



INDIAN INSTITUTE OF TECHNOLOGY, GUWAHATI

Department of Computer Science and Engineering

Project Report on

BILL GENERATOR

Based on Speech Recognition System

Submitted to:

Prof. P. K. Das

Submitted by:

Lijo Raju (224101033)

Binayak Behara (224101014)

Sri Harsha R (224101048)

For course fulfilment of CS566: Speech Processing

ACKNOWLEDGEMENT

This project is being submitted as a requirement for course fulfilment of CS566 - Speech Processing. It is a pleasure to acknowledge our sense of gratitude to Prof. P.K. Das who guided us throughout the project work. His timely guidance and suggestions were encouraging. We thank to the Teaching Assistants who were always helpful in clearing doubts. Finally, we thank to our classmates for the support.

1. Lijo Raju (224101033)
2. Binayak Behara (224101014)
3. Sri Harsha R (22410148)

Contents

| | | |
|----------|--------------------------------------|-----------|
| 1 | Abstract | 4 |
| 2 | Introduction | 4 |
| 2.1 | What is Speech Recognition | 4 |
| 2.2 | Our Project | 4 |
| 2.3 | Future improvements | 4 |
| 3 | Experimental Setup | 5 |
| 4 | Proposed Techniques | 5 |
| 4.1 | Flowchart | 5 |
| 4.2 | Model description | 5 |
| 5 | Result | 11 |
| 5.1 | Application Screen | 11 |
| 5.2 | Live Testing | 11 |
| 5.3 | Live Training | 11 |
| 6 | Source Code | 11 |
| 6.1 | main.dart | 12 |
| 6.2 | util.dart | 22 |

List of Figures

| | | |
|---|------------------------------------|----|
| 1 | Flowchart of the project | 6 |
| 2 | Application Screen | 7 |
| 3 | Final Bill | 8 |
| 4 | Live Training | 9 |
| 5 | Add Existing User | 9 |
| 6 | Add New User | 10 |
| 7 | Add New Item | 10 |
| 8 | Live Training Complete | 11 |

1 Abstract

Speech Based Bill Generator is a speech based solution to generate a bill for the speech input given by the user. The Application asks the user for the item name and the quantity. After the user input, the application asks the user if the order is completed. Based on user input more items are added accordingly. Ultimately after the user is provided with the complete order, the application displays the bill. Initially it is developed for simple words and limited quantity. But it can be expanded further for a bigger area of words.

The backend of the project is developed using C++/C in Visual Studio IDE, the frontend application and the web server is developed using Node.js and Flutter. It is build upon the core concept of Hidden Markov Model to store the properties of the speech sample and compare the new sample with these properties to detect which word has been spoken.

2 Introduction

2.1 What is Speech Recognition

Speech Recognition is a technique which is quite popular now-a-days. When we speak into a microphone which is connected to the computer/mobile, it converts it to a text file which contains some amplitude values. Those values are basically the deviation of the speech signal from X-axis. Then we can use this file, do some calculations which can detect which word has been spoken and then further steps can be taken as per the requirement. One such application is Alexa.

2.2 Our Project

This project uses a similar technique. There is a set of predefined words - Maggie, Vada, Tea, Yes, No, One, Two, Three, Four, Five, Six and Seven. We can run the project and speak any of these words from the menu and do the same for the quantity. After the user input is complete, the application displays the bill containing the respective items, their quantities and the final total amount to be paid. We can also train these words for new speakers.

2.3 Future improvements

Since the backend of the project is developed using C/C++, whereas web server and the application is developed using Node.js and Flutter. Future improvements include connecting the application to the live training module in the backend. This will enable the adding new speaker feature to be dynamic.

3 Experimental Setup

Basic requirements for this project are as follows-

- Windows OS
- Microsoft Visual Studio 2010
- C++11 integrated with VS2010
- Recording Module
- A good microphone
- A Smartphone

4 Proposed Techniques

4.1 Flowchart

Figure 1 is a flowchart of the project. Those steps can be followed for successful execution of the project.

4.2 Model description

We are using the famous Hidden Markov Model to store the speech properties. Hidden Markov Model is a probabilistic model which is used to explain or derive the probabilistic characteristic of any random process. When we apply log function to the spectral representation of the speech during reverse fourier transform, it is converted to cepstrum whose coefficients are steady because of application of log function and it represents the speech in a nice manner. This representation can be used as the speech property. We take all such cepstral coefficients and build a codebook which helps in generating the observation sequences. Codebook contains 10 speech samples for each word.

We use feed-forward model for modelling speech samples. While speaking, we speak a word from start to end. So, there is no need of backward movement. Also, the stress on current phoneme is more than moving to the next phoneme. Hence we use feed-forward model. Then while testing, we score each model using the forward process and pick the word with highest score as the result.

Since, speech signals depend a lot on the environment, live testing might not be very good. But, if we train the model live and test it immediately, then we get significantly better accuracy.

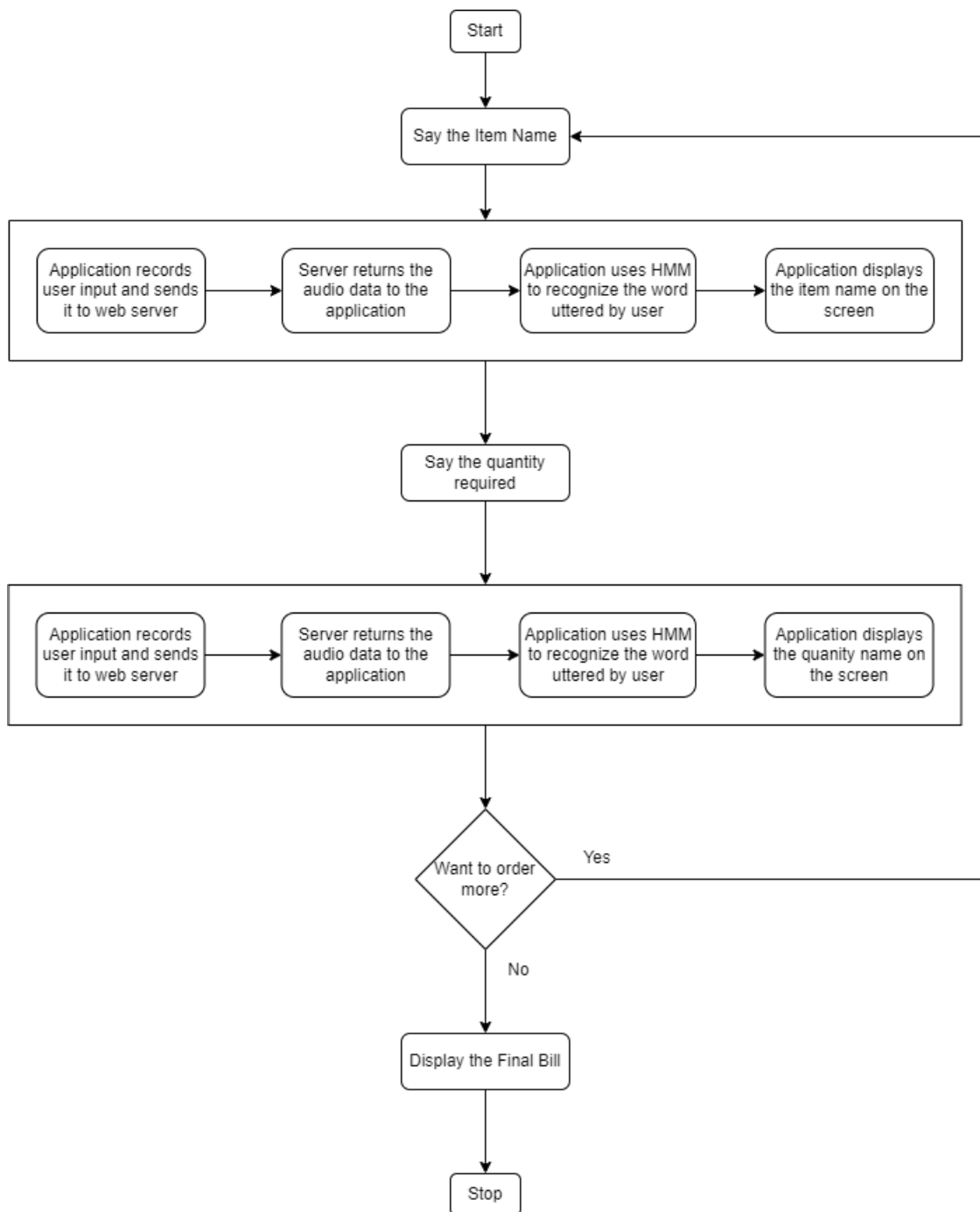


Figure 1: Flowchart of the project

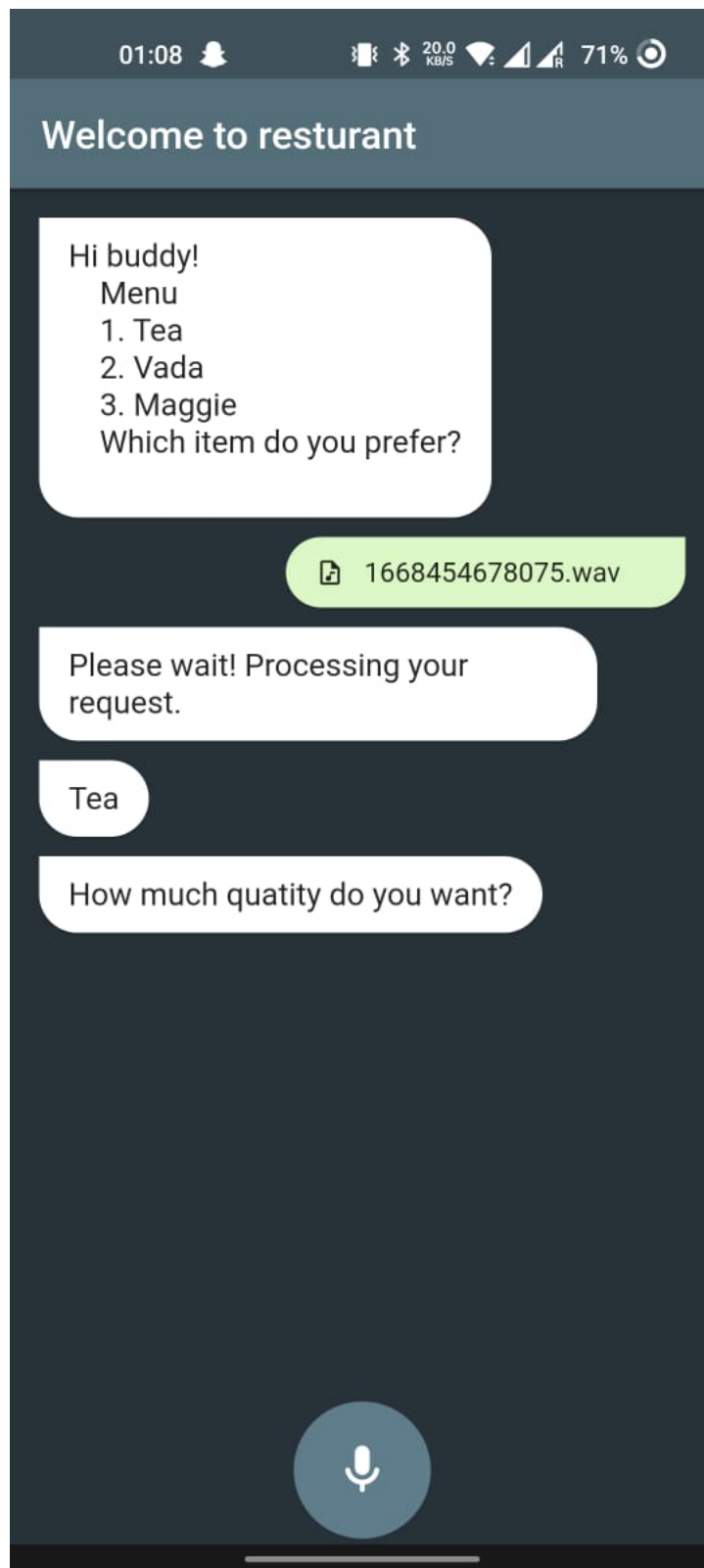


Figure 2: Application Screen

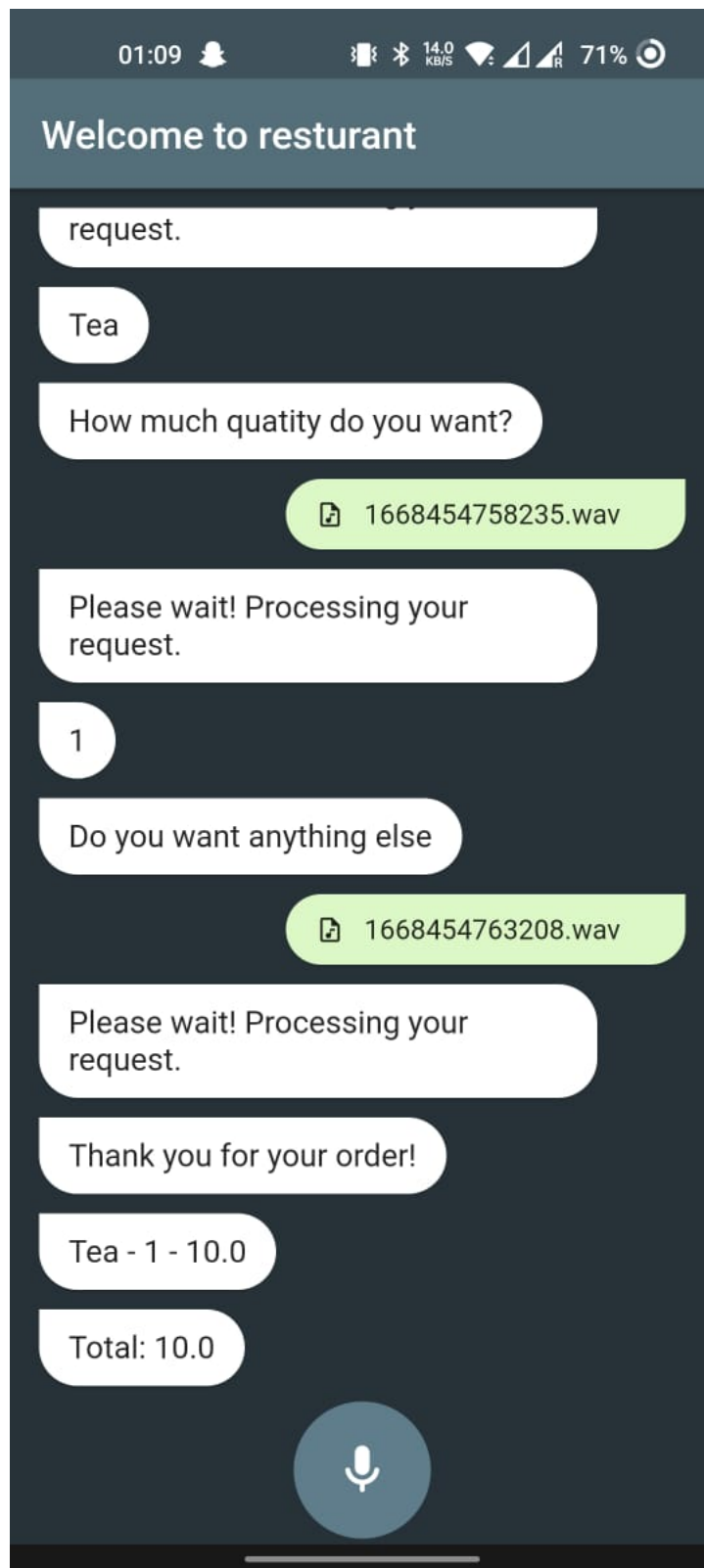


Figure 3: Final Bill

```
BillGenerator_Training

***BILL GENERATOR TRAINING***

MENU

1. Add New User
2. Add New Item
3. Exit

Your Choice? 1

What's the new username? Harsha

New folder created for Harsha under /InputData

Enter the item you want to add
```

Figure 4: Live Training

```
BillGenerator_Training

***BILL GENERATOR TRAINING***

MENU

1. Add New User
2. Add New Item
3. Exit

Your Choice? 1

What's the new username? Lijo

User already exist.
Press any key to continue . . .
```

Figure 5: Add Existing User

```
BillGenerator_Training

***BILL GENERATOR TRAINING***

MENU

1. Add New User
2. Add New Item
3. Exit

Your Choice? 1

What's the new username? Harsha

New folder created for Harsha under /InputData

Enter the item you want to add Coffee

Record your voice and save file in .txt format( itemname_utteranceno ) under InputData/Harsha
Press any key to continue . . .
```

Figure 6: Add New User

```
BillGenerator_Training

***BILL GENERATOR TRAINING***

MENU

1. Add New User
2. Add New Item
3. Exit

Your Choice? 1

What's the new username? Harsha

New folder created for Harsha under /InputData

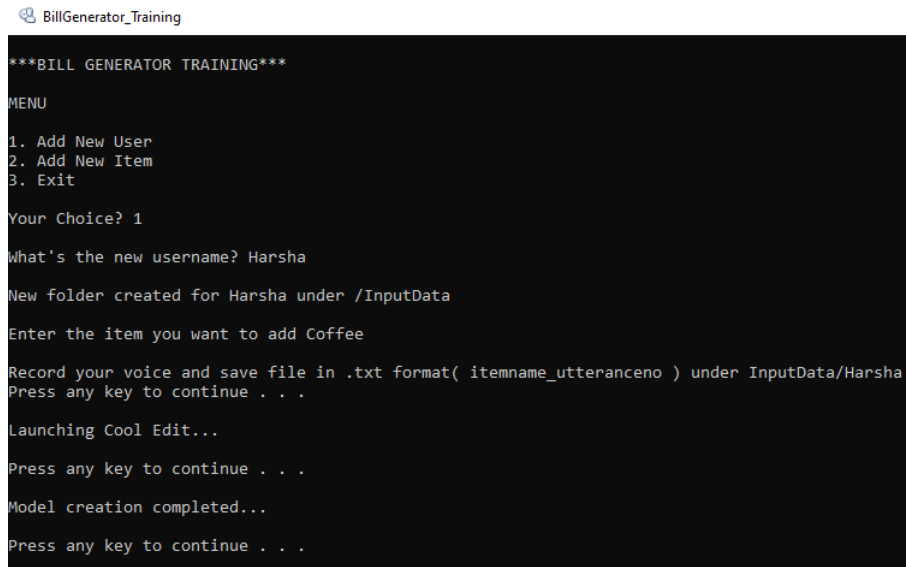
Enter the item you want to add Coffee

Record your voice and save file in .txt format( itemname_utteranceno ) under InputData/Harsha
Press any key to continue . . .

Launching Cool Edit...

Press any key to continue . . .
```

Figure 7: Add New Item



```
BillGenerator_Training

***BILL GENERATOR TRAINING***

MENU

1. Add New User
2. Add New Item
3. Exit

Your Choice? 1

What's the new username? Harsha

New folder created for Harsha under /InputData

Enter the item you want to add Coffee

Record your voice and save file in .txt format( itemname_utteranceno ) under InputData/Harsha
Press any key to continue . . .

Launching Cool Edit...

Press any key to continue . . .

Model creation completed...

Press any key to continue . . .
```

Figure 8: Live Training Complete

5 Result

5.1 Application Screen

Figure 2 shows the initial screen of the project. The user is asked to say the item and the quantity. If user input is complete then Figure 3 will be displayed.

5.2 Live Testing

For live testing, the application asks the user a question which is showed in Figure 7. A word can be spoken and recorded for testing. The corresponding Item will be displayed e.g. Figure 8.

5.3 Live Training

For live training, the user runs the executable file. Then Figure 4 will be displayed. A username can be entered. It will detect whether the username is already present i.e. Figure 5 or new i.e. Figure 6. If the username is already present, then recording is not allowed, otherwise record thirty times. For recording a new item, enter the item name as shown in the Figure 7 will be displayed. After recording, the final output is as shown in Figure 8.

6 Source Code

A few files are too large. Hence the entire code is not added here. A few short files are added.

6.1 main.dart

```
import 'dart:async';
import 'package:audio_session/audio_session.dart';
import 'package:http/http.dart' as http;
import 'package:http_parser/http_parser.dart';
import 'package:flutter/material.dart';
import 'package:flutter_sound/flutter_sound.dart';
import 'package:path_provider/path_provider.dart';
import 'package:permission_handler/permission_handler.dart';
import 'package:provider/provider.dart';
import 'package:vrecorder/message.dart';
import 'package:vrecorder/message_list.dart';
import 'package:vrecorder/util-functions.dart';
import "audiowaveform_response.dart";

/*
 * This is an example showing how to record to a Dart Stream.
 * It writes all the recorded data from a Stream to a File, which is completely stupid:
 * if an App wants to record something to a File, it must not use Streams.
 *
 * The real interest of recording to a Stream is for example to feed a
 * Speech-to-Text engine, or for processing the Live data in Dart in real time.
 *
 */

///'
enum QuestionType { item, quantity, other }

const int tSampleRate = 16000;
typedef _Fn = void Function();

/// Example app.
class RecordToStreamExample extends StatefulWidget {
  final ScrollController scrollController;
```

```

const RecordToStreamExample({
  super.key,
  required this.scrollController,
});

@override
RecordToStreamExampleState createState() => RecordToStreamExampleState();
}

class RecordToStreamExampleState extends State<RecordToStreamExample> {
  final QuestionType _qMode = QuestionType.quantity;
  FlutterSoundPlayer? _mPlayer = FlutterSoundPlayer();
  FlutterSoundRecorder? _mRecorder = FlutterSoundRecorder();
  bool _mPlayerIsInitd = false;
  bool _mRecorderIsInitd = false;
  bool _mplaybackReady = false;
  String? _mPath;
  String? filename;
  StreamSubscription? _mRecordingDataSubscription;
  List<Map<String, String>> questions = [
    {
      "question": "Which item do you prefer?",
      "type": "item",
    },
    {
      "question": "How much quatity do you want?",
      "type": "quantity",
    },
    {
      "question": "Do you want anything else",
      "type": "other",
    },
  ];

```

```

int questionIndex = 0;

scrollToBottom(ScrollController scrollController, {double value = 0.0}) {
  scrollController.animateTo(
    scrollController.position.maxScrollExtent + 70 + value,
    duration: const Duration(milliseconds: 200),
    curve: Curves.easeOut,
  );
}

Future<void> _openRecorder() async {
  var status = await Permission.microphone.request();
  if (status != PermissionStatus.granted) {
    throw RecordingPermissionException('Microphone permission not granted');
  }
  await _mRecorder!.openRecorder();

  final session = await AudioSession.instance;
  await session.configure(AudioSessionConfiguration(
    avAudioSessionCategory: AVAudioSessionCategory.playAndRecord,
    avAudioSessionCategoryOptions:
      AVAudioSessionCategoryOptions.allowBluetooth |
      AVAudioSessionCategoryOptions.defaultToSpeaker,
    avAudioSessionMode: AVAudioSessionMode.spokenAudio,
    avAudioSessionRouteSharingPolicy:
      AVAudioSessionRouteSharingPolicy.defaultPolicy,
    avAudioSessionSetActiveOptions: AVAudioSessionSetActiveOptions.none,
    androidAudioAttributes: const AndroidAudioAttributes(
      contentType: AndroidAudioContentType.speech,
      flags: AndroidAudioFlags.none,
      usage: AndroidAudioUsage.voiceCommunication,
    ),
    androidAudioFocusGainType: AndroidAudioFocusGainType.gain,
    androidWillPauseWhenDucked: true,
  );
}

```

```

));

setState(() {
  _mRecorderIsInitiated = true;
});
}

@override
void initState() {
  super.initState();

  _mPlayer!.openPlayer().then((value) {
    setState(() {
      _mPlayerIsInitiated = true;
    });
  });
  _openRecorder();
}

@override
void dispose() {
  _mPlayer!.closePlayer();
  _mPlayer = null;

  stopRecorder();
  _mRecorder!.closeRecorder();
  _mRecorder = null;
  super.dispose();
}

Future<String> createFile() async {
  var tempDir = await getExternalStorageDirectory();
  var timestamp = '${DateTime.now().millisecondsSinceEpoch}.wav';
  setState(() {

```



```

        filename = timestamp;
        _mPath = '${tempDir!.path}/${timestamp}';
    });
    return _mPath!;
}

```

```

Future<void> record() async {
    assert(!_mRecorderIsInitied && !_mPlayer!.isStopped);
    var fileName = await createFile();

    await _mRecorder!.startRecorder(
        toFile: fileName,
        codec: Codec.pcm16WAV,
        numChannels: 1,
        sampleRate: tSampleRate,
    );
    setState(() {});
}

```

```

void addBotMessage(int index) {
    Provider.of<MessageList>(context, listen: false).addMessage(
        Message(
            message: questions[index]["question"]!,
            sender: "bot",
        ),
    );
    scrollToBottom(widget.scrollController);
    return;
}

```

```

sendRequest(String audioPath) async {
    var postUri = Uri.parse("https://4670-14-139-196-15.in.ngrok.io/audiowave");
    var request = http.MultipartRequest("POST", postUri);
    try {

```

```

request.files.add(await http.MultipartFile.fromPath(
    'file',
    audioPath,
    contentType: MediaType('audio', 'mp3'),
));

print('Porcessing!');
final response = await request.send();

if (response.statusCode == 201) {
    print("Request Sent!");
    final respStr = await response.stream.bytesToString();
    AudiowaveFormResponse audiowaveFormResponse =
        audiowaveFormResponseFromJson(respStr);

    var res = calculateResult(audiowaveFormResponse.data);
    var item = findAns(res, questionIndex);
    final provider = Provider.of<MessageList>(context, listen: false);
    if (questionIndex == 1) {
        var lastItem = provider.posts[provider.posts.length - 4];
        debugPrint(lastItem.toString());
        provider.addOrder(lastItem.message, int.parse(item));
    }

    provider.addMessage(Message(message: item, sender: 'Bot'));
    scrollToBottom(widget.scrollController);
    if (questionIndex == 2) {
        provider.addMessage(
            Message(
                message: 'Thank you for your order!',
                sender: 'bot',
            ),
        );
        var orders = provider.order;
    }
}

```

```

double total = 0;
orders.forEach((key, value) {
  if (value.quantity != 0) {
    total += value.price * value.quantity;
    String itemString =
      '$key - ${value.quantity} - ${value.quantity * value.price}';

    provider.sendMessage(
      Message(
        message: itemString,
        sender: 'bot',
      ),
    );
  }
  scrollToBottom(widget.scrollController);
});
provider.sendMessage(
  Message(
    message: 'Total: $total',
    sender: 'bot',
  ),
);
scrollToBottom(widget.scrollController, value: 400);
}

debugPrint(audiowaveFormResponse.data.toString());
questionIndex = (questionIndex + 1) % 3;
addBotMessage(questionIndex);
} else {
  debugPrint("Request Failed!");
  Provider.of<MessageList>(context, listen: false).addMessage(
    Message(message: 'Request failed. Try again!', sender: 'Bot'));
  scrollToBottom(widget.scrollController);
}

```

```

    } catch (e) {
        Provider.of<MessageList>(context, listen: false).addMessage(
            Message(message: 'Request failed. Try again!', sender: 'Bot'));
        scrollToBottom(widget.scrollController);
    }
}

String checkAnswer(int n) {
    return n == 3 ? "Yes" : "No";
}

String findQuantity(int n) {
    return (n - 4).toString();
}

String findItem(int n) {
    switch (n) {
        case 0:
            return 'Tea';
        case 1:
            return 'Vada';
        case 2:
            return 'Maggie';
        default:
            return 'Other';
    }
}

String findAns(List<double> p, int questionIndex) {
    if (questionIndex == 0) {
        var n = itemTest(p);
        return findItem(n);
    } else if (questionIndex == 1) {
        var n = quantityTest(p);

```

```

        return findQuantity(n);
    } else {
        var n = yesnoTest(p);
        return checkAnswer(n);
    }
}

```

```

List<double> calculateResult(List<double> data) {
    // execute util functions
    var amp = split(data);
    var obs = findObsSeq(amp);
    var prob = test(obs);
    debugPrint(prob.toString());
    return prob;
}

```

```

Future<void> stopRecorder() async {
    await _mRecorder!.stopRecorder();
    if (_mRecordingDataSubscription != null) {
        await _mRecordingDataSubscription!.cancel();
        _mRecordingDataSubscription = null;
    }
    _mplaybackReady = true;
}

```

```

Provider.of<MessageList>(context, listen: false)
    .addMessage(Message(message: filename!, sender: 'Me'));

```

```

Future.delayed(const Duration(milliseconds: 500), () {
    Provider.of<MessageList>(context, listen: false).addMessage(Message(
        message: 'Please wait! Processing your request.', sender: 'bot'));
    scrollToBottom(widget.scrollController);
});

```

```

sendRequest(_mPath!);

```

```

}

_Fn? getRecorderFn() {
  if (!_mRecorderIsInitd || !_mPlayer!.isStopped) {
    return null;
  }
  return _mRecorder!.isStopped
    ? record
    : () {
        stopRecorder().then((value) => setState(() {}));
      };
}

@override
Widget build(BuildContext context) {
  return Container(
    margin: const EdgeInsets.all(3),
    padding: const EdgeInsets.all(3),
    height: 70,
    width: 70,
    alignment: Alignment.center,
    decoration: BoxDecoration(
      color: Colors.blueGrey.shade500,
      borderRadius: BorderRadius.circular(50),
    ),
    child: Row(mainAxisAlignment: MainAxisAlignment.center, children: [
      IconButton(
        onPressed: getRecorderFn(),
        color: Colors.white,
        icon: Icon(
          _mRecorder!.isRecording
            ? Icons.stop_circle_rounded
            : Icons.mic_rounded,
          size: 30,

```

```

        ),
    ),
  ]),
);
}
}

```

6.2 util.dart

```

import 'dart:math' as MATH;
import 'dataset.dart' as dt;

List<double> hammingWindow(List<double> amp) {
  int N = amp.length;
  List<double> window = List.filled(N, 0);
  for (int n = 0; n < N; n++) {
    window[n] = 0.54 - 0.46 * MATH.cos((2 * MATH.pi * n) / (N - 1));
  }
  return window;
}

// List<double> durbinAlgo(List<double> amp) {
//   List<double> r = List.filled(13, 0);

//   for (int l = 0; l < 13; l++) {
//     for (int p = 0; p < 320 - l; p++) {
//       r[l] = r[l] + ((amp[p]) * amp[p + l]);
//     }
//   }

//   double sum = 0;

//   List<double> e = List.filled(13, 0);
//   List<double> k = List.filled(13, 0);
//   List<List<double>> alpha = List.filled(13, List.filled(13, 0));

```

```

//    e[0] = r[0];

//    for (int i = 1; i < 13; i++) {
//        for (int j = 1; j <= i - 1; j++) {
//            sum = sum + alpha[i - 1][j] * r[i - j];
//        }
//        if (e[i - 1] != 0){
//            k[i] = (r[i] - sum) / e[i - 1];
//        }
//        else k[i] = 0;

//        alpha[i][i] = k[i];

//        for (int j = 1; j <= i - 1; j++) {
//            alpha[i][j] = alpha[i - 1][j] - (k[i] * alpha[i - 1][i - j]);
//        }

//        e[i] = (1.0 - (k[i] * k[i])) * e[i - 1];

//        sum = 0;
//    }

//    List<double> res = List.filled(13, 0);

//    for (int x = 1; x < 13; x++) {
//        res[x] = alpha[12][x];
//    }

//    return res;
// }

List<double> durbinAlgo(List<double> amp) {
    List<double> E = List.filled(13, 0);
    List<double> k = List.filled(13, 0);

```



```

List<double> alptemp = List.filled(13, 0);
List<double> A = List.filled(13, 0);
List<double> R = List.filled(13, 0);

for (int l = 0; l < 13; l++) {
    for (int p = 0; p < 320 - l; p++) {
        R[l] = R[l] + ((amp[p]) * amp[p + l]);
    }
}

for (int i = 0; i <= 12; i++) {
    A[i] = 0;
    alptemp[i] = 0;
}

E[0] = R[0];
for (int i = 1; i <= 12; i++) {
    double summation = 0;
    for (int j = 1; j <= i - 1; j++) {
        alptemp[j] = A[j];
        summation += alptemp[j] * R[i - j];
    }
    k[i] = (R[i] - summation) / E[i - 1];

    A[i] = k[i];
    for (int j = 1; j <= i - 1; j++) {
        A[j] = alptemp[j] - k[i] * alptemp[i - j];
    }
    E[i] = (1 - k[i] * k[i]) * E[i - 1];
}
return A;
}

List<double> calculateCepstal(List<double> a) {

```

```

double t, sw = 0;
List<double> c = List.filled(13, 0);

for (int m = 1; m < 13; m++) {
    c[m] = a[m];
    for (int j = 1; j <= m - 1; j++) {
        t = j / m;

        c[m] = c[m] + t * c[j] * a[m - j];
    }
}

// apply sine window
for (int m = 1; m < 13; m++) {
    sw = 1 + (6.0 * MATH.sin((MATH.pi * m) / 12.0));
    c[m] = c[m] * sw;
}

return c;
}

//@brief - calculate tokura distance with codebook
//@parma - a - vector under consideration
List<double> tokuraDist(List<double> c) {
    double t;

    List<double> tokuraWt = [
        1.0,
        3.0,
        7.0,
        13.0,
        19.0,
        22.0,
        25.0,

```

```

    33.0,
    42.0,
    50.0,
    56.0,
    61.0
];

List<double> distance = List.filled(32, 0);

for (int i = 0; i < 32; i++) {
    for (int j = 0; j < 12; j++) {
        t = (c[j] - dt.centroid[i][j]) * (c[j] - dt.centroid[i][j]);
        distance[i] = distance[i] + tokuraWt[j] * t;
    }
}

return distance;
}

List<int> findObsSeq(List<double> amp) {
    final double max = amp.reduce(MATH.max);

    final double nFactor = 5000 / max;

    List<int> obs = List.of([], growable: true);
    obs.add(0);
    for (int i = 0; i < amp.length; i += 80) {
        if (i + 320 > amp.length) return obs;

        List<double> subList = amp.sublist(i, i + 320);
        subList = subList.map((double x) => (x * nFactor)).toList();

        // hamming window
        //subList = hammingWindow(subList);

```

```

// find durbinAlgo res

List<double> a = durbinAlgo(subList);
// find cepstal

List<double> cepstal = calculateCepstal(a);
// find takura distance

List<double> distance = tokuraDist(cepstal);
// find min distance
obs.add(distance.indexOf(distance.reduce(MATH.min)) + 1);
}

return obs;
}

double forwardProcedure(List<double> pi, List<List<double>> a,
    List<List<double>> b, List<int> obs, int T) {
    double sum = 0;

    List<List<double>> alpha = List.filled(T + 1, List.filled(6, 0));
    for (int i = 1; i <= 5; i++) {
        alpha[1][i] = pi[i] * b[i][obs[1]];
    }

    for (int t = 1; t <= T - 1; t++) {
        for (int j = 1; j <= 5; j++) {
            for (int i = 1; i <= 5; i++) {
                sum += (alpha[t][i] * a[i][j]);
            }

            alpha[t + 1][j] = sum * b[j][obs[t + 1]];
            sum = 0;
        }
    }
}

```

```

    }
}

sum = 0;

for (int m = 1; m <= 5; m++) {
    sum += alpha[T][m];
}
return sum;
}

List<double> test(List<int> obs) {
    List<double> res = List.filled(12, 0);

    for (int i = 0; i <= 11; i++) {
        res[i] = forwardProcedure(
            dt.pi, dt.matrixA[i], dt.matrixB[i], obs, obs.length - 1);
    }

    return res;
}

int itemTest(List<double> p) {
    int max = 0;
    double maxVal = 0;

    for (int i = 0; i < 3; i++) {
        if (p[i] > maxVal) {
            maxVal = p[i];
            max = i;
        }
    }
    return max;
}

```

```

int quantityTest(List<double> p) {
    int max = 5;
    double maxVal = 0;

    for (int i = 5; i < p.length; i++) {
        if (p[i] > maxVal) {
            maxVal = p[i];
            max = i;
        }
    }
    return max;
}

int yesnoTest(List<double> p) {
    int max = 3;
    double maxVal = 0;

    for (int i = 3; i < 5; i++) {
        if (p[i] > maxVal) {
            maxVal = p[i];
            max = i;
        }
    }
    return max;
}

List<double> split(List<double> amp) {
    List<double> newAmp = List.of([], growable: true);
    int count = 0;
    int start = 0;

    for (int i = 0; i < amp.length; i++) {
        start++;
    }
}

```

```

        if (amp[i] > 400) break;
    }

    for (int i = start; i < amp.length; i++) {
        if (amp[i] > 100) {
            count = 0;
            newAmp.add(amp[i]);
        } else {
            newAmp.add(amp[i]);
            count++;
            if (count > 300) break;
        }
    }
    return newAmp;
}

```