3/28/2019

Speech Radar

Shah Ali 33455846

Supervised by Dr Jamie Ward

Computer Science

Goldsmiths University of London

Abstract

To do…..

Contents page

To do…

# Introduction

## Background

Since 1784, Speech recognition is something that was just a topic of talk. It wasn't until 1952 when a six-foot machine was created by Bell Labs, *capable of recognizing spoken digits with 90% accuracy* [3], but when uttered by its owner. The development would continue in 1962, where IBM created a machine the size of a shoebox that could *understand 16 English words* [3]. In 1971 a student of Carnegie Mellon University created the Harpy that could *comprehend 1011 words and some phrases* [3]. In 1986, IBM would create another ground-breaking machine that used the *Hidden Markov Model* [3] to recognise 20000 different words from various speakers and type them on paper. The list of inventions would go on with Google launching a voice search application in 2008, *bringing speech recognition to mobile devices* [3]. In 2011, Apple would announce Siri, *ushering in the age of the voice-enabled digital assistant* [3]*.*

At present, we are seeing digital assistants decentralise from smartphones and are seeing companies primarily focus on voice-activated home speakers that can query and control smart home devices. From a subjective point of view, these innovations appear to be an approach to accumulate billions of audio data from people that have different accents, so companies in the future can improve the detection rate for fluent and non-fluent English speakers. On the off chance that this improvement happens, we will see speech recognition being used for more advanced tasks, possibly in robotics.

## Purpose

In the area of speech recognition, it is said that Microsoft can now interpret *human speech with a 5.1% error rate* [1]. Google have enhanced its *accuracy by more than 20% in the past five years* [1]. And to date, Amazon's Alexa has been getting better at responding to users’ question. *Researchers asked the voice assistant 800 different queries* [2] in various categories. *On average, Alexa answered queries accurately 73% of the time, up 12 percentage points from 61%* [2]. These are little known facts of how top companies are taking speech recognition to a whole new level and how they've made it worth looking into and investing in for personal and business use. Given that the unexpected rise has happened so recently, this project will aim to contribute to the area by researching the tools and techniques that are being used to make it work so efficiently. The research will then be used to implement an Android application called Speech Radar, that will allow users to locate their phone through speech recognition. Suppose for instance an individual must rush to work and is unable to find their phone, but knows it’s located somewhere inside their room. Speech Radar can speed up the search time just by an utterance of a specific word from the individual. Once the phone detects the utterance, it will begin ringing at maximum sound level, which will help the individual locate their phone. This app can be of use to a great number of people as it is common for individuals nowadays to lose sight of their phone and to search everywhere for it.

## Aims and objectives

* To research on a feasible approach of creating a TensorFlow model for speech recognition by going through implementations of certain users public GitHub repositories
* To research on an appropriate dataset to use for training the model. Preferably, the dataset should contain audio files for each word in different accents, rather than phrases.
* To design our implementation in TensorFlow and Android Studio using pseudocode, flowcharts, wireframes and UML diagrams
* To implement our designs and test the speech recognitions detection rate with many random users. Gather the results for analysis, and use it to improve the application

## Section overview

**Chapter 2** - This chapter consists of the research for the project through literature reviews, in the attempt to critically evaluate other people’s work in the area of speech recognition

**Chapter 3** - This chapter contains the specification of the project. The requirements are stated precisely and in detail

**Chapter 4** - The design of the appearance and functionality is mentioned in this chapter.

**Chapter 5** – The implementation of the designs is critically discussed in this chapter

**Chapter 6** - This chapter specifies the results for user testing and analyse key areas that'll help improve the application. Software testing will also be included through black-box and white-box testing techniques. An evaluation will conclude this chapter, assessing the strengths/weaknesses of the application and what improvements could be made

**Chapter 7** - This chapter provides a conclusion for the report, with mentions of future work that can be done, successes/failures of the project and how it compared to what others have done.

**Chapter 8** – This chapter includes all bibliography’s and references used for the project

**Chapter 9** – Final chapter includes appendices to help explain all findings and analysis

# Literature review

This chapter introduces the research done in preparation for the next stages of the project. It analyses relevant published articles that can have an impact on the project. The research can help to find solutions for tasks that need to be prioritised and managed carefully during implementation phase. This will help gain knowledge and prepare for any problems that could take place. The focus area is in TensorFlow and Android Studio, where implementation of the project will take place.

## Speech Commands dataset

### Broad analysis

An article composed by Pete Warden examines his speech recognition implementation as a web app, using TensorFlow to create the model with the Speech Commands dataset. The main aim of the dataset is to supply a way in building and testing *small models that detect when a single word is spoken* [5] from a range of target words. This task is known as *keyword spotting* [5]. The final dataset consists of *105,829 utterances of 35 words* [5]. Each of the utterances in the Speech Commands dataset are stored as a *WAVE file- format* [5] lasting for one second. The sample data is *encoded as linear 16-bit single-channel PCM values* [5], at 16,000 Hz sampling rate. Over 2600 speakers are recorded, some with different accents.

The author of the article explains in a separate section how he wanted to have *a limited vocabulary* [5] of 10 words to ensure that the web app was lightweight and fast in making correct predictions. The choice of words were chosen carefully, so they do not have similar pronunciations and are easily detected. For example, the word 'three' is pronounced as 'TH R IY' and ‘tree’ similarly as 'T R IY'. This can cause the model to make wrong predictions, especially for someone with an accent. Thus, why the author decided to avoid these kinds of words.

Speech Commands dataset would be an incredible asset to the Speech Radar project. A limited vocabulary of code words would seem appropriate and feasible than having a large vocabulary. The article mentioned only 10 words were used. The Speech Radar app will attempt to use between 20 to 30 words from this dataset and ensure correct predictions are made most of the times. But for this to work, additional audio clips may need to be added to the dataset so the TensorFlow model can learn more.

### Additional information

What was incredibly helpful for the Speech Radar project was a brief mention in the article of how the TensorFlow model was loaded in the web app. The article states that it was exported as a protobuf file, which carries the graphs definition and weights of the model. This file can then be loaded in Android Studio via the assets folder. A visual flowchart inspired from this article can be seen in Appendix

## Convolutional neural network (CNN)

### Keyword spotting task

An article written by Tara N. Sainath and Carolina Parada mentions how they used a CNN to solve the keyword spotting problem on a mobile device. This is an issue that manages the recognition of keywords in articulations. Virtual assistants such as Google Assistant and Amazons Alexa use keyword spotting such as "Ok Google" or "Alexa" to wake up when their name is spoken. But for this to work properly especially with mobile devices, it must have a *small memory footprint and low computational power* [4]. Since this article has proved convincingly that CNN is the best choice for speech recognition tasks like keyword spotting, it will be analysed further to help in our implementation. Mention here cnn-trad-fpool3 in article.

### Wav to spectrogram

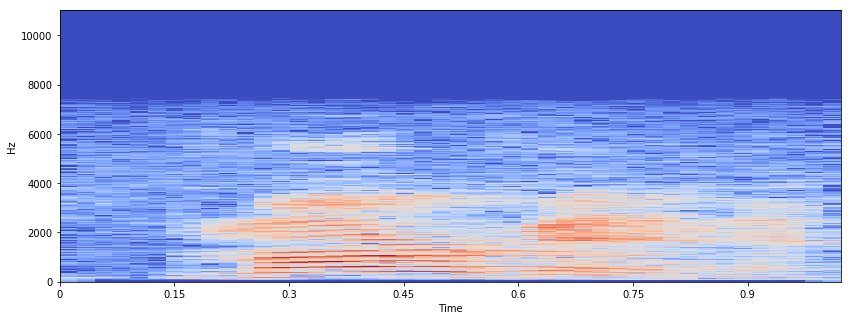
A WAV audio file can be represented as a spectrogram, which can be used as an input for a CNN. A spectrogram is an image of the signal strength/loudness of a signal over time at different frequencies present in a specific waveform. Not exclusively would one be able to see that there is energy at a certain hertz but can see how energy levels change after some time. Given that CNN's achieve great results in image classification problems, the information represented in the spectrogram is invaluable and can be extracted and learnt by the network. Figure 1 shows a spectrogram for the word ‘Marvin’. We see the frequency for every utterance needed to be made to say the word. Notice from 0.17 to 0.75 seconds, we see a high level of energy for the first part of the word pronounced as ‘M AA R’ and a low level of energy for the final part pronounced as ‘V IH N’ roughly from 0.79 to 0.95 seconds. This makes sense, since the loudness of voice for each part of the word alters i.e. goes from high to low. The low level of energy seen in 6000 hertz would most likely be the pronunciation of ‘AA’, as it is a high frequency vowel.

Figure 1: Marvin

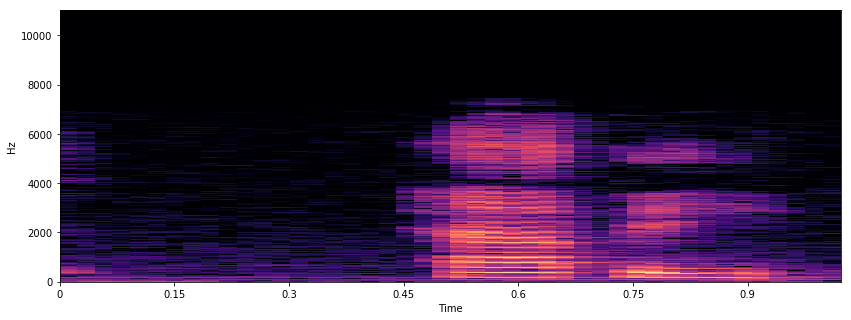
Figure 2 shows a spectrogram for another word ‘Happy’. Similarly, like figure 1 we see high levels of energy and frequency for the first part of the word pronounced as ‘HH AE’ and low levels for the final part pronounced as ‘P IY’. Notice how the frequency levels are much higher than figure 1. This tells us that the speech recognition system would be better at detecting the word ‘Happy’ than ‘Marvin’. Thus, it would be unsurprising for the results in the testing section to show different detection rates for each word.

Figure 2: happy

### Network architecture

A typical CNN architecture contains convolutional, pooling, normalization and fully connected layers. The table in figure 3 shows a basic model created for the keyword spotting task using the Speech Commands dataset (greater detail mentioned in section 2.1.2). We will call this model A. It is capable in recognizing three words i.e. bed, cat and happy, which is why the last layer contains only three neurons with Softmax as its activation function. Model A is small with only 82,368 parameters and will be interesting to see how it could be improved later.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Model | Layer | Neurons & rate | Activation function | Optimizer | Learning rate | Batch size | Parameters |
| A | Conv2d | 32 | Relu | Adadelta | 0.001 | 100 | 160 |
| Max\_pooling2d | - | - | 0 |
| Dropout | 0.2 | - | 0 |
| Flatten | - | - | 0 |
| Dense | 55 | Relu | 79,255 |
| Dropout | 0.2 |  | 0 |
| Dense | 50 | Relu | 2,800 |
| Dropout | 0.2 | - | 0 |
| Dense | 3 | Softmax | 153 |
| 82,368 |

Figure 3 model A

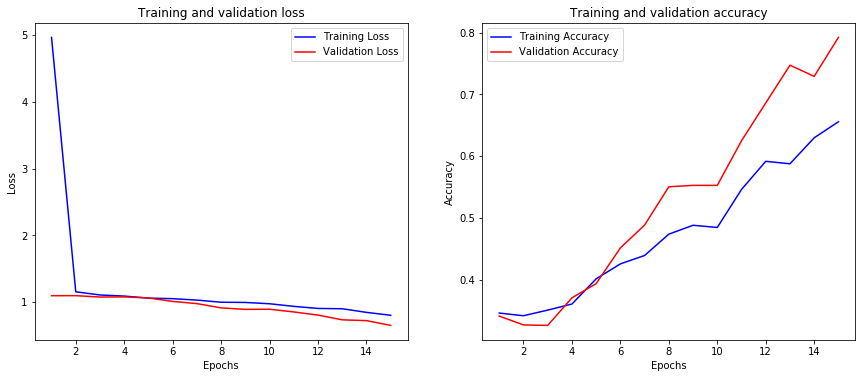
The graph in figure 4 shows the result for model A, where validation loss gradually reduces in each epoch. The loss for training and validation overall is very high and will certainly need improvement. A significant increase can be seen in the validation accuracy after 5 epochs, causing it to be much greater than the training accuracy. This means model A is underfitting. To avoid this, more parameters and layers can be added. Adjusting certain hyperparameters could also help (e.g. good learning rate, batch size etc.) Since a CNN architecture is being used, it might be best to add BatchNormalization layers to the model, as it works well with other regularization techniques such as dropout and L2. It can also speed up the learning process.

Figure 4: result

The table in figure 5 shows an extension from model A. we will call this model B. Everything highlighted in bold are changes/improvements made to the model. For example, the number of neurons has increased so that model B has more parameters to avoid underfitting. Another example would be the small reduction of the learning rate, which will hopefully prevent the performance from diverging and see the error rate reduce rapidly.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Model | Layer | Neurons & rate | Activation function | Optimizer | Learning rate | Batch size | Parameters |
| B | Conv2d | **80** | Relu | **RMSprop** | **0.0005** | **90** | 400 |
| **Batch\_normalization** | - | Relu | 320 |
| Max\_pooling2d | - | - | 0 |
| Dropout | 0.2 | - | 0 |
| Flatten | - | - | 0 |
| Dense | **85** | Relu | 306,085 |
| **Batch\_normalization** | - | - | 340 |
| Dropout | 0.2 | - | 0 |
| Dense | **90** | Relu | 7,740 |
| **Batch\_normalization** | - | - | 360 |
| dropout | 0.2 | - | 0 |
| dense | **3** | Softmax | 273 |
| 315,518 |

Figure 5 model B

The graph in figure 6 shows the result of model B. Lower training and validation loss can be seen compared to model A. High validation and training accuracy can also be seen. This tells us how well it can make predictions based on unseen data, which is vital for when the model is used in the Android app. Training the model in under 15 epochs was probably a good estimate, as we see the validation loss slowly diverging at the end.

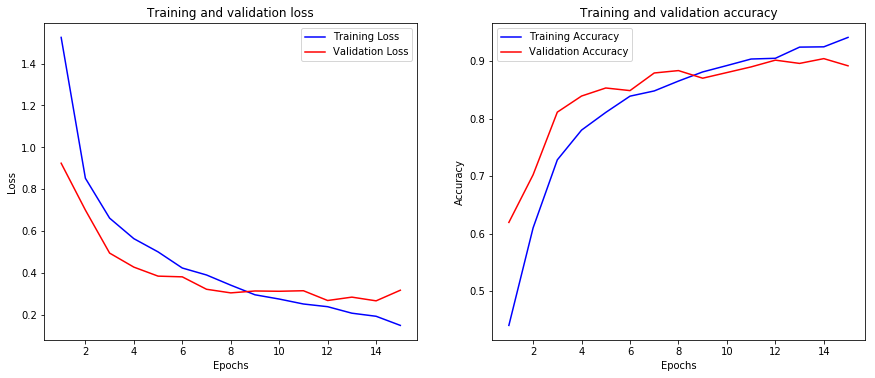


Figure 6 result

## PocketSphinx

### Brief

Another article written by Ajav Sharma and Rahul Bhalley explains their implementation of a *real-time speech recognition on portable devices* [6]. It mentions a library called PocketSphinx, a lightweight engine specifically made for mobile devices. It is a collection of *3 components: front end, decoder, and linguist* [6]. Linguist contains an acoustic model (reports the sounds of words in which any grapheme is uttered), dictionary and language model (states probability of each utterance.) Signals are inputted into the front end which are parameterized into an arrangement of features. The linguist interprets the pronunciation data present in the dictionary alongside the language model data and structural data from *acoustic model, into a search graph* [6]. Decoder incorporates *the search manager which inputs the features from front end and search graph from linguist* [6]. This is where the real decoding occurs and where results are produced. These results are sent to the application.

### Background service

The authors of this article used PocketSphinx to create an in-app speech recognition system. Given that this lightweight library can also run in the background of a mobile device continuously, the Speech Radar app may require PocketSphinx. On top of this, the article mentions it works without Internet connection, which can be useful if users’ phone were to suddenly disconnect.

Add section here for project aims

# Specification

Chapter 3 states a detailed explanation of all the requirements needed to fully create the app. Examples include how it should function, how it could handle failures, what measure should be taken to prepare for potential disaster, how user can interact with the interface and many more. This would be a great section to look back at after implementation has finished, to see whether all requirements have been met.

## Functional requirements

Describes the behaviour of the system. Java with Android development and XML will be used to achieve this.

* System should send an email verification when user creates account to prove that provided email address exists
* Verified user should be recognised by the system and grant them access to their account
* System should ask user for permission to record audio and access the Internet when using the app first time
* When correct button is pressed, audio should record for 3 seconds to receive user input of the code word
* If user is happy with the code word, background service of continuous speech recognition should activate when button is pressed
* Background service should deactivate when user utters the code word or terminates the app
* App should work on all android devices that have API level of 16 (Jelly Bean) or above

## Non-functional requirements

Explains requirements of how the system should perform or is unrelated to the functionality of the app.

* System should continue to operate in an event of a failure (e.g. no Internet connection)
* System should load the TensorFlow model via the assets folder and use it to make predictions
* Quick response time when converting speech to correct text
* APK file should be 30MB or less
* A backup of all user accounts should be kept in separate database in case of a disaster
* The continuous background service should not deeply affect performance of the phone (e.g. drain too much battery life, slow down the phone etc.)
* All android devices must be touch-screen to use the interface
* Interface should be intuitive (e.g. add few objects and spread them across the screen so the layout is clean)
* System should be able to handle large number of users all online at the same time

## Java requirements

Discusses what tasks should be completed with Java.

* Insert account details to Firebases Real-time Database when user creates an account
* first name field should be pulled from a specific record in database and displayed when user logs in to their account
* For security purposes, password should be hashed when added to database
* Utilise Google's free SMTP server to send email verification and password reset emails upon request with Firebase Authentication
* Authenticate email address and password in login screen
* Load the TensorFlow model and get it to make predictions in the speech recognition screen
* The SpeechRecognizer API in Android Studio should be used for continuous recognition in background. The API will stream audio to servers in remote locations to carry out speech recognition

## Database requirements

Backend database must meet these criterias.

* Offer quick data retrieval
* Data should be correct, consistent and update when needed to

## TensorFlow requirements

Following targets in TensorFlow should be met.

* Should have a similar architecture to the article revised in section 2.2
* Testing accuracy should be over 90%
* More than 15 words should be used for recognition
* Use Speech Commands Dataset explained in section 2.1

# Design

This chapter discusses the design of the Android application through diagrams, flowcharts, pseudocode and wireframes. Wireframes will be used to describe the appearance. Whereas the rest will be used to describe the structure or behaviour of the app. The chapter will only display the designs related to speech recognition so the main area of the app can be revised and established. Designs for login screen, create account and forget password can be found in appendix.

## Visual flowchart

### TensorFlow model

Before designing the application, it would be wise to plan the model architecture in TensorFlow first. Figure 1 shows a visual flowchart of how this would look like. The CNN model is very similar to the one used in the article analysed in chapter 2 section 2.2.1, where two convolutional layers are used along with one fully connected layer at the end with SoftMax activation function. Every convolution layer has a ReLU activation function which will follow with max pooling and dropout layer. Batch normalization could also be used, as we saw in chapter 2 section 2.2.3 a sudden increase in validation accuracy and a steady drop in validation loss when used. But, in this case two convolution layers are used, which will give us similar results.

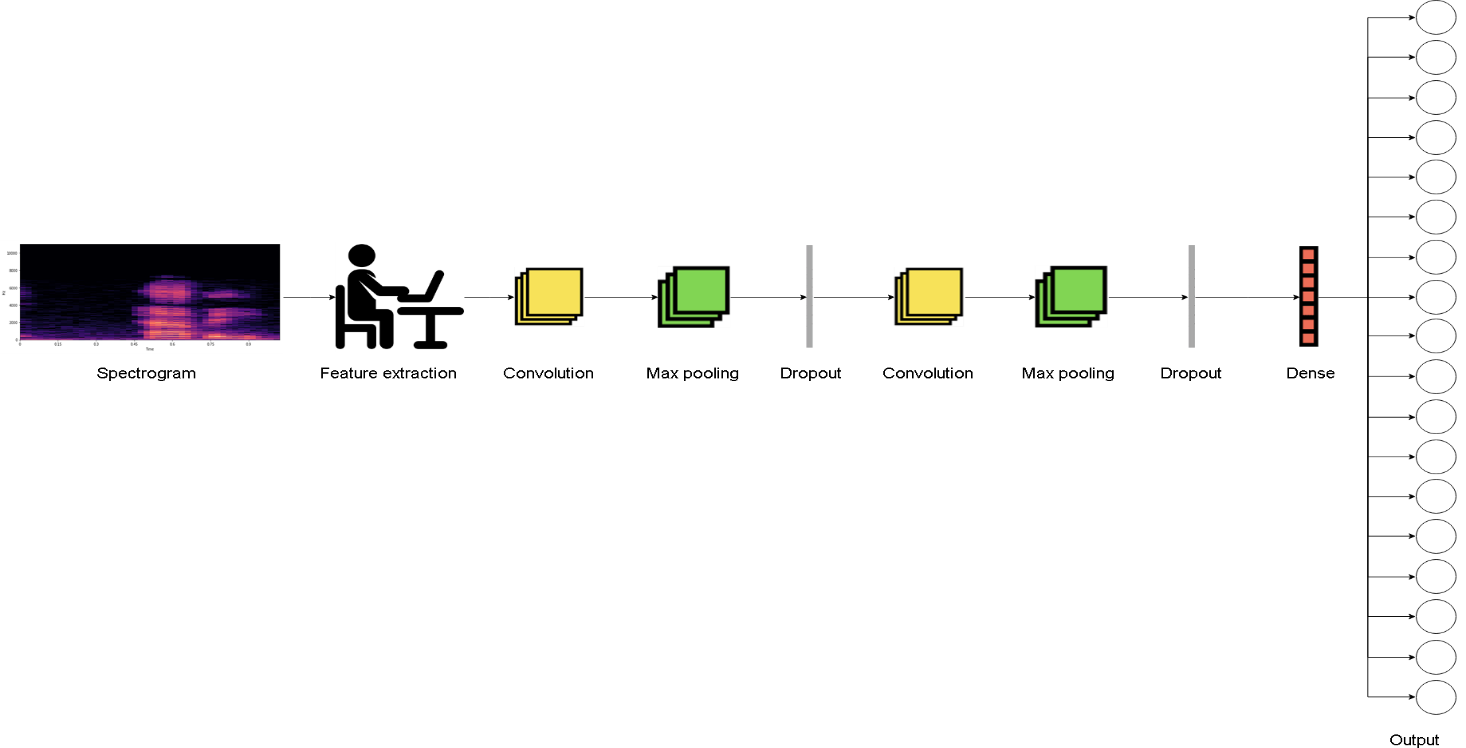
Features such as time and frequency are extracted from the spectrogram to help classify the audio signal. 18 different words will be used for classification. Thus, the reason why 18 outputs are shown in the flowchart, making this a multi-class classification problem.

Figure 1

### Loading model into Android Studio

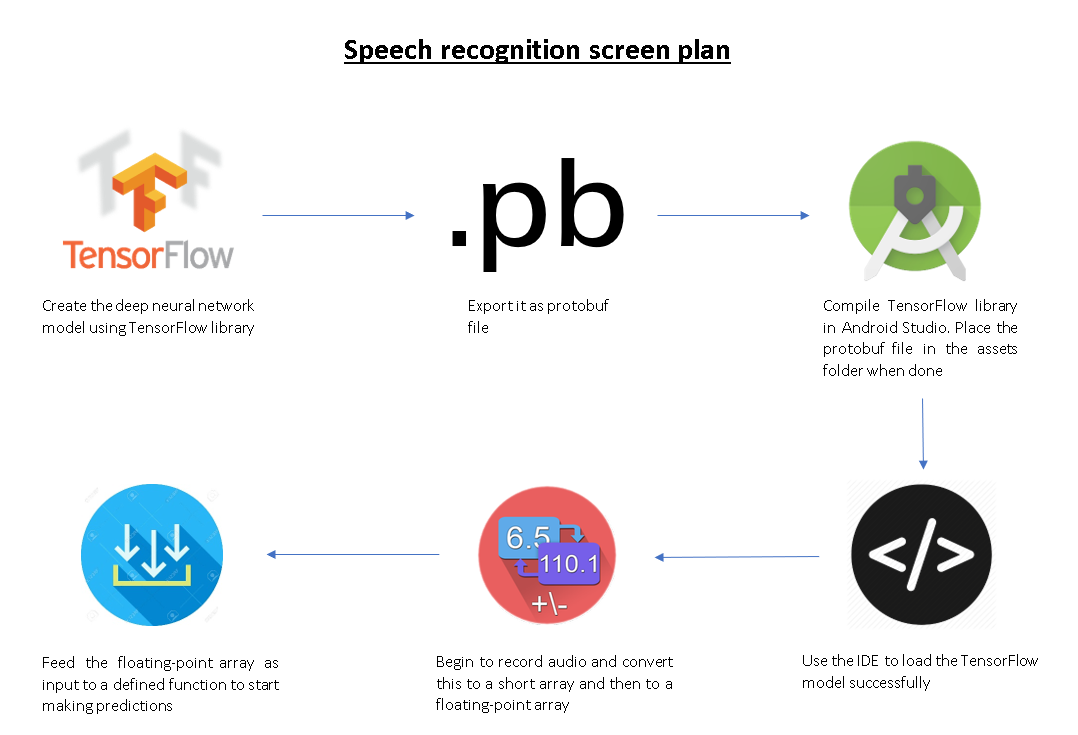
The flowchart in figure 2 clearly explains with text on how to load the model in Android Studio and get it ready to make predictions. The only step that may not have been explained as well is where we export as .pb (protobuf) file. This basically freezes the trained model that carries the graphs definition and weights, which will be needed to make predictions in Android Studio.

Figure 2

## UML diagram

### Class diagram

UML helps to provide a way of visualising the design of a system. Class diagrams were used first in this project to envision our classes in Java. It maps out the structure of a system by representing its classes, function and relation existing between objects. This section will go over the speech recognition and background service screen. All the class diagrams for the other screens can be found in appendix.

#### Speech recognition

The class diagram in figure 3 shows all classes needed by the speechrecognition class. First, we see a one-to-many relationship between FirebaseDatabase and DatabaseReference, as one database can have many references (e.g. firstname : "John", lastname: "Watch" etc.) Both classes are needed by speechrecognition mainly to save the code word in the database.

A one-to-one relationship between speechrecognition and loginscreen represents the extraction of email address provided by user in the loginscreen class, so the system can access the correct parent in the database and display the correct account details. The database structure is explained clearly in section 4.4.

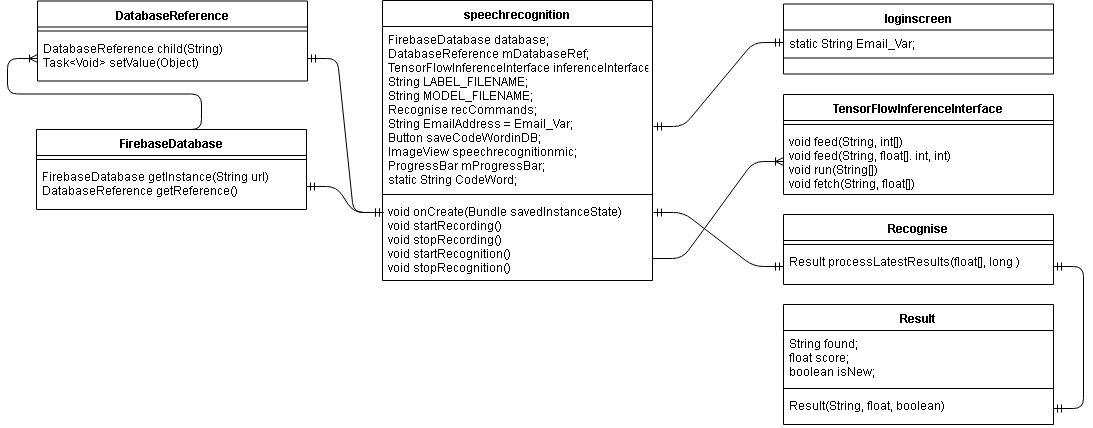
**** Another one-to-one relationship seen between speechrecognition and TensorflowInferenceInterface is needed to start or stop the model from making predictions. Finally, a one-to-one with the Recognise class is required to average the audio signals and return information about a particular label when there is enough proof to consider that a word has been found.

Figure 3

#### Background service

The class diagram in figure 4 shows a capable structure of the background service. BackgroundService is the main class that encompasses the visual aspect of the screen and in running the continuous service. A one-to-many relationship with MyService is essential, as most of the implementation for the never ending service is in the MyService class. All the BackgroundService class has to do is start it.

The one-to-one relationship between MyService and speechrecognition is necessary for the audio recognition system running in the background service to know which code word to spot. The code word is registered in the speechrecognition class analysed in previous class diagram. Once it detects the code word, it'll play the ringtone to max volume.

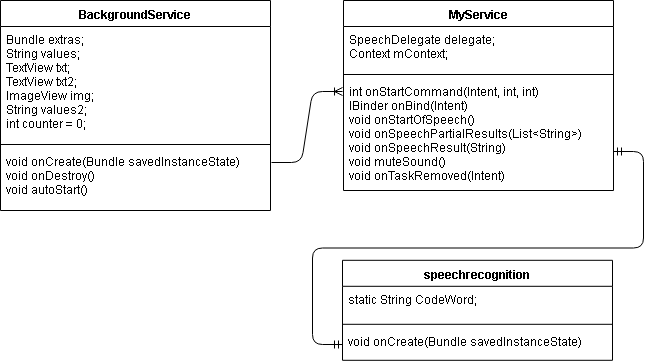
****

Figure 4

### Activity diagram

Activity diagram is important in UML to describe the behaviour of a system. It is essentially a flowchart showing the flow from an activity to another. The activities can be thought of as an operation carried out by the system. This can help visualise the sequence of actions that need to be taken in Speech Radar. Take for instance the diagram in figure 5, which shows the full activity of the app from beginning to end. Indeed, it has helped to avoid any exceptions/failures the user may potentially face. For example, if user has forgotten password, then they are sent an email to reset it and log in again with different password. Mid-way through the diagram, we see the user enter the speech recognition screen after logging in. Here, the user clicks the mic button to record audio of them saying a particular code word. The word is saved in the database and user has then the option to terminate the app. If not, then the continuous speech recognition background service will run forever until it detects the code word.

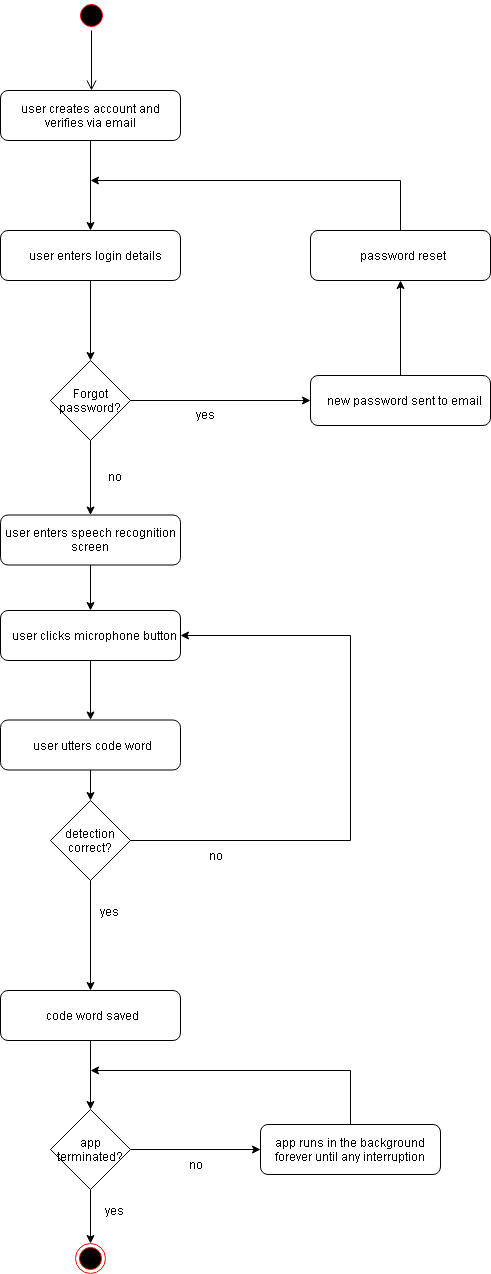


Figure 5

### Use-case diagram

A use case diagram is another important UML that describes the behaviour of a system. It models the process of a system using imaginary actors and use cases that consist of actions that the system must execute. This section will go over the speech recognition and background service screen. All the use case diagrams for the other screens can be found in appendix.

#### Speech recognition

The diagram in figure 6 further illustrates features in the speech recognition screen that users can interact with. On the other side, it shows what the use cases depend on in order to carry out the operation. For example, users can interact with the mic button, which needs the TensorflowInferenceInterface class to make predictions from the recording audio. User can also interact with the activate button, which depends on Firebase Database to add the code word. It depends on Intent class to open the background service activity. Similarly, the back button requires the Intent class to go back to the login screen activity.

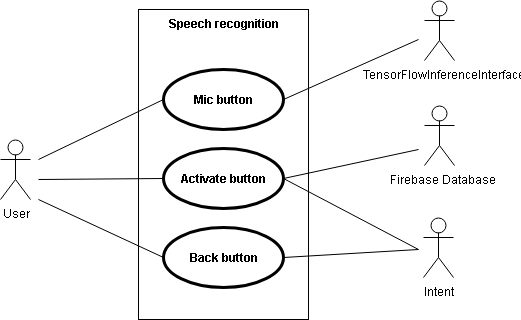


Figure 6

#### Background service

The diagram in figure 7 illustrates how the background service is activated. It shows that the user is not involved in activating it, but rather the system. Once the user enters the background service screen, the service will automatically start with a pop-up suggesting this to the user. This removes the need for an extra button. The system can also stop the service if the user decides to terminate the app or the user utters the code word detected by the background speech recognizer.

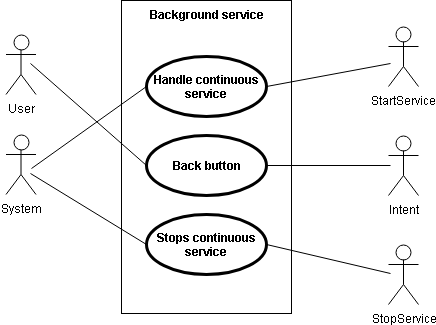


Figure 7

## Database schema

The tree-like diagram in figure 8 describes the structure of the database. It’s important to note that Firebase is NoSQL, so using an entity relationship diagram would not be correct. Underneath the root node, we see two parent nodes. Each parent represents a user account, which will be entitled as the user’s email. The child nodes are simply key-value that’s added from the app (e.g. firstname: "Alex", lastname: "Smith", password: "3dfk9p31s", codeword: "Marvin" etc.) The password and confirm password nodes contain hash value, to protect user account if database is exposed.

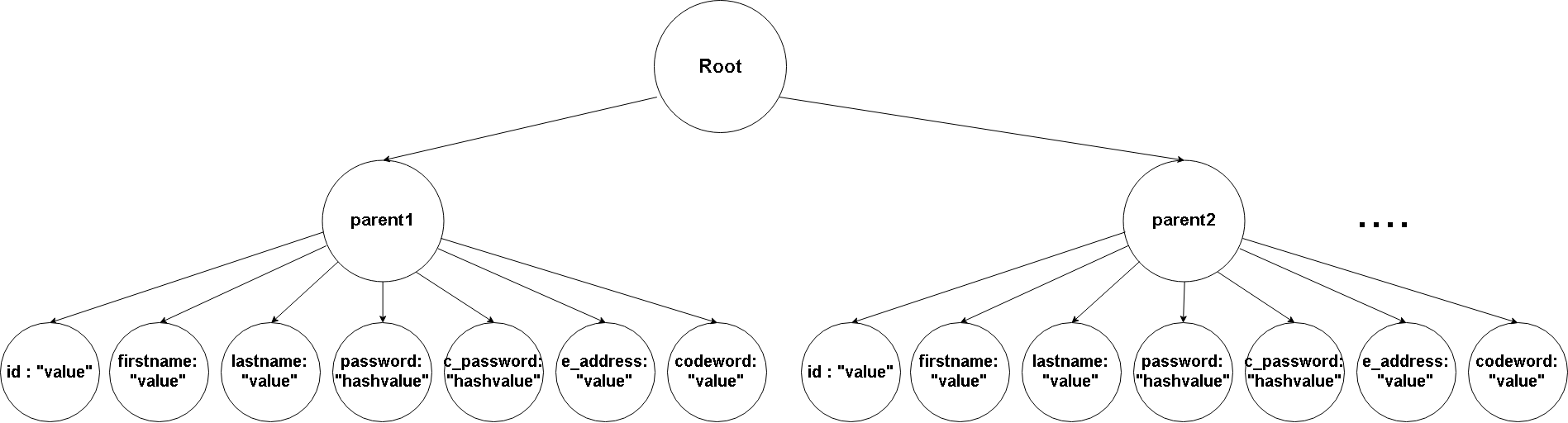


Figure 8

## Wireframes

Wireframes are a great way of representing the appearance of an app. Their purpose is to order various objects, so one can fulfil a particular objective. This section goes over the wireframes for speech recognition and background service screen. All the wireframes for the other screens can be found in appendix.

### Speech recognition

The wireframes in figure 9 and 10 shows us a first glimpse of how the speech recognition screen would look like. In figure 9, we see the screen before the mic button is pressed, where the code word and activate button are visible. Figure 10 shows the screen after the mic button is pressed, where we see a white rectangle appear showing the results from what the speech recognition system picked up. The layout is similar to Google Assistant. The two figures show a simple layout to make it intuitive for users.

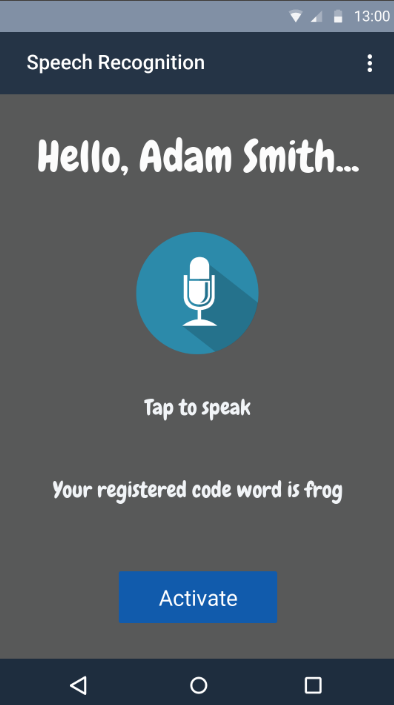
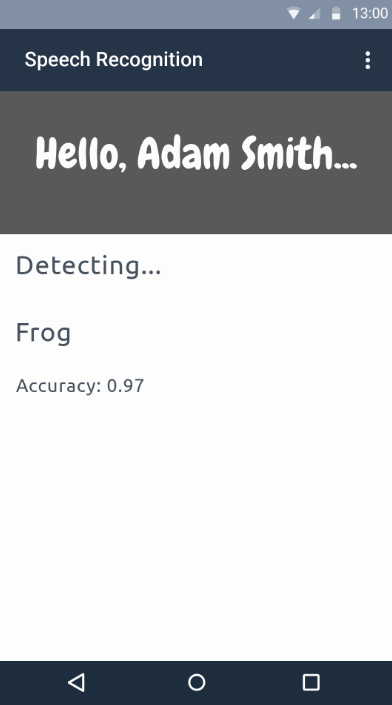


Figure 9 Figure 10

### Background service

The wireframe in figure 11 shows the background service screen. Here, there is nothing for the user to interact with. It is just to inform user that the continuous speech recognition service has automatically activated in the background. The screenshot within this screen will show what the user should not do (i.e. to not terminate the app) if they want the service to run.

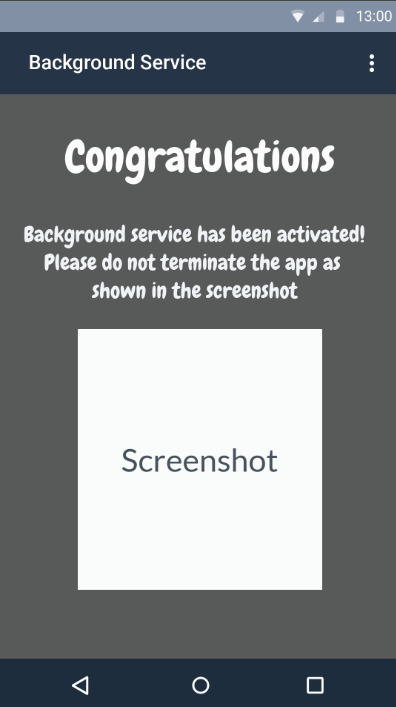


Figure 11

## Pseudocode

For the final part of design, pseudocode is written for every screen. It uses the structure of a programming language but is written in a way that can be understood by humans rather than a machine. This section will show the pseudocode written for speech recognition and background service screen. All pseudocode for the other screens can be found in appendix.

### Speech recognition

Figure 12 shows pseudocode for the speech recognition activity. It is written under the onCreate method, which is executed as soon as the activity starts. It states if user clicks mic button, then the system should read the protobuf file, read the labels file in .txt format, start the audio recording and get the model to make predictions of the recording audio. Otherwise, stop the recording and stop the model from making predictions. For the next if statement, it states that if activate button is pressed, then the code word should be added to the database and the background service activity should open. The final if statement states if the back button is pressed, then the login screen activity should open.

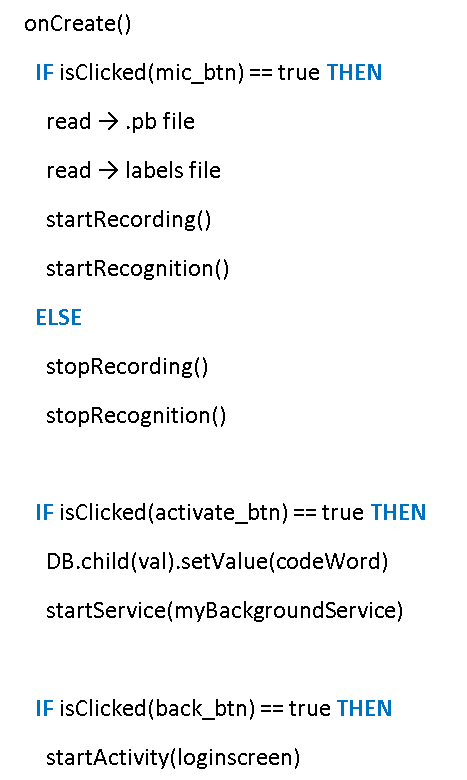


Figure 12

### Background service

Figure 13 shows pseudocode for the background service activity. It states if user clicks back button, then it should go to login screen activity. Under this if statement, a function entitled 'checkServiceRunning()' is used in onCreate. This function will simply check on the continuous speech recognition service to see whether it has stopped unexpectedly or whether the app has been terminated. If it has been stopped, then the function will try to restart it. If the app has been terminated, then the service will stop.

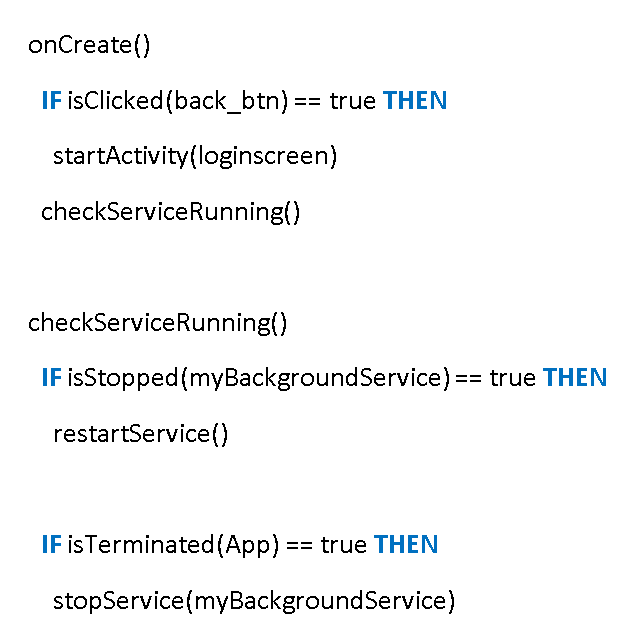


Figure 13

# Implementation

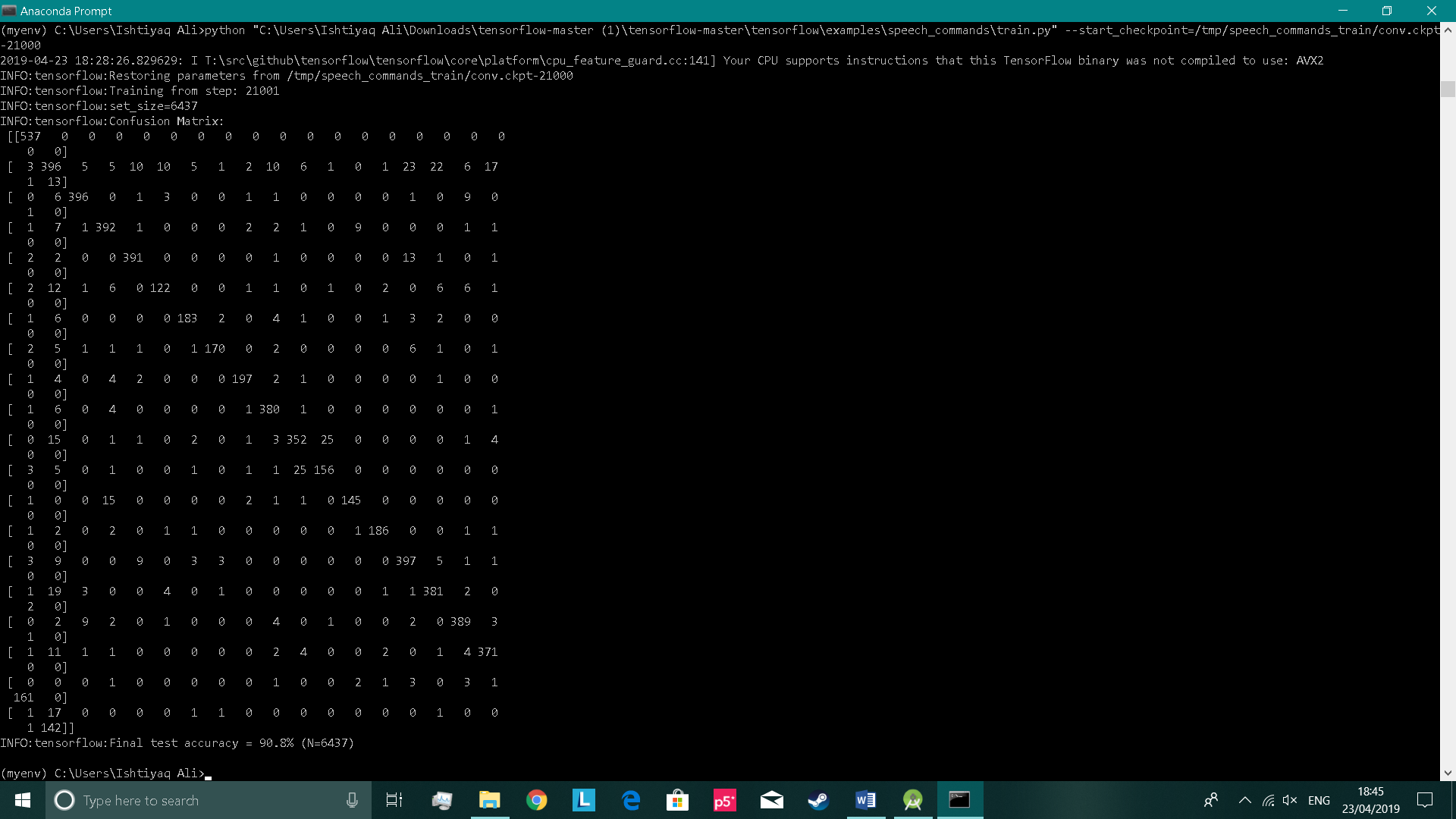
This chapter mentions how Specification and Design section were used to implement the app. It will describe thoroughly the implementation in TensorFlow, Java, XML, Firebase Database and Authentication. Screenshots will be used to aid in the explanation.

## TensorFlow

A good model was created that achieved a 93% testing accuracy. However, when it was exported into Android Studio, it could not make predictions for reasons that did not find proper solutions. Thus, another approach had to be taken. Thankfully, TensorFlow implemented a simple audio recognition for Android located in their GitHub repository. It followed the same article revised in section 2.2 for the keyword spotting task using the Speech Commands dataset. It used only 10 words for recognition. Since the original plan of creating our own model has failed, TensorFlow’s implementation will be extended for this project so that it can recognise 18 words. The main aim of the project is to implement the app, so this cannot really be seen as a setback. The model architecture seen in Design section will still be used.

### Test accuracy with confusion matrix

At first, TensorFlow’s implementation outputted low test accuracies around 86% to 89%. To boost the accuracy, more data was needed in the dataset. Thus, hundreds of additional one second audio clips were added for each word. This helped boost the accuracy to 90.8%, which was enough to meet the specification for this project. The screenshot in figure 1 shows proof of the accuracy after taking 21000 training steps. It also shows a confusion matrix, which helped visualise the performance. Each row and column can be seen as a set of words that are going to be used. Figure 2 visualises this better and shows the words that are going to be used. The basic principle is each word in row will be checked by each word in column by the model to show whether it has predicted correctly, which in our case has. Although 18 words are used, the confusion matrix is 20 x 20. This is because ‘Silence’ and ‘Unknown’ labels are included.



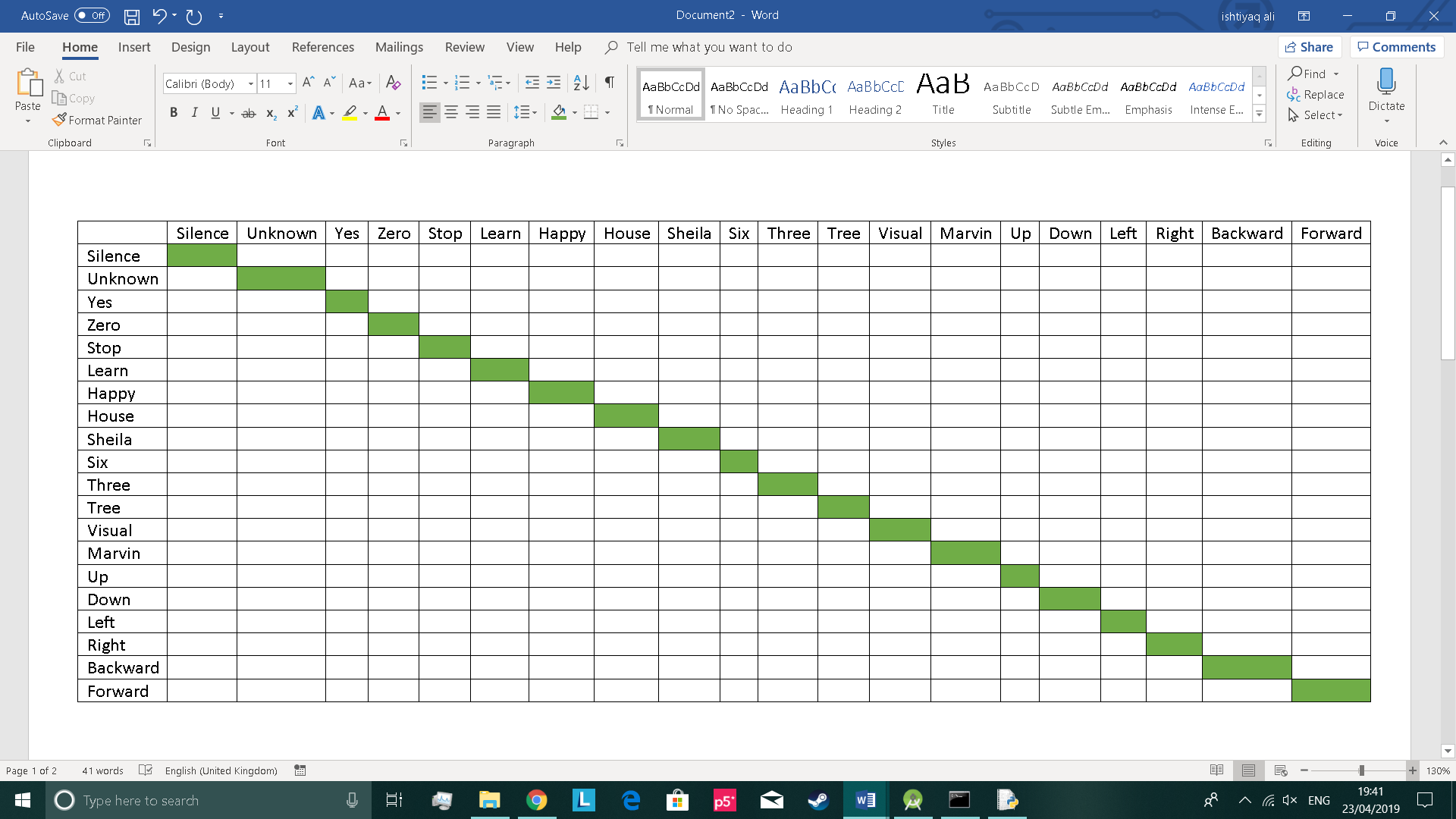
Figure 1

Figure 2

### Scalar diagram

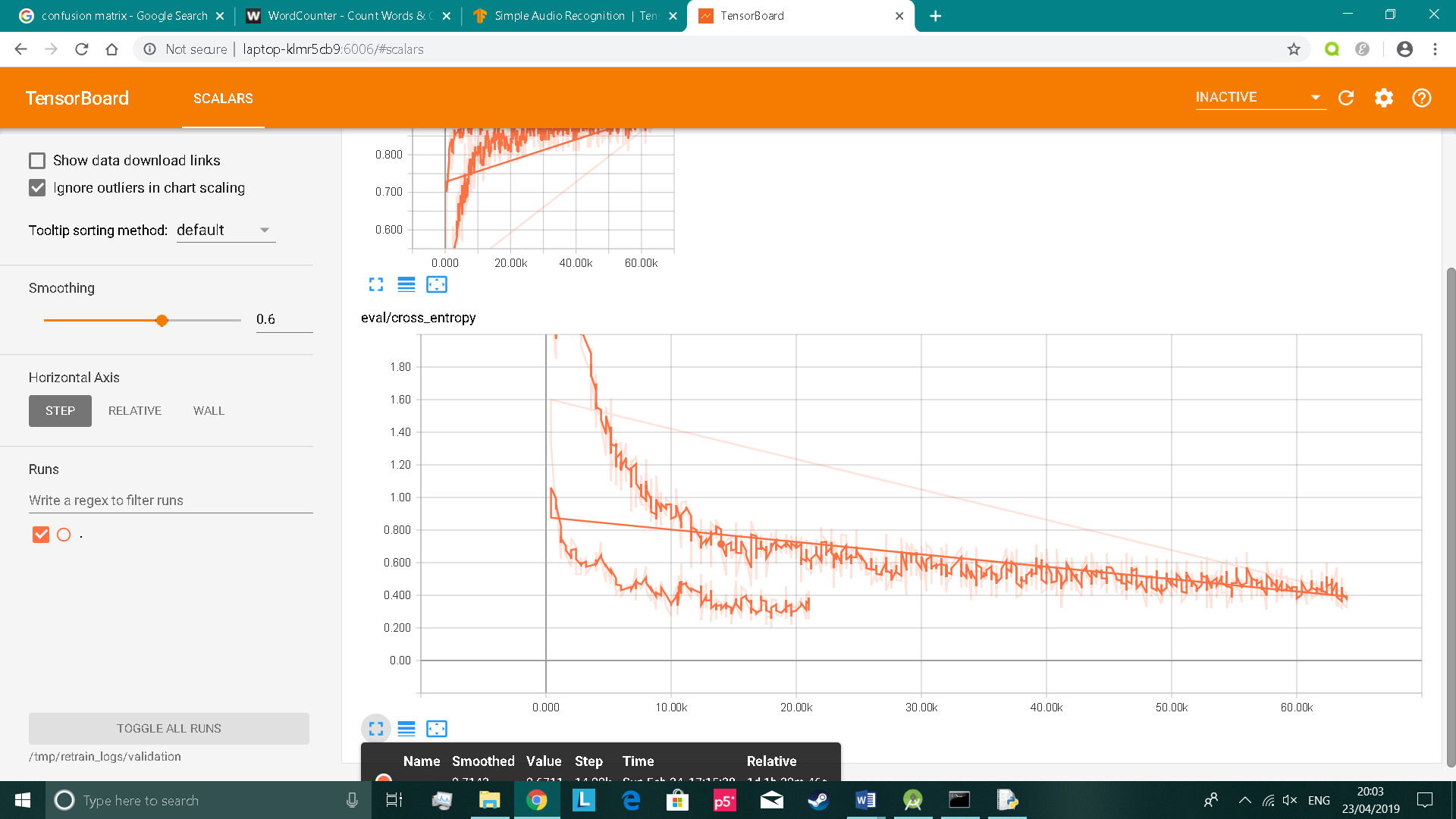
Figure 3 shows the validation loss in the form of a scalar diagram. It shows two different lines. The line ending at 60000 training steps is the first attempt that was done before adding hundreds of audio clips to the dataset for each word (as mentioned in the previous section). The line below it is the second attempt, which ends up with a similar loss with only 21000 training steps. The loss goes as low as 0.35. A low validation loss is important, as it tells us that the model will do well with unseen data.

Figure 3

## Java

This section goes over the main implementation for each class in Java. The aim of this is to explain the code and make sense of it at a human-level. The code was written with Android Studio IDE.

### Loginscreen

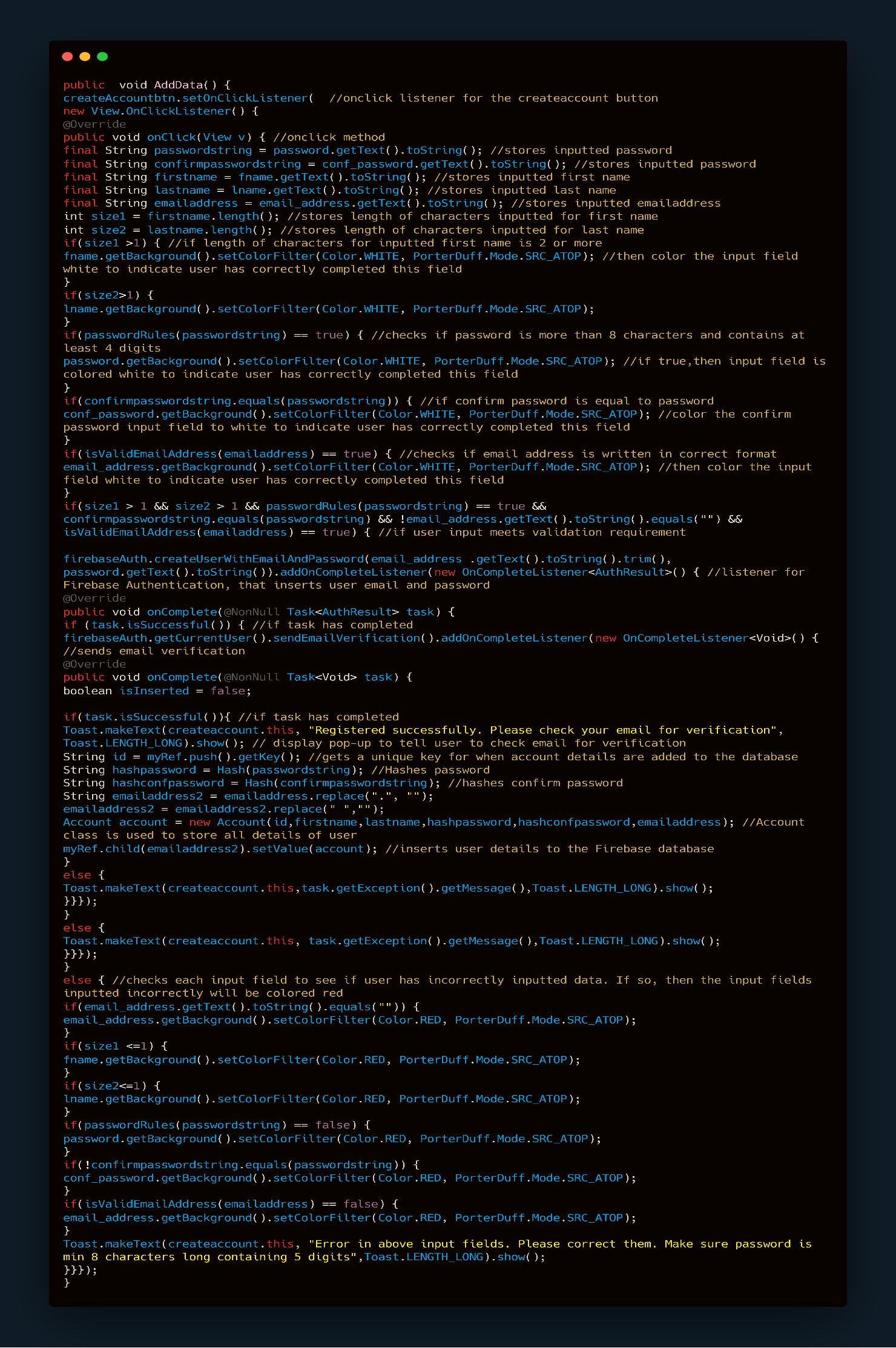
The main purpose of the loginscreen class is to enable users to login via email address and password. The check() method in code 1 runs when user clicks the login button. It shows how FirebaseAuth class is used to check whether the account exists, is correctly inputted and has been verified via email verification. If true, then the speechrecognition activity will start, where user can register the code word and activate the background service. Otherwise, a pop-up appears in the interface that displays the error details.

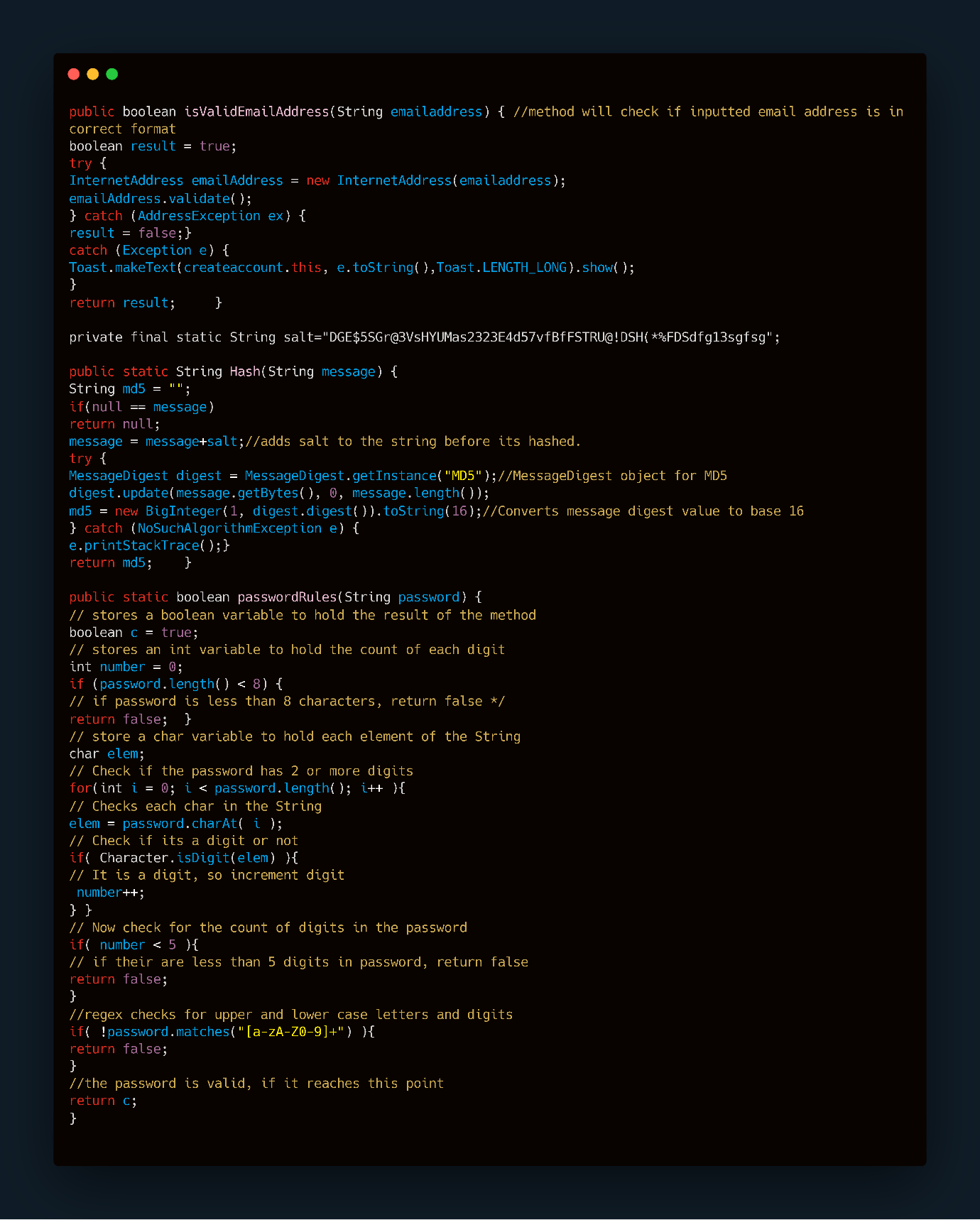
code 1

### Createaccount

The createaccount class is needed for when user creates an account. The following inputs fields require user input: first name, last name, password, confirm password and email address. The AddData() method seen in code 2 runs when user clicks the create account button. The first thing it defines is the validation of each input field. If the input goes against the rules, then it will stop user from creating an account, display an error message and colour the input fields red to indicate where the errors are present.

Otherwise, the program will store the account details in the database through help from DatabaseReference and FirebaseDatabase class to create a child and insert into it. The password field and confirm password field are hashed using the Hash(String message) method seen in code 3, to protect passwords against potential attacks on the database. Then, an email verification is sent to the inputted email through FirebaseAuth class, to authenticate user.

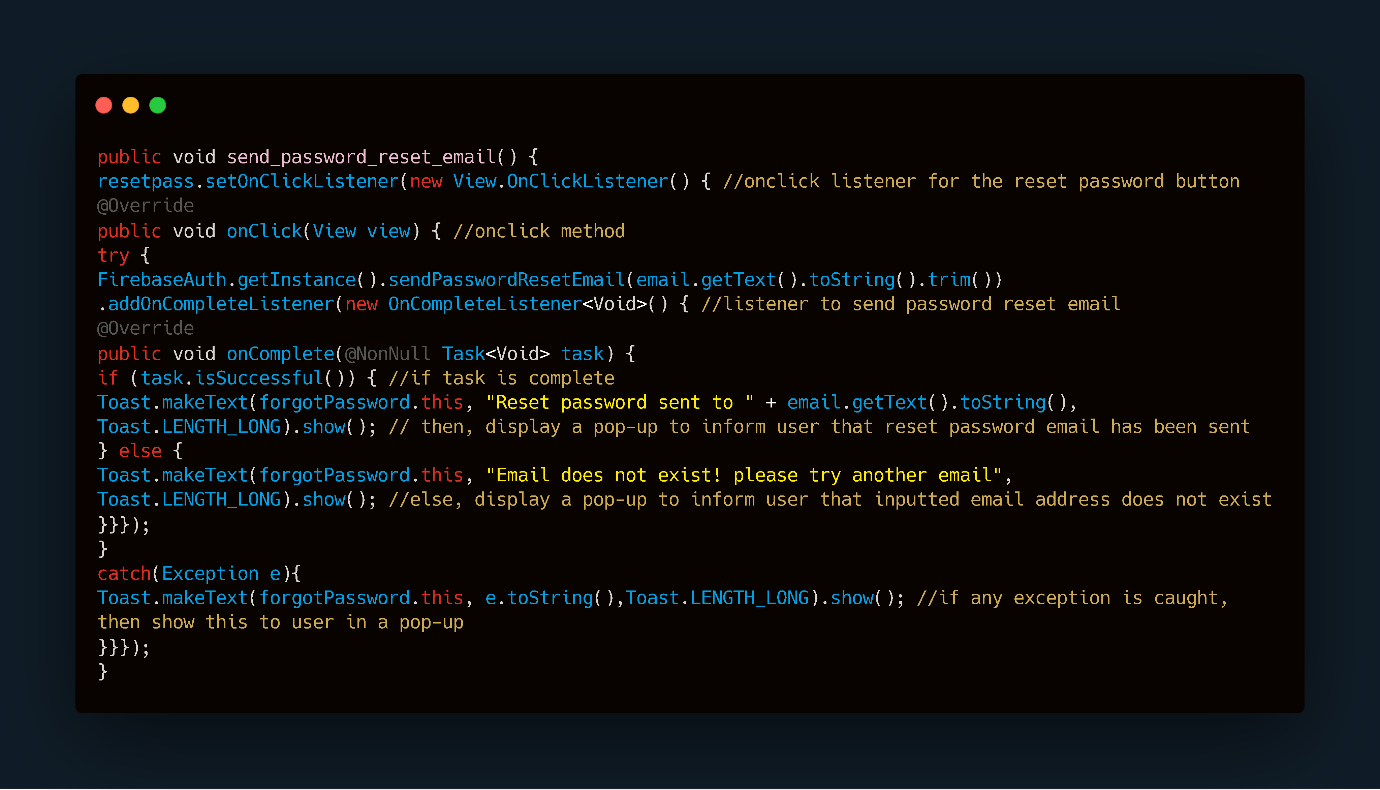
code 2



code 3

### forgotPassword

The forgotPassword class enables users to reset their password, if they have forgotten it. The send\_password\_reset\_email() method in code 4 runs when reset password button is pressed. It checks whether the inputted email address exists, and if it does then a password reset email is sent. The email will allow the user to create a new password. This is again done via the FirebaseAuth class, which has proven to shorten the lines of code for all classes it has been used for.



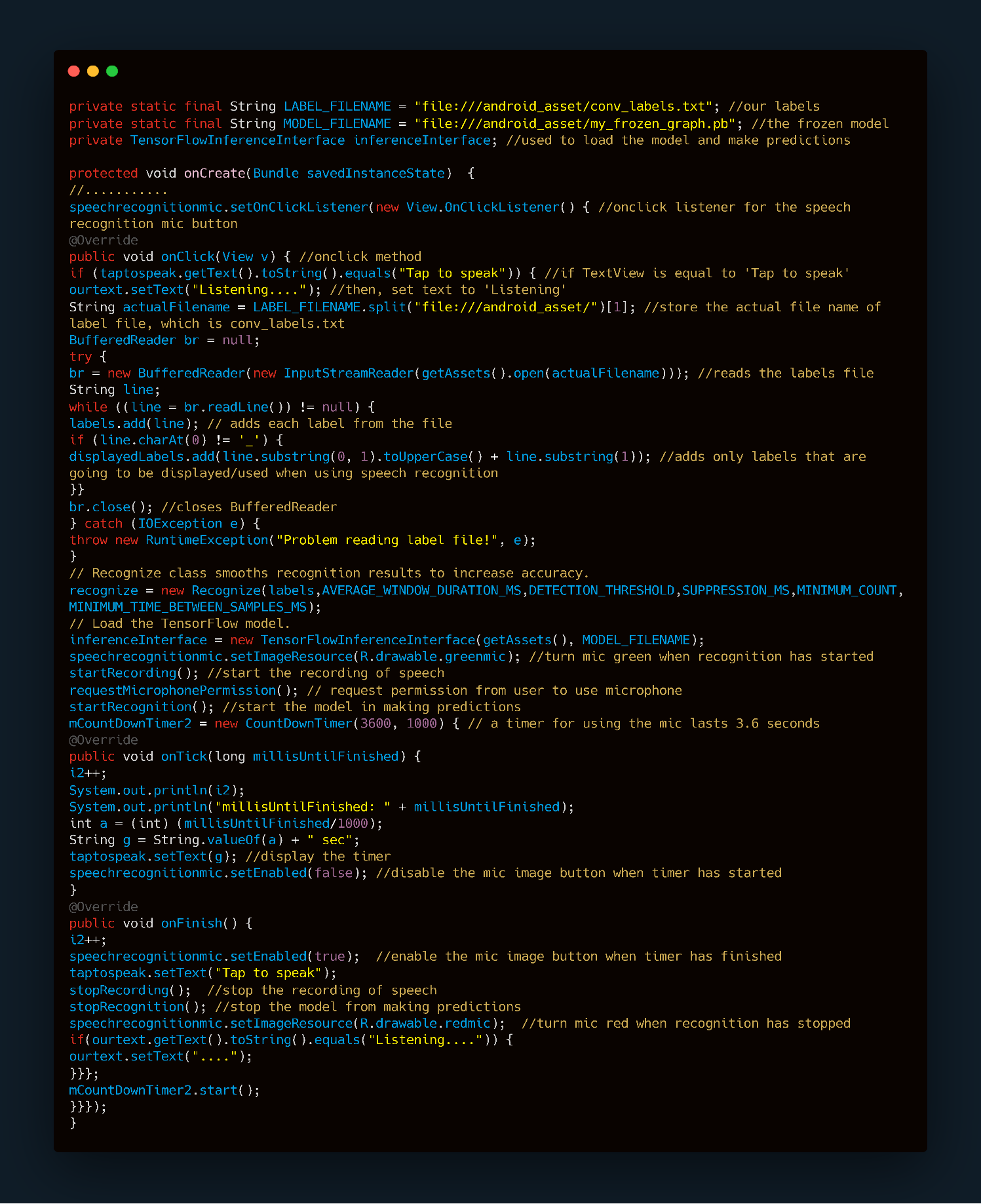
Code 4

### speechrecognition

The speechrecognition class is where the main implementation of Speech Radar lies. It contains a lot of code that had to be separated out. The implementation was obtained and further extended from TensorFlow’s GitHub repository, where it was mentioned previously that the same repository was used to acquire and use a network architecture in TensorFlow that works in Android. The onCreate(Bundle SavedInstanceState) method seen in code 5 is used to initialise the activity. It contains code that runs when the speech recognition mic button is pressed. This button will begin using TensorFlowInferenceInterface class, which loads the model located in assets folder.

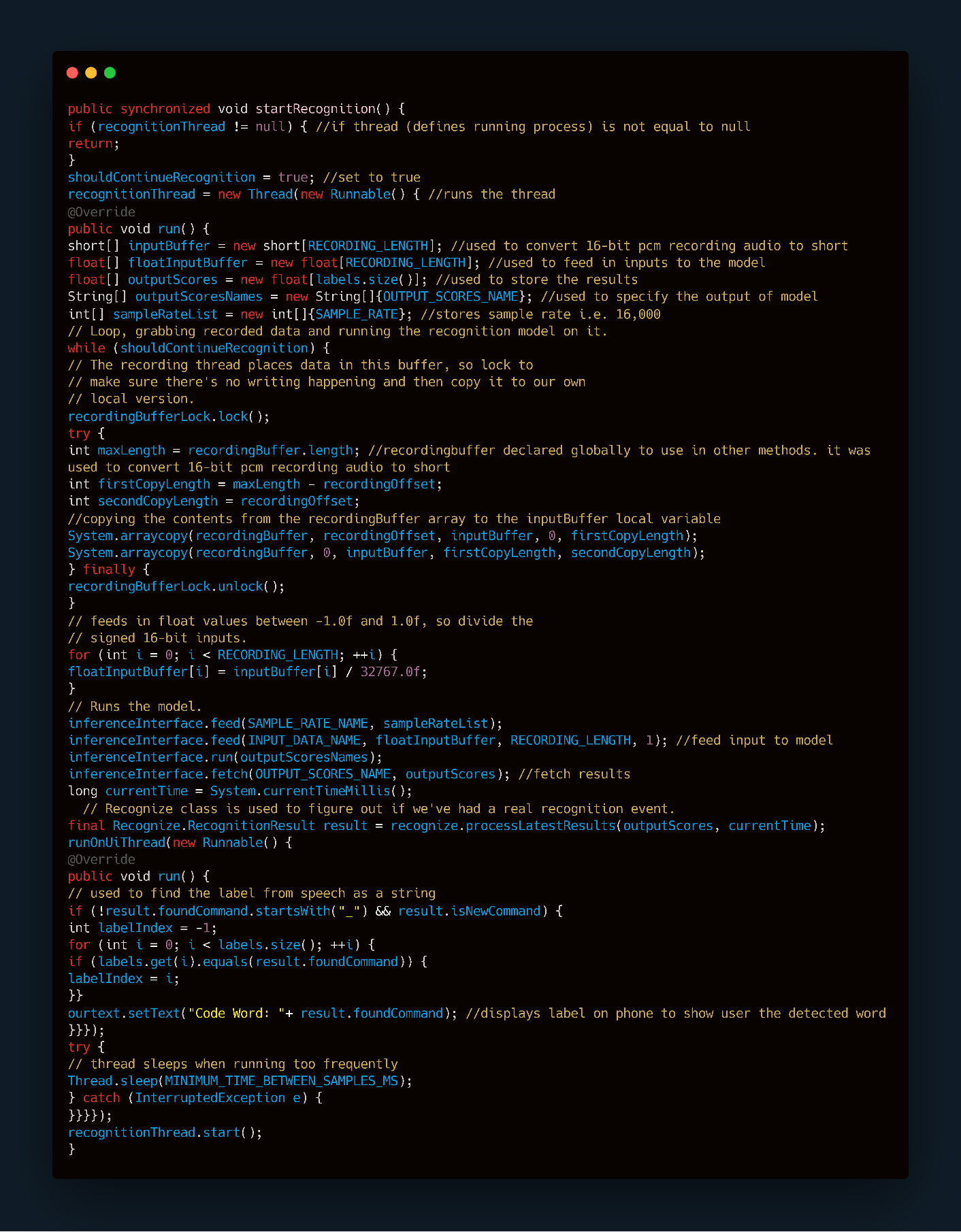
Then, the labels file is loaded and added into an ArrayList. Labels include: \_silence\_, \_unknown\_, yes, 0, stop, learn, happy, house, Sheila, 6, 3, tree, visual, Marvin, up, down, left, right, backward, forward. After, Recognize class is instantiated to smooth recognition result and increase accuracy. This class was also obtained from TensorFlow’s GitHub repository. Implementation of this can be seen in appendix.

Finally, audio starts recording using the startRecording() method and display predictions based on the uttered speech in the startRecognition() method. If the button is pressed again, audio will stop recording through stopRecording() method and predictions will stop being made through stopRecognition() method. These implementations can be seen below and are explained further.

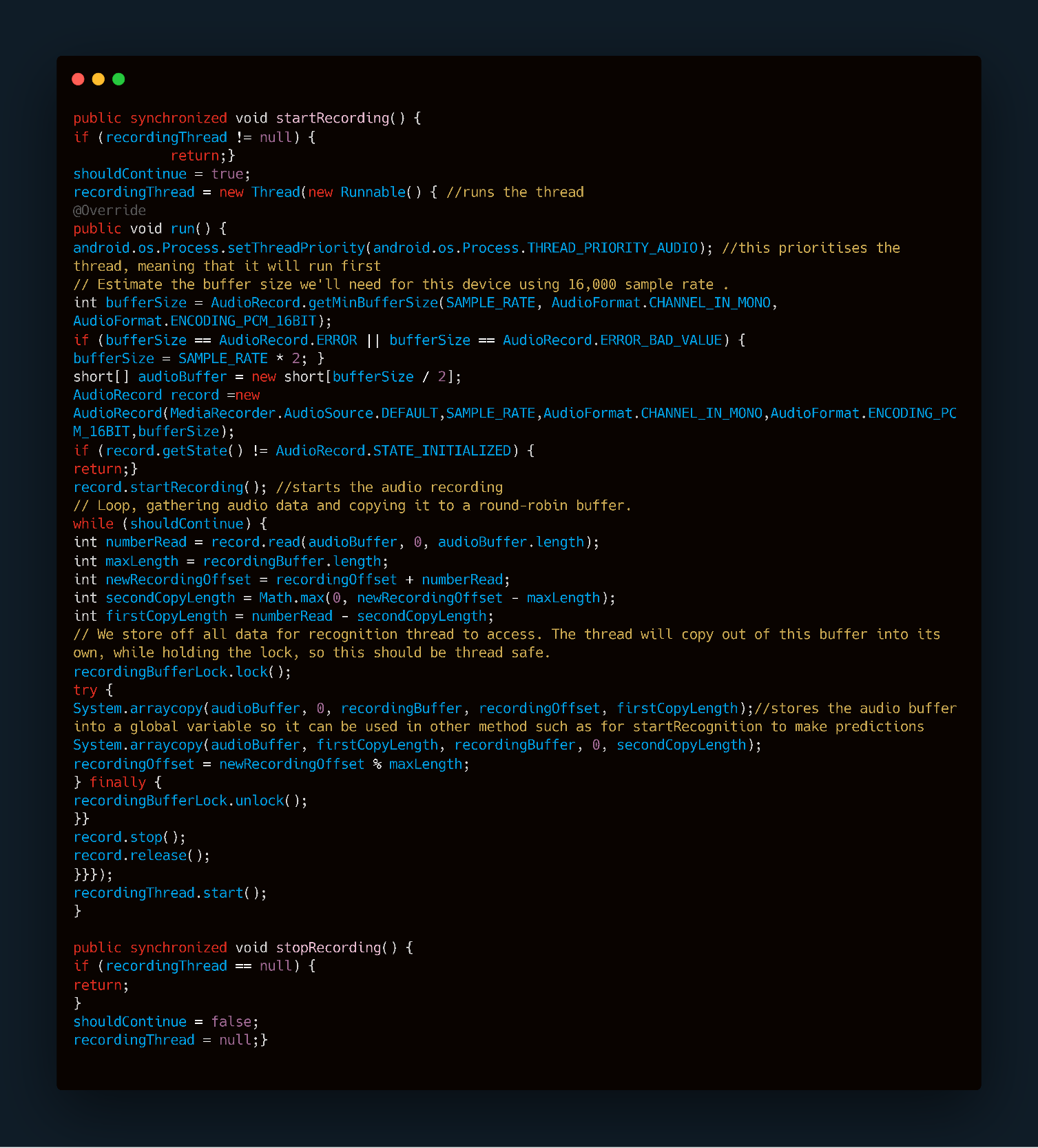


code 5

The startRecognition() method in code 6 runs a separate thread to make predictions from the imported TensorFlow model. It first stores the recording audio as a Short array. Then, it copies the values into a float array and divides it by 32767, since the encoding of audio is set to 16-bit PCM. The TensorFlowInferenceInterface class is then used to feed the float array to the model. The class is used again to fetch the output i.e. the prediction. Finally, the output is stored as a String and displayed in a TextView within the interface. The stopRecognition() method in code 7 stops predictions from being made in a separate thread.

code 6

Code 7

The two methods implemented in code 8 are: startRecording() and stopRecording(). The startRecording() method runs in a thread that's prioritised to execute first. The recorder is defined using Androids AudioRecord class, which lets you define the sampling rate, encoding and buffer size. The sampling rate used was 16000 and encoding was 16-bit PCM. The class is then ready to use to start recording. The stopRecording() method is run in a separate thread, which simply stops the audio from recording.

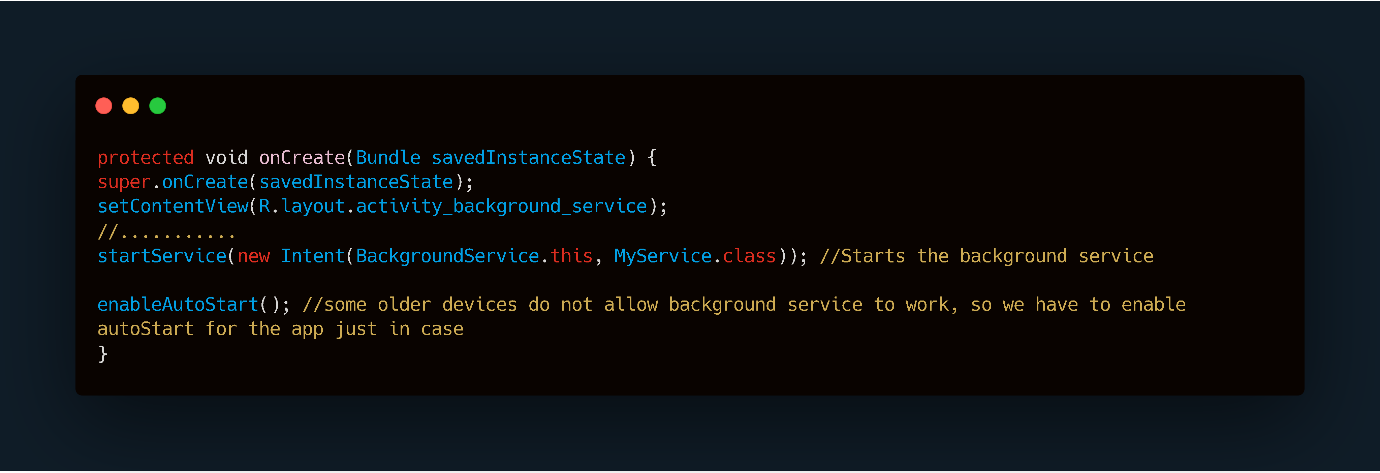
Code 8

The onCreate(Bundle SavedInstanceState) method seen in code 9 covers another important task. It contains code that runs when the activate button is pressed. This button will save the users code word in the database and start the backgroundService activity. Through the help of FirebaseDatabase and DatabaseReference class, it lets us enter the correct child and update the code word in the database.



Code 9

### BackgroundService & MyService

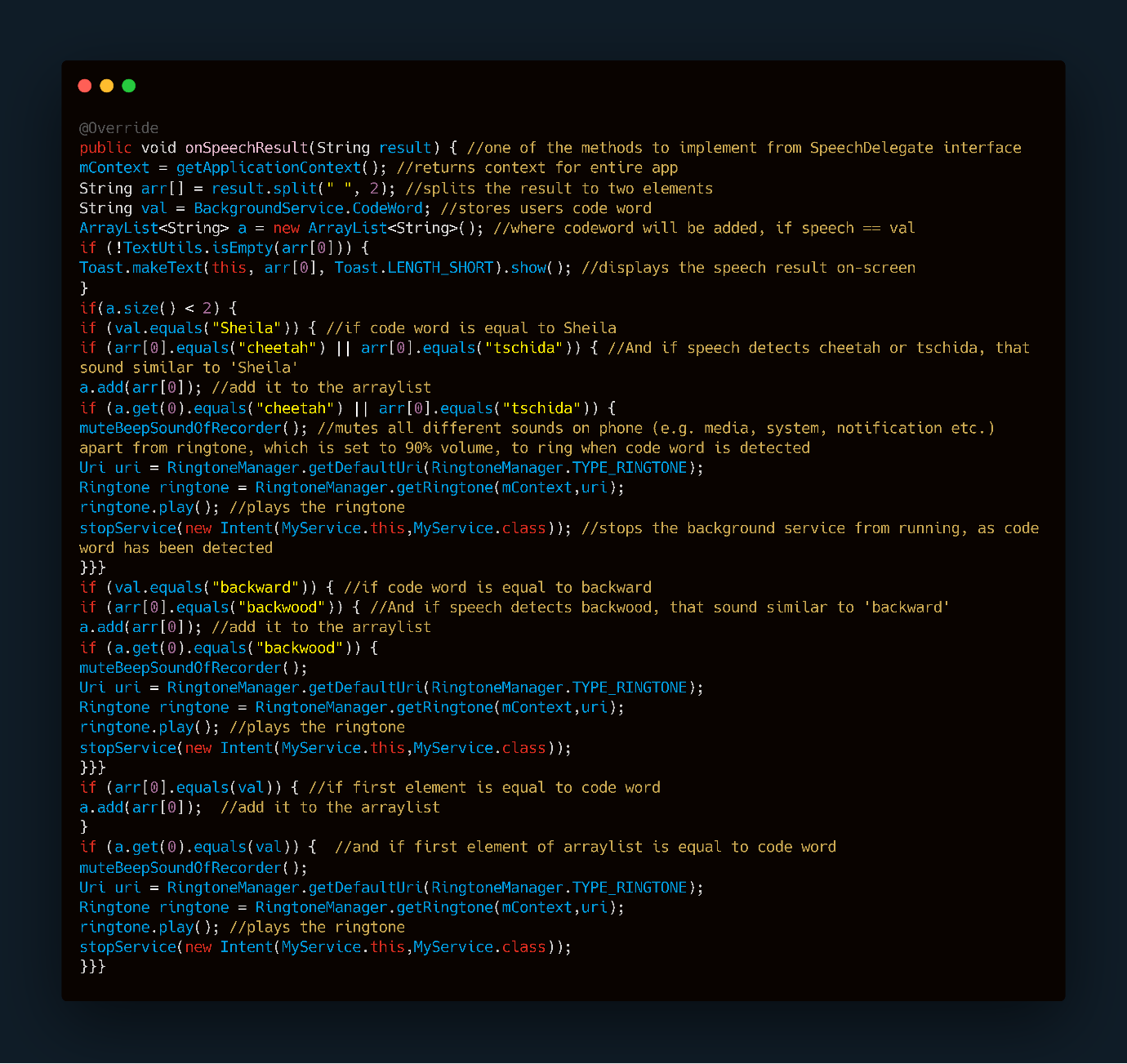
The onCreate(Bundle SavedInstanceState) method in code 10 shows how the backgroundService class starts the service. It uses Androids startService function, that starts the continuous speech recognition running in the background.

Code 10

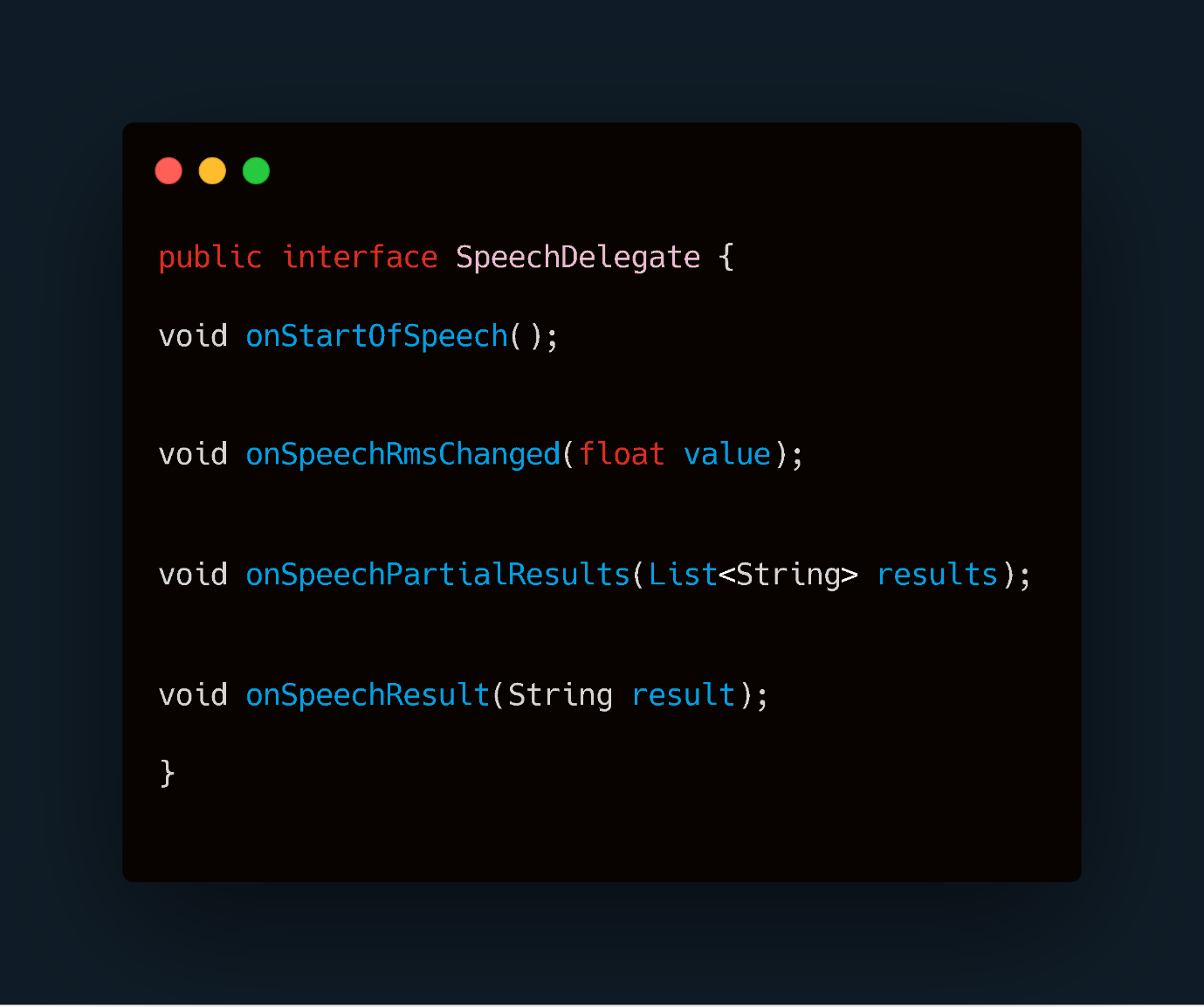
Now, we investigate the code for the continuous background service. This is another important task for Speech Radar. The onStartCommand method in code 11 is the first method implemented in the myService class. Like how onCreate is called at the beginning of execution for activities, onStartCommand is similar in that aspect but it works preferably with background services. It simply instantiates the Speech class, which is a library created in GitHub that enables users to easily use Google's speech recognizer in the background. Using the TensorFlow model to make predictions in the background was the initial plan, but it would not run continuously. Thus, we rely on Google's speech recognizer, which although takes a lot of battery power, works effectively. Using this library allows us to use defined method that continuously listens or stops listening for speech. At the end, it returns Service.START\_STICKY, which recreates the service when there is enough memory.

Code 11

SpeechDelegate is an interface, which contains four methods. onSpeechResults(String results) seen in code 12 is one of them and is important to mention. It outputs the result of what the speech recognizer has detected in String format. This String is then used to make if statements that state whether it is equal to the code word. If it is, then devices ringtone will start playing and will terminate the service completely. There are also if statements that express whether the result is equal to words that sound similar to the code word. For example, 'backward' is often misinterpreted by the speech recognizer as 'backwood.' If true, then ringtone will play. muteBeepSoundRecorder() is called in this method, which is explained in the next section.

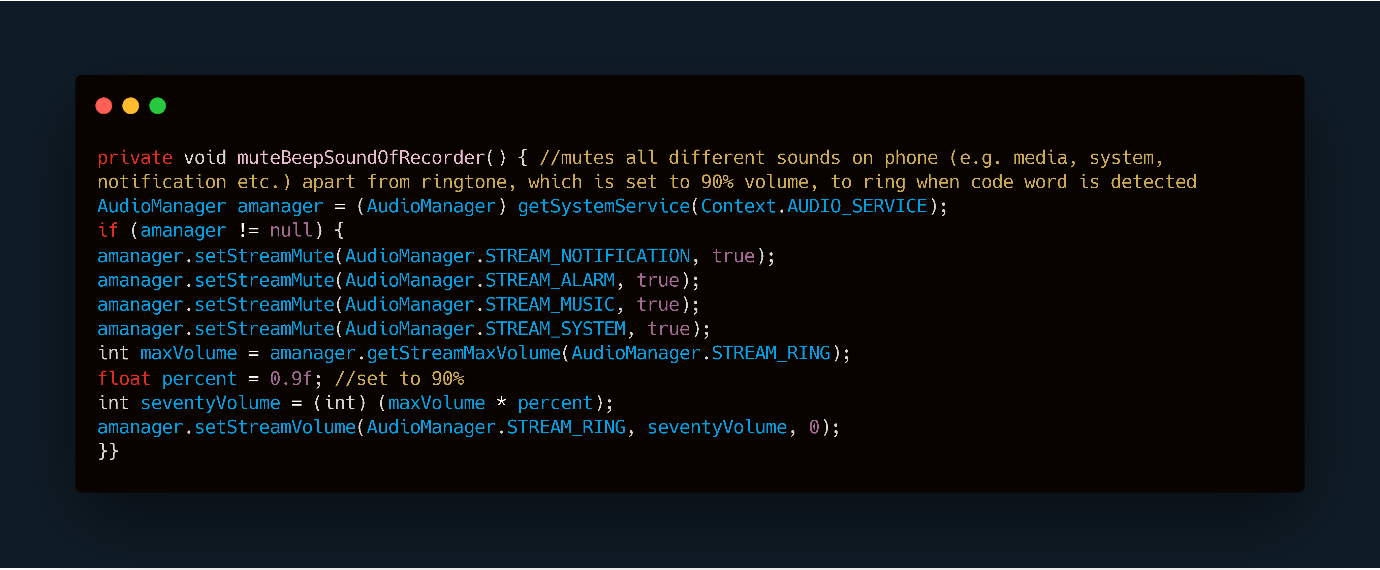


Code 12



Code 13

The method in code 14 mutes sounds such as media, notification and system, to block out the beep sound made by the speech recognizer. The ringtone sound is set to 90%, so when user utters the code word, it will play the ringtone loudly.



Code 14

## Firebase database

Firebase uses a NoSQL key-value database, which has a similar structure to a tree. There is a root node, which has many children nodes. Each child must be unique, so it was decided to use users email address without full-stop as the title, since every email address is/has to be different. Each child contains more children, that define the account information of user (e.g. firstname, lastname, password, confirmpassword etc.) Screenshot of this can be seen in appendix.

## XML

XML, similar to HTML is a mark-up language that uses tags to represent data. It is readable by humans and machine. XML is used in Android to design the layout, as it is lightweight which makes it lighter and quick to load on devices. All layouts made for each activity consist of a ScrollView at the top of the component tree to cater for small devices that cannot fit the whole screen and will need to scroll down. It then consists of a ConstraintLayout, which adds responsiveness to each activity. It defines the amount of spacing needed from each object (left, right, up and down), that translates for all screen sizes. The full code of each layout can be found in appendix.

XML is also used to define the manifest, which is a key resource file. It holds details required by Android about the app. It acts as link between the developer and the platform. No Android app can run without this file. The full code of the manifest can be found in appendix.

# Testing and evaluation

To do…

# Conclusions and future work

To do…

# Appendices

## **Design**

### Class diagram

### Activity diagram

## **Implementation**

### Class diagram

### Activity diagram

## **Testing**

### **Class diagram**

### **Activity diagram**

# Reference list/bibliography

**[1]** Erin Myers. "Little Known Facts About Speech Recognition Technology.” (2017). Retrieved from <https://www.temi.com/blog/little-known-facts-about-speech-recognition-technology/>

**[2]** Peter Samoff. "Alexa is getting better at answering users' questions.” (2018). Retrieved from <https://www.businessinsider.com/amazon-improving-alexa-voice-accuracy-2018-12?r=US&IR=T>

**[3]** Clark Boyd. "The Past, Present, and Future of Speech Recognition Technology.” (2018). Retrieved from <https://medium.com/swlh/the-past-present-and-future-of-speech-recognition-technology-cf13c179aaf>

**[4]** Sainath, Tara, and Carolina Parada. "Convolutional neural networks for small-footprint keyword spotting." (2015). *Keyword Spotting Task*, p. 1, *CNN architectures,* p. 2

**[5]** Graves, Alex, and Navdeep Jaitly. "Towards end-to-end speech recognition with recurrent neural networks." *International Conference on Machine Learning*, pp. 1764-1772. (2014). *Network Architecture,* p. 2

<https://arxiv.org/pdf/1804.03209.pdf>

<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7749004>

# Evaluation

Personal statement..