor.
 */
function getLabel(labels, dataset, iteration, batch) {
  const data = [];
  const shape = [batch, labels.length];

  for (j = 0; j < batch; j++) {
    let label;

    if (iteration + j >= dataset.length) label = dataset[(iteration + j) % dataset.length].label;
    else label = dataset[iteration + j].label;

    for (k = 0; k < labels.length; k++) {
      if (label === labels[k]) data.push(1);
      else data.push(0);
    }
  }

  return tf.tensor2d(data, shape);
}

```

Figure 5.29 getLabel on nextBatch Train application code.

`saveModelDB` save `location` to `models` collection in mimic_speech database in MongoDB.

saveModelDB.js

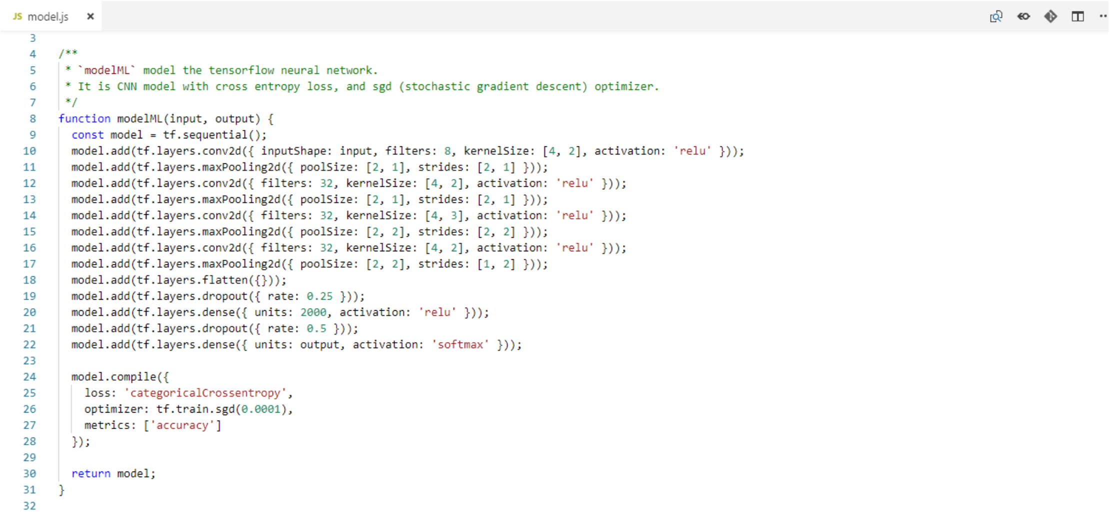
```javascript

/**
 * `saveModelDB` save `location` to `models` collection in mimic_speech database in MongoDB.
 */
async function saveModelDB(location) {
  try {
    model.models.db = await connection.connect();
    await model.models.create({ location });
    await connection.disconnect();
  } catch (err) {
    error = err.errmsg;
    console.log('Error:', error);
  }
}

```

Figure 5.30 saveModelDB on Train application code.

`model` as `modelML` model the TensorFlow neural network. It is CNN model with cross entropy loss, and SGD (Stochastic Gradient Descent) optimizer.



```
model.js

/**
 * `modelML` model the tensorflow neural network.
 * It is CNN model with cross entropy loss, and sgd (stochastic gradient descent) optimizer.
 */
function modelML(input, output) {
  const model = tf.sequential();
  model.add(tf.layers.conv2d({ inputShape: input, filters: 8, kernelSize: [4, 2], activation: 'relu' }));
  model.add(tf.layers.maxPooling2d({ poolSize: [2, 1], strides: [2, 1] }));
  model.add(tf.layers.conv2d({ filters: 32, kernelSize: [4, 2], activation: 'relu' }));
  model.add(tf.layers.maxPooling2d({ poolSize: [2, 1], strides: [2, 1] }));
  model.add(tf.layers.conv2d({ filters: 32, kernelSize: [4, 3], activation: 'relu' }));
  model.add(tf.layers.maxPooling2d({ poolSize: [2, 2], strides: [2, 2] }));
  model.add(tf.layers.conv2d({ filters: 32, kernelSize: [4, 2], activation: 'relu' }));
  model.add(tf.layers.maxPooling2d({ poolSize: [2, 2], strides: [1, 2] }));
  model.add(tf.layers.flatten({}));
  model.add(tf.layers.dropout({ rate: 0.25 }));
  model.add(tf.layers.dense({ units: 2000, activation: 'relu' }));
  model.add(tf.layers.dropout({ rate: 0.5 }));
  model.add(tf.layers.dense({ units: output, activation: 'softmax' }));

  model.compile({
    loss: 'categoricalCrossentropy',
    optimizer: tf.train.sgd(0.0001),
    metrics: ['accuracy']
  });

  return model;
}
```

Figure 5.31 modelML on model Train application code.

`train` contains preparation command for train and `train` itself. `train` train the `model` to fit the `dataset.datatrain` and validate the `model` on `dataset.datavalidation` data by its loss and accuracy. The train iterate `trainIteration` times and test every `testIteration` in `trainIteration`. Every iteration `loss` and `accuracy` are printed. Every test is occurred and the last iteration, `confusionMatrix` is printed, `model` is saved, the saved model location is saved to MongoDB, `loss` and `accuracy` is saved as csv.

```javascript
train.js

// load all data to `dataset`.
const dataset = require('./lib/loadData');

// define input and output tensorflow shape.
const time = dataset.datatrain[0].data.length;
const freq = dataset.datatrain[0].data[0].length;
const inputShape = [time, freq, 1];
const outputShape = dataset.labels.length;

const model = require('./model')(inputShape, outputShape); // load tensorflow model to `model`.
const batch = 32; // n inputs at the same time to be trained.
const epochs = 16; // n times the batches to be trained.
const trainIteration = 1500; // n times the train iteration.
const testIteration = 300; // Every n times in `trainIteration` the `model` is tested.

// define all test input and output from `dataset.datatest` to `testData` and `testLabel` for prediction.
const testData = nextBatch.getData(dataset.datatest, i, dataset.datatest.length, time, freq);
const testLabel = nextBatch.getLabel(dataset.labels, dataset.datatest, i, dataset.datatest.length);

```

Figure 5.32 preparation command on train Train application code.

```javascript
train.js

/**
 * `train` train the `model` to fit the `dataset.datatrain` and validate the `model` on
 * `dataset.datavalidation` data by its loss and accuracy.
 *
 * Train iterate `trainIteration` times and test every `testIteration` in `trainIteration`.
 * Every iteration `loss` and `accuracy` is printed. Every test is occured and the last iteration,
 * `confusionMatrix` is printed, `model` is saved, the saved model location is saved to MongoDB,
 * `loss` and `accuracy` is saved as csv.
 */
async function train() {
  for (i = 0; i < trainIteration; i++) {
    // define train input and output from `dataset.datatrain` and validation from `dataset.datavalidation`.
    const batchData = nextBatch.getData(dataset.datatrain, i, batch, time, freq);
    const batchLabel = nextBatch.getLabel(dataset.labels, dataset.datatrain, i, batch);
    const batchValidation = nextBatch.getData(dataset.datavalidation, i, batch, time, freq);

    // train the model and get `loss` and `accuracy`.
    const history = await model.fit(
      batchData,
      batchLabel,
      { batchSize: batch, validationData: batchValidation, epochs: epochs, verbose: 0 }
    );
    loss.push(history.history.loss[0].toFixed(10));
    accuracy.push(history.history.acc[0].toFixed(5));

    console.log(new Date().toISOString() + ' | Step ' + (i + 1) + ' | accuracy: ' + accuracy[i] + ' | loss: ' + loss[i]);

    // check if it is n `testIteration`.
    if ((i !== 0 && i % testIteration === 0) || (i + 1 === trainIteration)) {
      // test the model and show `confusionMatrix` the test result.
      const labels = tf.argMax(testLabel, 1);
      const predictions = tf.argMax(model.predict(testData), 1);
      const confusionMatrix = tf.math.confusionMatrix(labels, predictions, dataset.labels.length);
```

Figure 5.33 train on train Train application code.

5.2.5 Library Package Code

CHAPTER VI

SYSTEM TESTING

6.1 Testing Environment

The testing environment specify the environment during testing. The environment specification are as follows:

1. Windows 10.
2. Chrome browser.

There is additional environment only on the identify process. User that the records is trained by the application is called Tester 1. Male user that the record hasn't trained by the application called Tester 2. Female user that the record hasn't trained by the application called Tester 3. Every tester is test in each the following environment:

1. Noisy background (Loud music or people chit-chat).
2. Semi noisy background (AC noise or sound from the other rooms).
3. No noisy background.

6.2 Testing Scenario

The testing scenario conducted by evaluating all the features with a set of cases or scenario in the application based on its functionality requirement and defined testing environment.

The testing scenarios of the application is categorized based on Collect application, Train application, and Mimic application.

6.2.1 Collect Application

The Collect application has 3 subcategorized scenarios, Collect server, Home page and Phonemes page. The Collect server has scenarios to allow user to access Home page, Phonemes page in the browser and process the upload record request. The Home page has scenarios to allow user to direct to Phonemes page to start the Collect application and direct to Mimic application Home page to change to Mimic application. The Phonemes page has scenarios to allow user to upload the record to the server and direct to the next Phonemes page or direct back to Home page. Figures 6.1 shows some screenshot of expected result from the scenarios.

Table 6.1 Collect application scenarios.

No	Scenario	Expected Result	Result
1	Access Home page	Home page is shown	As expected
2	Access Phonemes page	Phonemes page is shown	As expected
3	Process upload record request	Printed process results information on the console	As expected
4	Direct to Phonemes page	Directed to Phonemes page	As expected
5	Direct to Mimic application Home page	Directed to Mimic application Home page	As expected
6	Upload the record	Audio element is shown with the user record	As expected
7	Direct to the next Phonemes page or direct back to Home page	Directed to the next Phonemes page or Directed back to Home Page	As expected

Figures 6.1 Some screenshot from the Collect application scenarios.

6.2.2 Train Application

The Train application has scenario to allows user to train collected data. When train collected data is run, it loads and splits collected data, extracts the data, gets the corresponding batch data, trains the TensorFlow model to fit the batch train data, tests the TensorFlow model from test data, and saves the TensorFlow model to database. Figures 6.2 shows some screenshot of expected result from the scenarios.

Table 6.2 Train application scenarios.

No	Scenario	Expected Result	Result
1	Load and split collected data	Printed information on the console	As expected
2	Extract the data	Printed information on the console	As expected
3	Get corresponding batch data	Printed information on the console	As expected
4	Train the TensorFlow model to fit batch train data	Printed information on the console	As expected
5	Test the TensorFlow model from test data	Printed information on the console	As expected
6	Save the TensorFlow model to database	Recorded document in mimic_speech database models collection	As expected

Figures 6.2 Some screenshot from the Train application scenarios.

6.2.3 Mimic Application

The Mimic application has 4 subcategorized scenarios, Mimic server, Home page, Identify page, and Generate page. The Mimic server has scenarios to allow user to access Home page, Identify page, and Generate page in the browser and process the upload record and identify speech on Identify request and load speech data and generate speech on Generate request. The Home page has scenarios to allow user to direct to Identify page to start the identify speech application, direct to Generate page to start the generate speech application, and direct to Collect application Home page to change to Collect application. The Identify page has scenarios to allow user to upload the record to the server, send form request for identify speech to the server, and direct back to Home page. The Generate page has scenarios to allow user to load speech data from the server, send form request for generate speech to the server, and direct back to Home page. Figures 6.3 shows some screenshot of expected result from the scenarios.

Table 6.3 Mimic application scenarios.

No	Scenario	Expected Result	Result
1	Access Home page	Home page is shown	As expected
2	Access Identify page	Identify page is shown	As expected
3	Access Generate page	Generate page is shown	As expected
3	Process upload record on Identify request	Printed process results information on the console	As expected
4	Process identify speech on Identify request	Printed process results information on the console	As expected
5	Process load speech data on Generate request	Printed process results information on the console	As expected
6	Process generate speech on Generate request	Printed process results information on the console	As expected
7	Direct to Identify page	Directed to Identify page	As expected
8	Direct to Generate page	Directed to Generate page	As expected
9	Direct to Collect application Home page	Directed to Collect application Home page	As expected
10	Upload the record	Audio element is shown with the user record	As expected
11	Send form for identify speech	Alert information despite success or error in the process	As expected
12	Direct back to Home page from Identify page	Directed back to Home Page	As expected
13	Load speech data	Datalist element store the speech data or alert if no speech data is found	As expected

14	Send form for generate speech	Audio element is shown with the generated speech or alert if error is occurred	As expected
15	Direct back to Home page from Generate page	Directed back to Home Page	As expected

Figures 6.3 Some screenshot from the Mimic application scenarios.

The dataset during the speech recognition testing on identify process is 1000 male speech data on no noisy background. Corrected result shows from the more than 75% of model accuracy. The following tests table results below and figures that shows some screenshot of identify process results:

Table 6.4 Tester 1 speech recognition on identify process results.

No	**Phoneme**	**Noisy Background**	**Semi Noisy Background**	**No Noisy Background**	**Result**
1	a				/3
2	i				/3
3	t				/3
3	na				/3
4	ma				/3
5	mu				/3
6	di				/3
7	ri				/3
8	ku				/3
9	kan				/3
10	Unknown 1				/3
11	Unknown 2				/3
12	Unknown 3				/3

Figures 6.4 Some screenshot from the Tester 1 results.

Table 6.5 Tester 2 speech recognition on identify process results.

No	Phoneme	Noisy Background	Semi Noisy Background	No Noisy Background	Result
1	a				/3
2	i				/3
3	t				/3
3	na				/3
4	ma				/3
5	mu				/3
6	di				/3
7	ri				/3
8	ku				/3
9	kan				/3
10	Unknown 1				/3
11	Unknown 2				/3
12	Unknown 3				/3

Figures 6.5 Some screenshot from the Tester 2 results.

Table 6.6 Tester 3 speech recognition on identify process results.

No	Phoneme	Noisy Background	Semi Noisy Background	No Noisy Background	Result
1	a				/3
2	i				/3
3	t				/3
3	na				/3
4	ma				/3
5	mu				/3
6	di				/3
7	ri				/3
8	ku				/3
9	kan				/3
10	Unknown 1				/3
11	Unknown 2				/3
12	Unknown 3				/3

Figures 6.6 Some screenshot from the Tester 3 results.

CHAPTER VII

CONCLUSIONS AND FUTURE WORK

7.1 Conclusion

The following list sums up that the application is achieved based on this research objective:

1. This application enables to recognize speech in Bahasa Indonesia speech from record audio.
2. This application enables to generate speech in Bahasa Indonesia speech from text.
3. This application enables to mimic speech through the website.
4. This application enables to collect speech data through the website.
5. This application enables to train the collected speech data through the console.

7.2 Future Work

The following suggestion for further development and improvements of the research or application:

1. User Interface

   Improvement on UI will always help the user experience. With some colourful theme, clear button or inputs, the application will comfortable to be used.

2. Speech Recognition

Improvement on machine learning model can be made. When there is no right or wrong in modelling the machine learning model, there is always optimal model to get the best accuracy. A research to find the optimal model or a research developing application with different machine learning model are big improvement in the application or even in Speech Recognition itself.

3. Speech Synthesis

Improvement on Speech Synthesis is when generating the speech. By removing some of silence or unused part of the speech and also reducing background will make generated speech more fluently and good to hear.

REFERENCES

[1] Hurwitz, J., & Kirsch, D. (2018). *Machine Learning For Dummies®, IBM Limited Edition.* New Jersey: John Wiley & Sons, Inc.

[2] Nilsson, N. J., & Laboratory, R. (2005). *INTRODUCTION TO MACHINE LEARNING.* California: Nils J. Nilsson.

[3] dictionary.cambridge.org. (2018). *MACHINE*. Retrieved from Cambridge English Dictionary: https://dictionary.cambridge.org/dictionary/english/machine

[4] dictionary.cambridge.org. (2018). *LEARNING*. Retrieved from Cambridge English Dictionary: https://dictionary.cambridge.org/dictionary/english/learning

[5] Society, T. R. (2017). *Machine learning: the power and promise.* London: The Royal Society.

[6] edureka.com. (2018, October 18). *What is Machine Learning?* Retrieved from edureka: https://www.edureka.co/blog/what-is-machine-learning/

[7] Geitgey, A. (2016, January 3). *Machine Learning is Fun! Part 2*. Retrieved from Medium: https://medium.com/@ageitgey/machine-learning-is-fun-part-2-a26a10b68df3

[8] Magdi Zakaria, M. A.-S. (2014). Artificial Neural Network : A Brief Overview. *Int. Journal of Engineering Research and Applications*, 6.

[9] A.D.Dongare, R. A. (2012). Introduction to Artificial Neural Network. *International Journal of Engineering and Innovative Technology*, 6.

[10] courses.lumenlearning.com. (n.d.). *Introduction to Language*. Retrieved from lumen: https://courses.lumenlearning.com/boundless-psychology/chapter/introduction-to-language/

[11] readingdoctor.com.au. (2016). *Phonemes, Graphemes and Letters: The Word Burger*. Retrieved from Reading Doctor: http://www.readingdoctor.com.au/phonemes-graphemes-letters-word-burger/

[12] Yanti, N. T. (n.d.). FONEM BAHASA INDONESIA. *Academia.edu*, 10.

[13] dictionary.cambridge.org. (2018). *VOWEL*. Retrieved from Cambridge English Dictionary: https://dictionary.cambridge.org/dictionary/english/vowel

[14] puebi.readthedocs.io. (n.d.). *Huruf Vokal*. Retrieved from PUEBI Daring: https://puebi.readthedocs.io/en/latest/huruf/huruf-vokal/

[15] dictionary.cambridge.org. (2018). *DIPHTHONG*. Retrieved from Cambridge English Dictionary: https://dictionary.cambridge.org/dictionary/english/diphthong

[16] puebi.readthedocs.io. (n.d.). *Huruf Diftong*. Retrieved from PUEBI Daring: https://puebi.readthedocs.io/en/latest/huruf/huruf-diftong/

[17] dictionary.cambridge.org. (2018). *CONSONANT*. Retrieved from Cambridge English Dictionary: https://dictionary.cambridge.org/dictionary/english/consonant

[18] puebi.readthedocs.io. (n.d.). *Huruf Konsonan*. Retrieved from PUEBI Daring: https://puebi.readthedocs.io/en/latest/huruf/huruf-konsonan/

[19] dictionary.cambridge.org. (2018). *CLUSTER*. Retrieved from Cambridge English Dictionary: https://dictionary.cambridge.org/dictionary/english/cluster

[20] puebi.readthedocs.io. (n.d.). *Gabungan Huruf Konsonan*. Retrieved from PUEBI Daring: https://puebi.readthedocs.io/en/latest/huruf/gabungan-huruf-konsonan/

[21] Dave, B., & Pipalia, P. D. (2014). SPEECH RECOGNITION: A REVIEW. *International Journal of Advance Engineering and Research*, 7.

[22] M.A.Anusuya, & S.K.Katti. (2009). Speech Recognition by Machine: A Review. *International Journal of Computer Science and Information Security*, 25.

[23] Geitgey, A. (2016, December 24). *Machine Learning is Fun Part 6*. Retrieved from Medium: https://medium.com/@ageitgey/machine-learning-is-fun-part-6-how-to-do-speech-recognition-with-deep-learning-28293c162f7a

[24] en.wikipedia.org. (2018, October). *Nyquist–Shannon sampling theorem*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Nyquist%E2%80%93Shannon_sampling_theorem

[25] Olshausen, B. A. (2000). Aliasing. *Redwood Center for Theoretical Neuroscience*, 6.

[26] Rabiner, L. R., & Schafer, R. W. (1978). *Digital Processing of Speech Signals.* Prentice Hall: New Jersey.

[27] Hande, S. S. (2014). A Review on Speech Synthesis an Artificial Voice Production. *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, 8.

[28] lyrebird.ai. (2018). *Our Voice Products*. Retrieved from Lyrebird: https://lyrebird.ai/products

[29] translate.google.com. (2018). *Languages*. Retrieved from Google Translate: https://translate.google.com/intl/en/about/languages/

[30] w3school.com. (2018). *Node.js Introduction*. Retrieved from W3School: https://www.w3schools.com/nodejs/nodejs_intro.asp

[31] mongodb.com. (2018). *NoSQL Databases Explained*. Retrieved from MongoDb: https://www.mongodb.com/nosql-explained