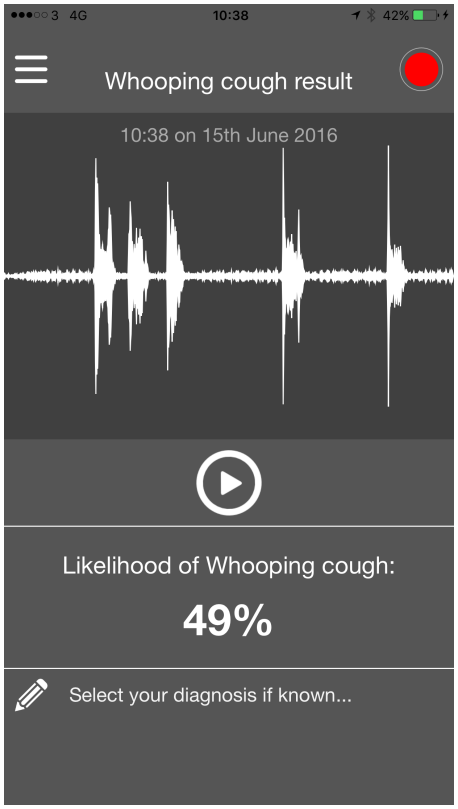
Imperial College London

Department of Electrical and Electronic Engineering

Final Year Project Report 2016



Project Title:	**A New Software Tool for Automatic Identification of Whooping Cough**
Student:	**Daniil Tarakanov**
CID:	**00691892**
Course:	**EEE4**
Project Supervisor:	**Dr Esther Rodriguez-Villegas**
Second Marker:	**Dr Christofer Toumazou**

Abstract

This project concerns design and implementation of an application suite for automatic whooping cough identification and data collection. The suite consists of an iOS whooping cough identification application, an iOS data gathering application, and a web-based administration panel. The main emphasis of this project is on selecting the most suitable whooping cough identification method, and implementing it on an iOS device. Conventionally, whooping cough is diagnosed using laboratory tests, which are time-consuming and unreliable, making such tests impractical during an outbreak. Therefore, automatic identification of whooping cough became a novel area of research, and two solutions have been proposed. This project compares these identification methods, and describes implementation of one such method on an iOS device using Swift programming language. Research into audio-based whooping cough identification is hindered by the lack of available audio recordings. The application suite aims to address this issue by collecting audio recordings, symptoms and diagnoses from the users iOS devices, and uploading this data onto a research server. In this project, an administration panel is also designed and implemented in order to facilitate data visualisation and the identification algorithms evaluation. The resultant application suite shows very promising results for whooping cough identification and data collection.

Acknowledgements

I would like to thank Dr. Esther Rodriguez-Villegas for the advice and guidance throughout this project. I would also like to thank Dr. Anas Imtiaz for the multitude of advice, suggestions that made this project possible.

And of course, I am most grateful to my family and friends for being so supportive throughout this journey, and helping me to become who I am today.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Project rationale and overview

Whooping cough, formally known as Pertussis, is a highly contagious respiratory tract disease that occurs without productive cough and exhibits a whooping sound between the bouts of cough. 172,940 new cases of whooping cough were reported in 2014, and 80,987 deaths were caused by pertussis in 2015, of which 77,162 deaths are attributed to children under the age of 4 years old. Reported pertussis cases and deaths caused by pertussis over the recent history are illustrated in Table 1.1.

Year	Reported cases	Number of deaths	Child deaths (0-4 years old)
2030	Unknown	49 069	46 856
2015	Unknown	80 987	77 162
2014	172 940	Unknown	Unknown
2013	161 889	Unknown	63 137
2012	249 746	67 061	62 571
2011	171 740	Unknown	62 059
2010	160 710	Unknown	61 305
2000	190 475	68 753	60 128

Table 1.1: Actual and projected numbers of pertussis cases and pertussis-related deaths in 21st century [7][8][9]

Pertussis is primarily treated by a course of antibiotics. Macrolide antibiotics are considered

to be the most effective agents for the treatment of pertussis. Specifically, erythromycin is prescribed to infants older than 1 month old, children and adults, while azithromycin is prescribed to infants younger than 1 month old. However, in order for this treatment to be effective, it needs to be conducted within three weeks of the infection. [5] In severe cases of pertussis, exchange transfusion therapy is oftentimes used, as this acts to reduce hyperleukocytosis. This alternative method of treatment has demonstrated the most effectiveness in before the hypotension and organ failure has occurred. [14] [17]

The current gold standard for diagnosing this disease is a culture test, which is performed in a lab over the course of 7-14 days, and has a Sensitivity of 12-60% and Specificity of 100% [6]. This method of diagnosis is only suitable for the first 2 weeks after the onset of cough, which makes it unreliable and cost-prohibitive in the case of outbreaks. An overwhelming majority of the whooping cough cases occur in the developing countries, thus access to specialist equipment and training for diagnosis is scarce, resulting in late diagnosis, and hence in high mortality rate for young children.

With the overall mortality rate approaching 50%, a technological solution for rapid automatic identification of whooping cough is desired, as this will allow for pertussis to be diagnosed in the early stages of the disease, while antibiotic or transfusion treatments are still effective at preventing death. This project aims to address this need by designing and implementing an iOS application for whooping cough identification, aiming to achieve a faster and more reliable diagnosis in cases of pertussis outbreaks.

Whooping cough typically exhibits characteristic paroxysmal cough and whooping sounds that occur when patients gasp for air, therefore this project will attempt to identify pertussis based on these audible features. Development of such identification method would require acquisition of a large number of audio recordings containing these features, in order to improve the potential algorithms training and validation. However, audio recordings are not typically produced during diagnosis and treatment of whooping cough by medical staff, but are instead produced at home by patients or caretakers, and uploaded onto online resources, such as YouTube. This results in a very limited number of recordings, which are not always of suitable quality, and often cannot be

verified. Therefore, a dedicated technological solution for recording whooping cough is needed to facilitate the development of a suitable identification method. This project proposes a suite of applications which is specifically designed to acquire sounds of the whooping cough, along with additional data and upload them onto a dedicated research server. These recordings will be used by the researchers to improve and validate any potential whooping cough identification algorithm.

1.2 Project scope

This project will be based around the design and implementation of a software suite aimed at automatic whooping cough identification, data collection and facilitation of the further research. The software suite is to consist of two mobile applications and suitable back-end infrastructure. The mobile applications are aimed at whooping cough patients, their parents, caretakers and medical personnel. The first mobile application will be able to make audio recordings, automatically identify presence of the whooping cough in these recordings within a reasonable timeframe, inform users of the identification results and upload the recordings and supplementary data onto a research server. This application targets iPhone devices running iOS8 or higher, and is localisable into different languages in order to account for a large proportion of pertussis cases in the developing countries. In order to implement the automatic whooping cough identification function, this project needs to research and compare different methods, and implement the most suitable algorithm. The application needs to be visually appealing in order to encourage user adoption, various functionalities need to be intuitively accessible and the identification results need to be displayed clearly.

The second mobile application will be able to make audio recordings, present medical questionnaires and upload the collected data onto a research server. Similarly to the first application, it targets iPhone devices running iOS8 or higher, is localisable into various languages, and is visually appealing in order to encourage more users to participate in the research study, which should result in more available data. Finally, the back-end infrastructure needs to receive and

store recordings and supplementary data from the multiple instances of the aforementioned applications, while providing a web-based interface for the researchers to access and visualise the collected data.

Due to the limited timeframe of this project, several aspects will be excluded from the project scope. Firstly, the resultant application suite is intended to be a working proof of concept, but is not intended to comply with any applicable healthcare and research regulations, nor with the AppStore guidelines set out by Apple. However, a reasonable effort will be made to adhere to these regulations and guidelines, in order to allow for the application suite to be publically deployed in the future. Secondly, this project will not be evaluated with pertussis patients, due to the lack of adherence to the aforementioned regulations, and a relatively low number of confirmed whooping cough cases in the United Kingdom.

1.3 Report structure

This report first provides some additional background insight into clinical manifestations of whooping cough and existing audio-based whooping cough identification methods in the Background section. It then proceeds to describe the proposed system, specifically any assumptions made, system overview, requirements of each of the three applications, and their use cases in the System description section. Next, the Interface design section provides an overview of visual and interactive elements of each application from the users perspective, while providing justifications for any design decisions.

System design section described the technical architecture of each application in the suite, their components, and provides justifications for specific design decisions. On the other hand, Implementation section dives into specific implementations of non-trivial components within the system, and is split into iOS and web-development subsections.

Testing section describes the initial usability tests in terms of meeting functional requirements of this project, and details results of qualitative and quantitative tests. Evaluation section provides an overview of project outcomes with regards to the initial requirements, and compares the

project to existing alternatives. Finally, the Conclusions and Further work section summarises project outcomes, existing limitations and how they could be addressed in the further work.

Chapter 2

Background

2.1 Whooping cough

Whooping cough is caused by the Bortadella Pertussis bacteria, which is a small, aerobic gram-negative rod [4]. Pertussis has an incubation period of 7-10 days, and the course of the classical illness can be divided into three stages: catarrhal, paroxysmal and convalescent stages, as illustrated in Figure 2.1.a.

The catarrhal stage is characterized by the insidious onset of runny nose, sneezing, mild fever and cold, similar to the common cold [4]. This stage lasts for 1-2 weeks, during which the cough becomes gradually more severe.

The paroxysmal stage lasts for up to 10 weeks, during which the patient experiences bouts of rapid coughs, followed by a high-pitched whooping sound. Following the episode, vomiting and exhaustion commonly occur. Such paroxysmal attacks increase in frequency during the first 1-2 weeks of paroxysmal stage, remain at similar frequency for 2-3 weeks, and then gradually decrease.

The convalescence stage may span from weeks to months, and is characterized by gradual recovery, during which the cough becomes milder and disappears after 1-2 weeks.

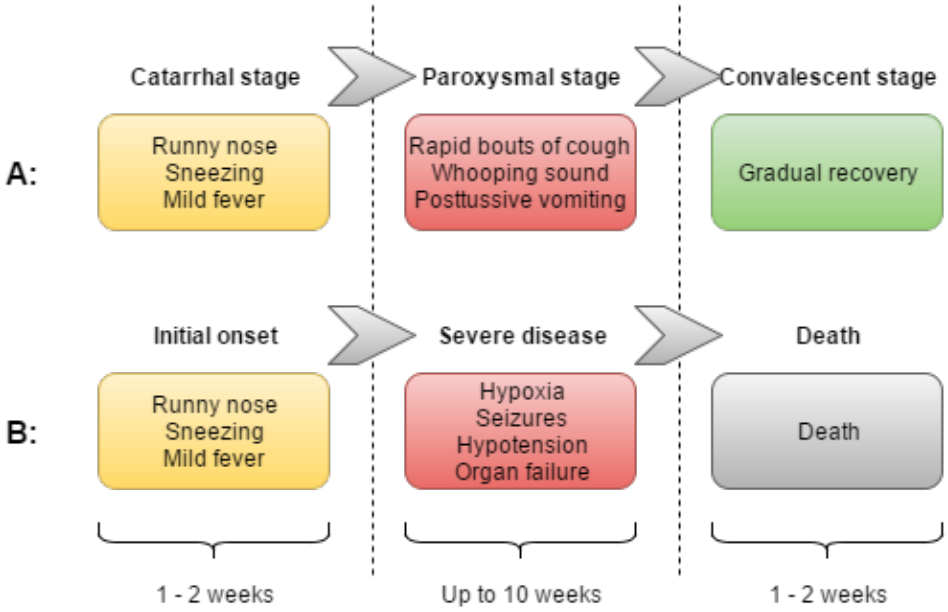


Figure 2.1: Course of disease for pertussis in classical (a) and severe (b) cases

Whooping cough is most severe in infants and young children, although adolescents and adults may also become infected, experience milder symptoms, and infect other children. Children under 6 months of age often experience gagging, gasping for air and vomiting, following a bout of cough.

In severe cases, the paroxysmal stage may develop into hypoxia, seizures, hypotension, organ failure, and in some cases, death.[15] Many pertussis-related deaths are caused by complications, such as the secondary bacterial pneumonia and various neurologic complications, such as seizures and encephalopathy, which often occur as a result of reduced oxygen supply from coughing.[4]

For the purpose of this study, the paroxysmal stage is of the most concern, since the audio features of this stage are the most distinctive for whooping cough, as illustrated in Figure 2.2. Diagnosing pertussis within the first week of the paroxysmal stage could potentially allow for the disease to be treated by antibiotics. If whooping cough is identified in the later course of the paroxysmal stage before organ failure, the disease could still be treated via exchange transfusion.

Typically, coughs follow a three-phase pattern illustrated in Figure 2.3.a, or a two-phase pattern illustrated in Figure 2.3.b. In the three-phase pattern, there are three phases: explosive,

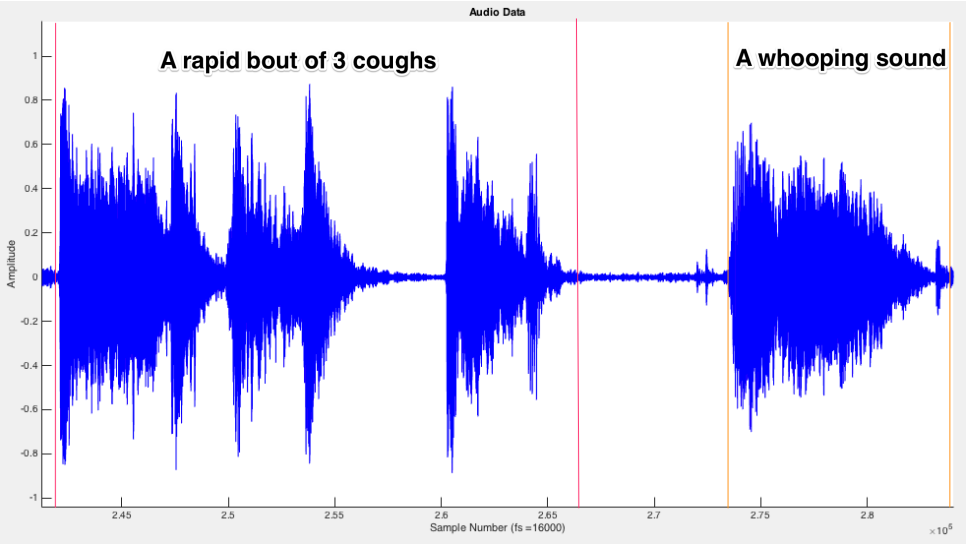
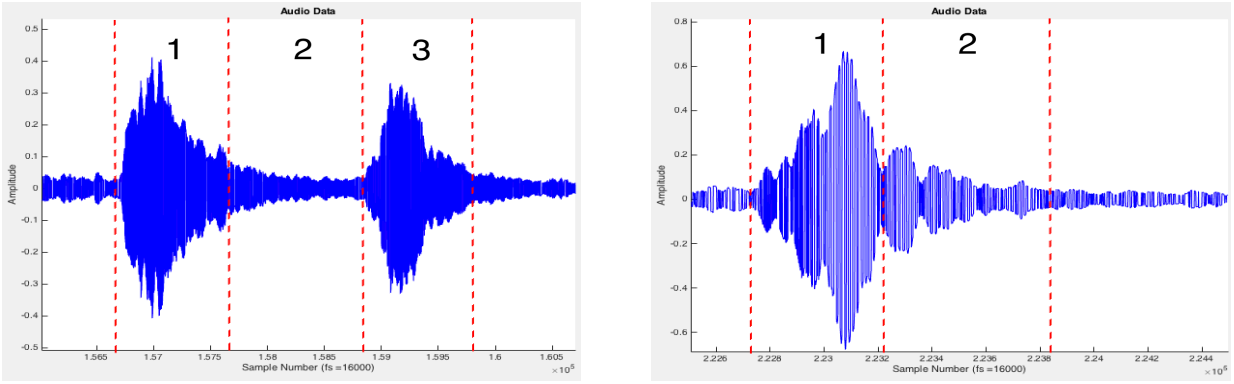


Figure 2.2: Pertussis signal in time domain

intermediate and voiced. The two-phase pattern only consists of explosive and intermediate phases, as the voiced phase is not visible. [16]



(a) (b)

Figure 2.3: a) Three-phase cough signal; b) Two-phase cough signal

Whooping cough is a nonproductive cough, which doesnt bring up any mucous.[16] Thus, the second stage is characterised by a significant reduction of amplitude, as depicted in Figure 2.3.a. On the other hand, productive cough brings up mucus, thus the amplitude of second stage is similar to the other two stages, which is illustrated in Figure 2.4.

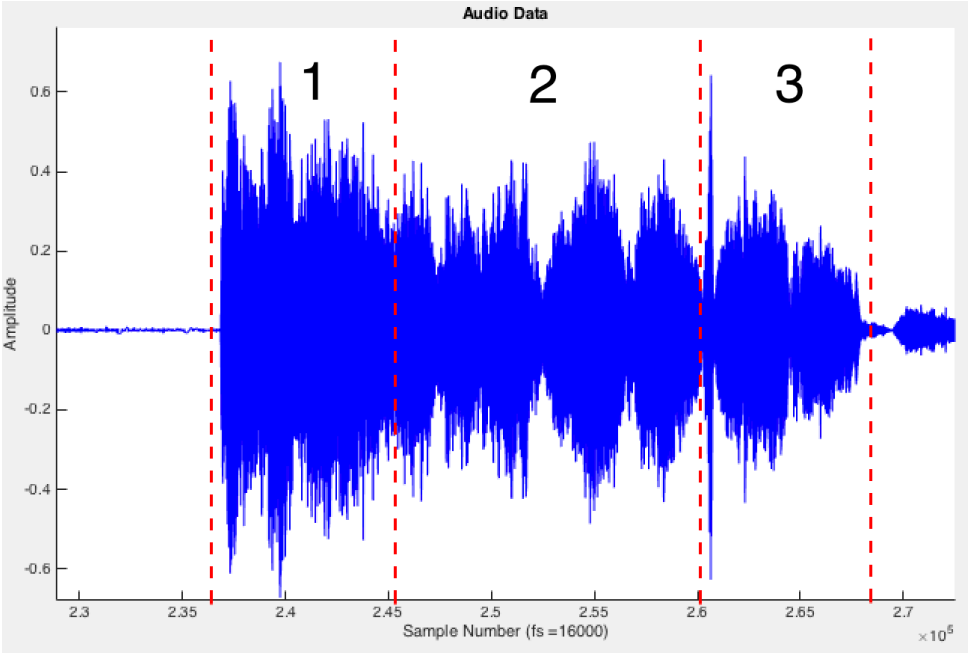


Figure 2.4: Pertussis signal in time domain

2.2 Audio-based whooping cough identification

A study into audio-based whooping cough identification was conducted by Parker et al. in [10], which separately utilised and evaluated 3 classifiers: Feed-Forward Neural Network, Random Forest and K-Nearest Neighbour. In this study, 31 pertussis audio recordings and 16 non-pertussis recordings from the public domain were used, recordings were manually classified, and each cough signal was manually isolated. The proposed algorithm automatically extracted Mel-Frequency Cepstral Coefficients, which were used to train the classifiers.

The NN classifier achieved sensitivity and specificity of 92% and 93% respectively. The RF classifier achieved sensitivity and specificity of 100% and 88% respectively. Finally, the KNN method achieved sensitivity of 100% and specificity of 75%.

Specificity is of paramount importance in the desired automatic whooping cough detection algorithm, as false positive results may cause major health risks, such as unnecessary antibiotic treatments, and a reduced patients trust. Thus, the NN classifier in [10] would be considered an optimal classifier, among the classifiers used in that study. However, this project aims to further improve specificity of audio-based whooping cough detection. Additionally, the study in [10] relied on manual isolation of coughs, which would need to be performed programmatically,

in order to achieve a truly automatic detection, which could be used rapidly by patients and doctors.

A further study into developing an automatic identification of whooping cough was conducted by Pramono in [12]. This approach used a Logistic Regression Model in order to compute probability of whooping cough being present within the supplied recording.

Pramono used sound recordings from various online resources, such as YouTube in order to construct the LRM models and evaluate performance of the identification method. These recordings were manually segmented, and then automatically pre-processed in order to standardise gain and the sampling rate, as well as to remove the silent frames to decrease the processing time.

A number of features were extracted from the recordings, as shown in Table 2.1. These features were initially used build the LRM models for the cough and whooping sound detection, as well as whooping cough classification. The LRM models were trained incrementally, so to minimise the model deviance.

Feature	Domain
Mel-Frequency Cepstral Coefficients	Power
Zero-Crossing Rate	Time
Crest Factor	Time
Maximum frequency	Frequency
Skewness	Frequency
Kurtosis	Frequency
Spectral Standard Deviation	Frequency
Band-power	Frequency
Spectral roll-off	Frequency
Spectral centroid	Frequency
Spectral spread	Frequency
Spectral decrease	Frequency
Spectral flatness	Frequency
Spectral slope	Frequency

Table 2.1: Features used by Pramonos pertussis identification algorithm in[**?**]

In the subsequent recordings, Pramono extracted the aforementioned features and performed automatic segmentation in order to identify segments with cough sound, whooping sound and whooping cough sound. Furthermore, cough rate was counted from the cough segments. This

resulted in 3 indicators, which were combined to produce the overall pertussis identification result. The identification result took a value from 0 to 1, which represented the probability of whooping cough being present in the recording. The identification performance for each stage is given in Table 2.2. Overall, pertussis was identified correctly for all recordings, resulting in a 100% sensitivity and specificity, as shown in Figure 2.5.

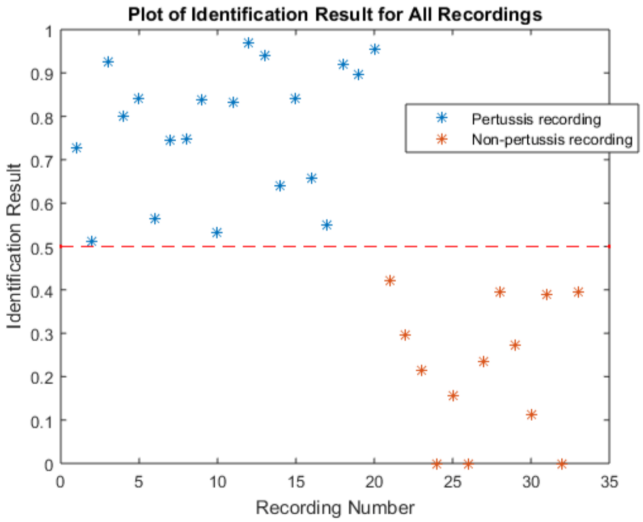


Figure 2.5: Pertussis identification results of Pramonos method[?]

Stage	Sensitivity	Specificity
Cough detection	83%	98%
Whooping sound detection	77%	99%
Cough classification	95.12%	85.42%

Table 2.2: Performance in each stage of Pramonos pertussis identification algorithm in [?]

Recordings used in the evaluation of Pramonos whooping cough detector were collected from the internet, and thus they may be biased to represent cases where whooping cough is evident. More specifically, all of the pertussis recordings present in the data set exhibit the paroxysmal stage, therefore the proposed identification method may not work for other stages. Hence a wider collection of recordings needs to be present to evaluate the identification method. Additionally, pertussis recordings were over-represented in the dataset, thus more non-pertussis recordings are needed to evaluate the specificity of the algorithm. Moreover, it can be seen in Figure 2.5 that the recording 2 was identified correctly with a probability close to 0.5, which indicates that the performance may deteriorate for a larger set of recordings. This was caused by low

83% sensitivity of the cough detection stage, thus improvement in this stage would benefit the overall identification performance.

2.3 Comparison of existing methods

Currently, the gold standard of whooping cough diagnosis is a laboratory based culture test, which is able to diagnose pertussis with sensitivity of 2-60% and specificity of 100% within 2 weeks after the onset of cough [6]. Sensitivity of the culture test decays to 1-3% three weeks after onset of cough [18]. Despite its high specificity, culture test suffers from a long 2-week diagnosis time, and is only useful within the first 2 weeks of cough. Additionally, the culture test is easily influenced by the use of antibiotics, which may further decrease its sensitivity [3].

Another widely accepted laboratory test is polymerase chain reaction (PCR), which is able to rapidly diagnose pertussis with sensitivity of 70-99% and specificity of 86-100% within the first 4 weeks after the onset of cough [6]. Despite PCRs high sensitivity and specificity, it is not a standardised test with over 100 different protocols reported, hence the performance may vary between laboratories [18].

Paired serology is a laboratory test for pertussis, which is conducted at the symptom onset and again 4-6 weeks later. It is able to diagnose pertussis with 90-92% sensitivity and 72-100% specificity [6]. The main disadvantage of this test is the late diagnosis.

Single serology test is optimal 4-8 weeks after the symptom onset and provides sensitivity of 36-76% with specificity of 99%. This laboratory test is usually used as a confirmation of pertussis due to the long intervals between the onset of cough and applicability of this test [6].

The laboratory methods described above require specialist training and equipment, which may not be available during a pertussis outbreak, and are often avoided likelihood of patients contracting pertussis being low, due to the costs involved. This means that a rapid low-cost method of pertussis diagnosis is needed that would be applicable at any point during the paroxysmal stage. In order to achieve rapid automatic whooping cough identification, audio-based methods

described by Parker and Pramono are more suitable than the laboratory-based counterparts. Of these two methods, Pramonos algorithm will be chosen and used throughout this project, as it exhibits a much better performance with 100% sensitivity and specificity. Additionally, Pramonos Matlab implementation was made available for the purpose of this project, which would greatly speed up the iOS implementation process, ensuring that the project fits within the available time frame.

Chapter 3

System description

3.1 Assumptions

Throughout development of this system, several assumptions will need to be made in order to make suitable design decisions, while ensuring that the projects adheres to the given timeframe. I will assume that the users possess an iPhone 4s or newer, running iOS8 or newer. In order to facilitate data collection, I will assume that the users devices are capable of connecting to the internet.

Next, I will be making several assumption regarding the users. Firstly, lets assume that the participants consent to anonymous data collection, and are able to perform recordings in a quiet environment. Secondly, this project assumes that the users have exclusive access to their device, in order to protect their own privacy.

3.2 System overview

In this section, I will provide a high-level description of the proposed system. The software suite consists of three applications: Whooping Cough Detector (WCD), Whoop Research (WR) and Administration Panel (AP). These applications communicate with the back-end server via

HTTP networking, as outlined in Figure 3.1.

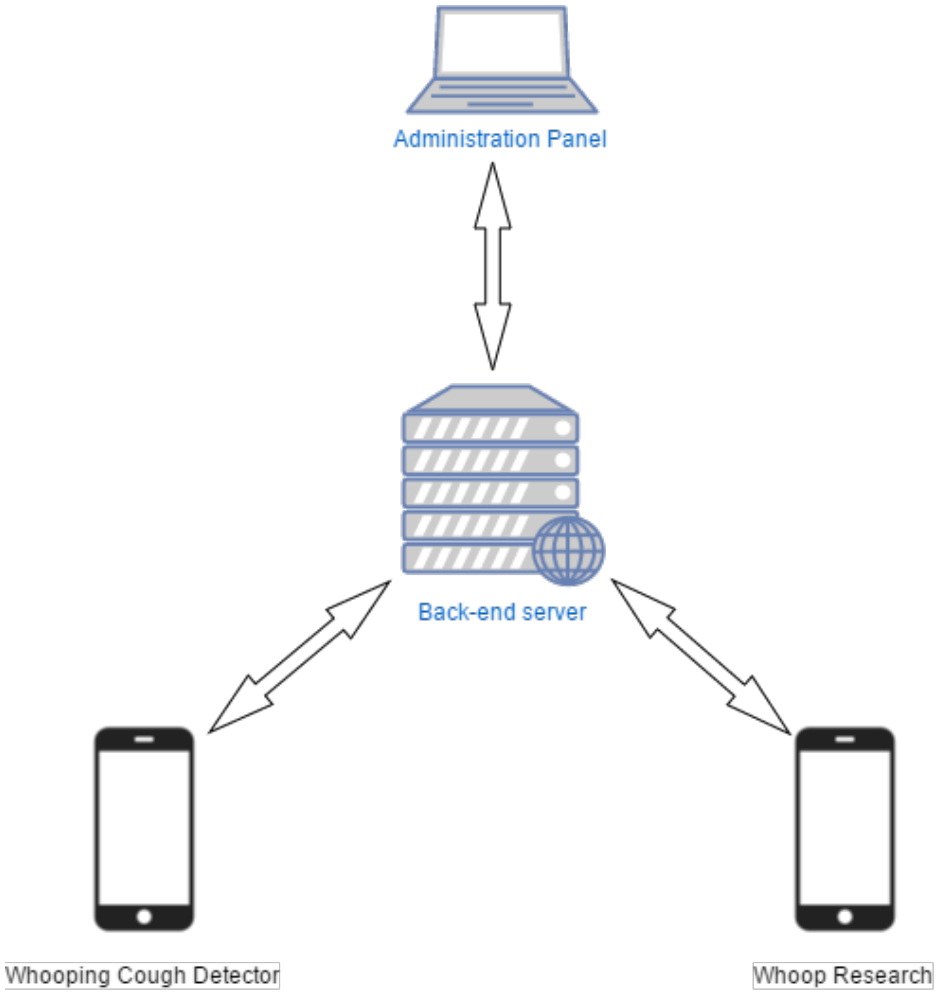


Figure 3.1: System overview

WCD is an iOS application responsible for recording audio, performing whooping cough identification, and displaying of the identification results to the user. It also prompts users to answer questionnaires regarding their symptoms and diagnosis, and uploads the captured data onto the back-end server. WR is a similar iOS application, responsible for recording audio, presenting questionnaires, and uploading data onto the server for research purposes. Unlike WCD, it does not perform whooping cough identification, as it is aimed at collecting data to be used during the development of the identification algorithm.

Back-end server hosts a database for storing and accessing the data collected from the WCD and WR applications, and provides API endpoints which can be accessed by the iOS applications, as well as the administration panel, allowing the client applications to create, read, update and delete stored records. The AP is a web-based tool for researchers to access recordings,

symptoms and diagnoses. It also allows the research team to evaluate the performance of the whooping cough identification algorithm.

3.3 Requirements

This section will discuss the detailed requirements of each application within the proposed software suite.

3.3.1 Whooping Cough Detector

The WCD application needs to be able to record cough sounds, which would be later be used for whooping cough identification. User should be able to enter their symptoms and diagnosis into the application, which could potentially be used for performing whooping cough identification, and are needed to facilitate research into the field. The WCD application should be able to perform whooping cough identification within a reasonable timeframe. Past recordings, symptoms, diagnoses and identification results should be viewable by the user within the WCD application. The user should have an option to delete past results, which serves to keep the list of past records manageable for the user, and allows them to protect their privacy.

Automatic whooping cough identification is an area of continuous research, thus the researchers will be using past results to improve the identification algorithm and release updates. To facilitate this, WCD needs to upload audio recordings, intermediate and final identification results, symptoms and diagnoses onto the research server. This transmission will be performed over the internet, thus it is necessary to encrypt the data prior to transmission, in order to protect it from eavesdropping. Finally, it is expected that the users might not have received a diagnosis from their doctor prior to using this application, thus WCD should be able to schedule notifications to remind the user to enter their diagnosis. This should enable the researchers to validate performance of the algorithm by comparing the identification result with an actual diagnosis.

3.3.2 Whoop Research

The WR application shares several requirements with the WCD application. In particular, it needs to be able to record cough sounds, which will be used by researchers for development of the whooping cough identification algorithm. Additionally, users should be prompted to enter their symptoms and diagnosis in order to assist the research. Finally, the collected data needs to be encrypted and uploaded onto the research server.

3.3.3 Administration panel

AP should enable the researchers to view intermediate and final identification results from multiple instances of the WCD application, as well as the symptoms and diagnoses from both WCD and WR apps. In case of the recordings taken from WCD app, the AP should be able to visualise these recordings, as well as the intermediate results, in order to enable evaluation of the identification algorithm. Researchers should also be able to listen to the recordings acquired from both WCD and the WR applications, in order to manually identify various artefacts. Traditionally, algorithms are developed using Matlab, and only then implemented on the target platform. Therefore, AP should allow the users to download recordings for use within Matlab. Finally, researchers may decide that some records are no longer needed as they may be of little value, thus AP should allow its users to delete individual records.

3.4 Use cases

In this section I will describe some typical use cases for each application, in order to provide an overview of the desired user flow.

3.4.1 Whooping cough detector

Jane and Joe are worried that their child has whooping cough, however the conventional diagnosis methods are time-consuming and cost-prohibitive. The parents wants to verify or disprove their suspicions, so they download the Whooping Cough Detector from the AppStore. They proceed using the app to record their childs cough, enter childs symptoms into the app, wait for a couple of minutes, and receive the identification result through the app. The app tells them, that the recording has 76% likelihood of having whooping cough, thus the parents book an urgent appointment with a doctor, and inform them of this result. The doctor conducts further tests, and informs parents that their child indeed has whooping cough, and begins treating the child. A week after the initial recording, a notification arrives onto Janes phone, asking her to enter a diagnosis which she received from the doctor earlier. Jane swipes notification, the application opens, and she enters the doctors diagnosis. A few weeks after the treatment began, the childs health has improved, although they still experience odd bouts of cough. Jane and Joe continue to worry, so they use the WCD app to update the list of symptoms, and analyse their childs cough. This time, the likelihood of whooping cough is calculated to be 12%. The parents puts their minds at ease, but consult a doctor during the next check-up. The doctor informs them that the child has now recovered from pertussis, however the prolonged illness has irritated the childs throat, which is the reason for continued cough. Jane and Joe can finally stop worrying, and enter an updated diagnosis under the latest recording within WCD app.

3.4.2 Whoop Research

Rita recently started experiencing a very unusual cough, and began suspecting that she has pertussis. She scheduled an appointment with a doctor, who reassured Rita that she has bronchitis. Rita has also learned from the doctor that researchers at Imperial College London are working on a new tool for automatic identification of whooping cough, and are using the Whoop Research application to collect data from patients with suspected cases of pertussis. Rita is excited to assist the research, so she downloads the WR app from the AppStore, records her cough, and enters her symptoms and diagnosis. The WR app uploads this data onto a

server, and thanks Rita.

3.4.3 Administration panel

Researchers at Imperial College London have recently published the WR application, hoping to collect data about suspected cases of pertussis, in order to improve their automatic identification algorithm, and eventually publish the WCD app. One morning, a researcher opens the AP and discovers that a new recording is available. On the first glance, they notice that the patient experienced sore throat, runny nose and intense bouts of coughing, and they were diagnosed with Bronchitis. The researcher then plays the recording and verifies that cough sounds are clearly audible. Eager to test specificity of their identification algorithm, the researcher downloads the recording, and performs whooping cough identification using Matlab, which returns a likelihood of 63A few months later, more people have submitted their data through the WR app, and researchers have improved the identification algorithm to the point where they are comfortable releasing the WCD application. This app becomes hugely popular among worried parents, who use WCD to record and analyse their childs cough. A researcher opens the WCD administration panel, inspects a table of identification results, and finds a record with 76% likelihood of pertussis to be particularly intriguing, as it is somewhat close to the 50% threshold, despite being confirmed as whooping cough. They click on that record, and are presented with a visualisation of the recording and intermediate results. During playback of the recording, they noticed that their algorithm has mistaken several cough sounds for speech. This urges them to focus on improving the cough detection module.

Chapter 4

Interface design

This chapter details the interface design of each application from the users point of view. Each of the three sections is dedicated to one of the three applications, and details UI and interactive elements of various screens inside these applications, as well as provides justifications for design decisions.

4.1 Whooping cough detector

4.1.1 Overview

The WCD application utilises a navigation bar design, as it allows the user to focus on the pages content, making navigating away from the main workflow undesirable to the user. It is in the researchers best interest that the user follows the main workflow, as outlined in the use case on page 17, as this will result in more useful data being collected, while making it easy for the user to diagnose their cough. The navigation bar contains only two items: side menu button, and a recording button. These can be seen in Figure 4.3.b.

Side menu allows the user to view their past records, instructional screen, about screen, and to update their symptoms. This design enables the user to easily access screens outside of

the main workflow, while minimising their temptation to do so, and not occupying the screen real-estate. The side menu is illustrated in Figure 4.1.

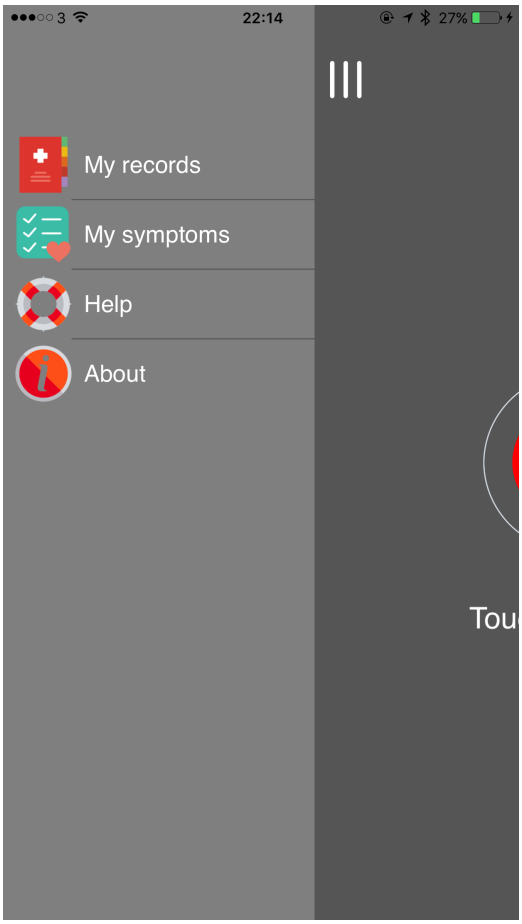


Figure 4.1: Side menu in WCD

The navigation bars recording button enables the user to open the recording screen from any page of application, as this leads to the most desirable workflow, where the user makes a recording, receives their diagnosis, and data gets uploaded onto our server, which facilitates further research and rapid identification of the whooping cough. The recording button is intentionally placed in the top-right corner of the screen, since most people are right-handed, so this placement allows for recording to be felt a more natural action. [CITATION NEEDED]

Throughout the application, the navigation bar never has a back-button. This is done for several reasons: it encourages users to follow the desired workflow, allows space for the side menu button, and is unnecessary as the two buttons in the navigation bar already allow the user to access every part of the application.

The entire application follows a dark colour scheme in order to make the usage easier on the eyes, and to allow for usage of bright colours to ensure that the most important elements, such as the red recording button, stand out. Most text is typed in a white font, as this contrasts well on a dark background, and some of the less important text is typed in a light grey colour.

On the first application launch before the user can begin recording their cough, the application requests users permission to use the microphone, and to send notifications, as illustrated in Figure 4.2. Permissions need to be requested, due to the restrictions set out by Apple. This dialogue takes a form of the pop-up, which is designed to attract users attention, while reassuring them that they will be able to continue with their desired actions once the task is complete. When the user grants the relevant permission, the button changes colour to indicate that the permission has been granted.

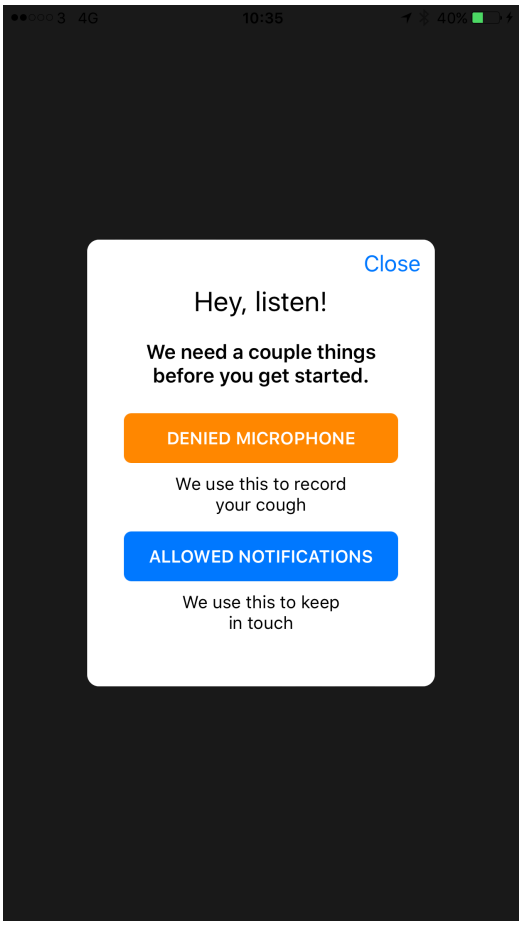


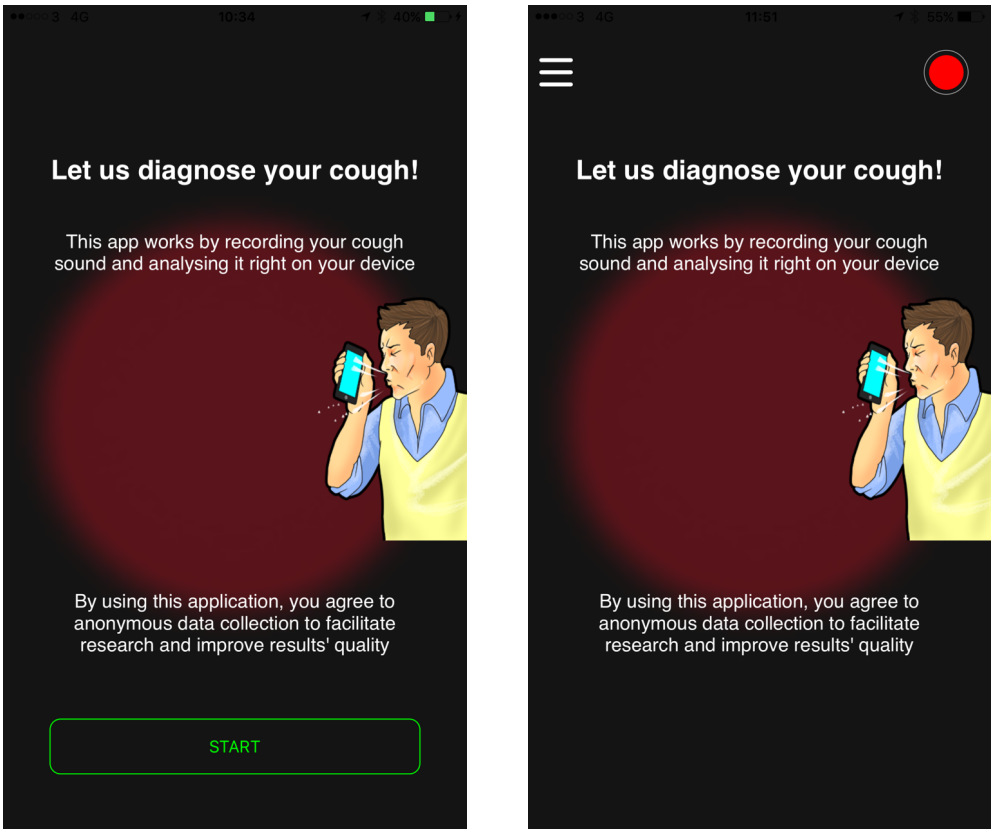
Figure 4.2: Requesting permissions from the user

Before the user produces their first recording, the application starts with an instructional screen, in order to inform the user about the purpose of the application and the intended workflow.

Afterwards, the application starts up with the recording screen to make it easier for the user to begin recording and follow the main workflow. Additionally, before the first recording is complete, the side-menu button is hidden for two reasons: to encourage the user to follow the linear workflow and because there would be no elements to display in the My Records section, and no symptoms to be updated in the My Symptoms section.

4.1.2 Help

This is the instructional screen which is first opened during the app launch before the user makes at least one recording, and is later accessible through the side menu button, as illustrated in Figure 4.3.



(a) (b)

Figure 4.3: Help screen in WCD

This screen features a black background with a blurred red recording button visible in the middle, in order to indicate that the user will be prompted to make a recording later in the app. The screen contains three pieces of text: title, usage description and consent. The title

text reads Lets diagnose your cough!, and is intended to inform the user about the purpose of this application, and to set their expectations. The usage description is located below the title text, and is intended to inform the user about the intended workflow. Consent text is located below the instructional image, and is intended to inform the user that their anonymised data will be used for research. This allows us to collect implicit consent during the first launch of the application.

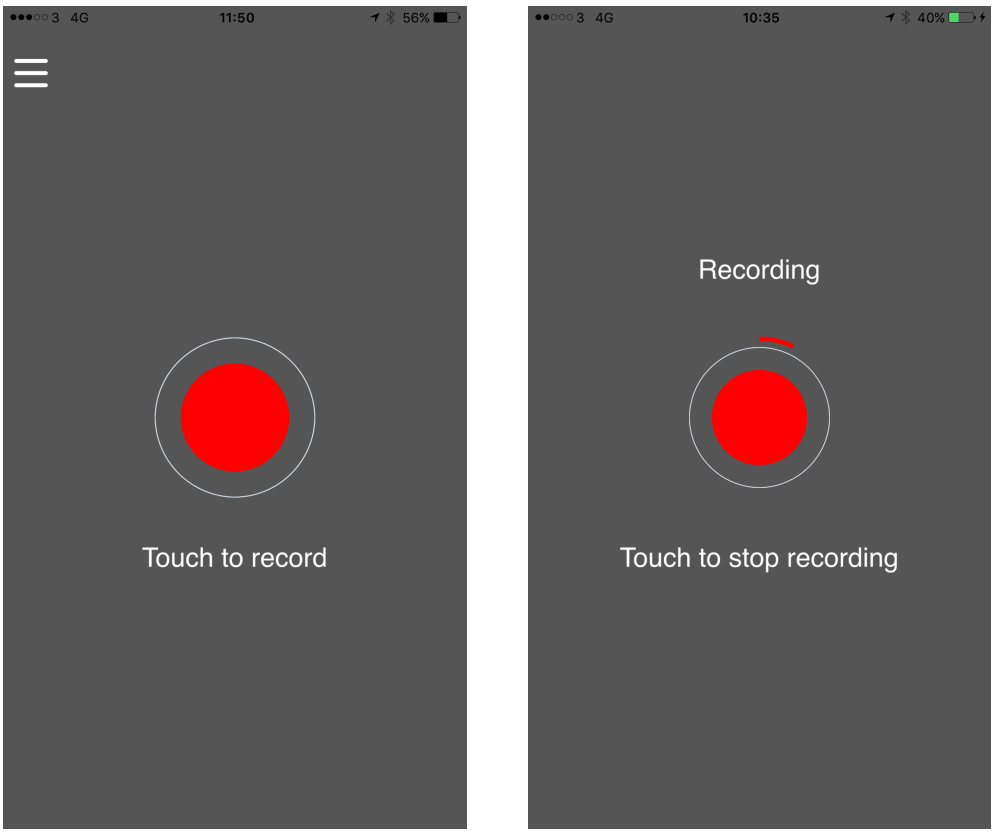
In the middle of this screen, theres an instructional image which further illustrates that the used will be expected to cough into their phone. This image is used in conjunction with the instructional text in order to quickly familiarise the user with the application, and encourage them to move to the recording screen. Finally, before the first recording is complete, the green start button is visible at the bottom of the screen, as illustrated in Figure 4.3.a. In this state, the start button is the only interactive element on the screen, thus it facilitates a linear flow by allowing the user just one choice of interaction. When at least one recording is complete, the start button becomes hidden, and the navigation bar is shown instead, which gives the user a choice of starting a new recording, or navigating to the side menu.

4.1.3 Recording

This is the screen where the user is prompted to record their cough. It features a dark grey background to complement the applications dark scheme, and a large red recording button in the middle, designed to contrast the applications primary colours in order to draw attention to itself. This is illustrated in Figure 4.4.

Instructional text is placed below the recording button, which instructs the user to touch the recording button to begin recording (Figure 4.4.a), and later to stop the recording (Figure 4.4.b). This text has been added after the initial usability tests, since some users didnt realise that they need to press the recording button, thus were unclear about the purpose of this page.

Status text is placed above the recording button, which is only visible when the recording is in progress (Figure 4.4.b), in order to re-assure the user that the application is recording their



(a) (b)

Figure 4.4: Recording screen in WCD

cough sounds.

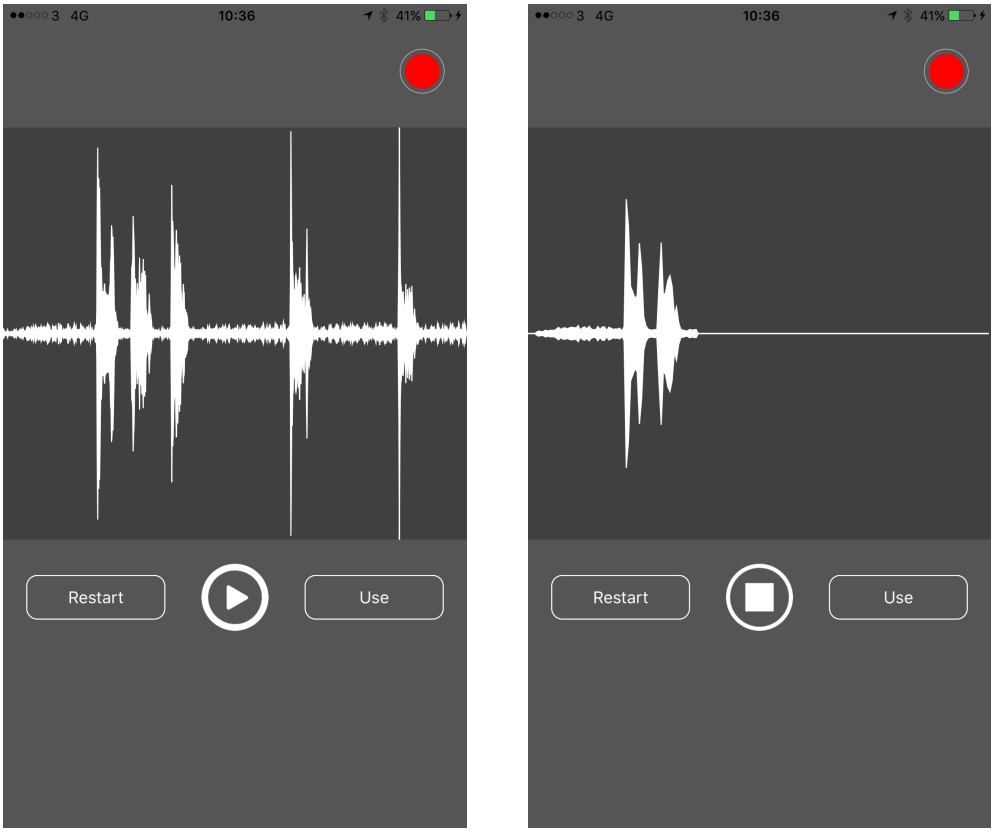
Once the recording button is pressed, it becomes smaller and the recording begins. This is done to clearly illustrate the change of application state, and to emulate the behaviour of a physical button being pressed down. While the recording is in progress, a red progress bar is visible around the recording button to illustrate the progress being made. The progress bar fills up continuously for 30 seconds, as the cough sounds are being recorded. At the 30 seconds mark, the progress bar would make a full circle, the recording would stop, and the user would be directed to the playback page.

The 30 seconds time limit per recording was implemented, as the recordings longer than 30 seconds would take a much longer time to get analysed, which might lead to the user growing tired of waiting for the identification to complete, and quitting the application prematurely. On the other hand, the chosen whooping cough identification algorithm becomes more accurate for longer recordings, so the time limit would inform the user about the expected duration of

the recording. If the user clicks on the recording button before the 30 seconds time limit is reached, the recording stops, and the user is taken to the playback screen.

4.1.4 Playback

This screen is only accessible once the user finishes recording their cough sounds on the Recording screen. It allows the user to visualise and play their recording, as illustrated in Figure 4.5.



(a) (b)

Figure 4.5: Playback screen in WCD

The visualisation is manifested in a plot of the audio waveform on a darker background to indicate an explicit plot area. The waveform itself is drawn in white colour, in order to contrast against the background. This plot reassures the user that the application has recorded their cough, as the coughs tend to be clearly visible, due to a larger magnitude relative to noise and other user-initiated sounds. This is meant to instil confidence in the scientific basis of this application, which should make the identification results seem more trustworthy. Additionally,

the plot is scaled according to the waveforms magnitude, in order to ensure that plot real-estate is not wasted, and the user is able to visualise their recording in as much detail as possible.

The Play/Stop button resides below the plot, and is centre-aligned. This button uses commonly-recognised icons for play and stop actions, and changes its state when the recording is played or stopped. The purpose of this button is to enable the user to play the recording to ensure that the relevant cough sounds are clearly distinguishable. Many applications featured audio playback allow the user to perform three actions: play, pause and stop. However, in this application the pause action is deliberately omitted in order to simplify the workflow and entice the user to move to the next item in the workflow. Due to the 30 seconds time limit, the recordings are usually short enough to ensure that restarting the playback wont take much more time that pausing and resuming.

During the recordings playback, the waveform is plotted in real time, which allows the user to visualise individual segments of their recording, and recognise the typical shapes of cough, whooping, speech and noise sounds. This feature is designed to mentally engage the user in the identification process, which should boost their interest in the application, and go further to ensure that the application exhibits scientific professionalism.

Restart and Use buttons are placed symmetrically around the Play/Stop button, to give the screen a cleaner look. The Restart button allows the user to create a new recording if they are unhappy with the quality or contents of the current recording. Use button redirects the user to symptoms questionnaire screen.

4.1.5 My symptoms

The symptoms questionnaire is a screen which allows the user to indicate which symptoms they have experienced since the beginning of their illness, as illustrated in Figure 4.6. The question is displayed clearly in large font at the top of the screen, so to ensure that this is the first item on the screen to attract users attention.

Symptoms are presented as a multiple-choice list, occupying the whole screen. Multiple selection

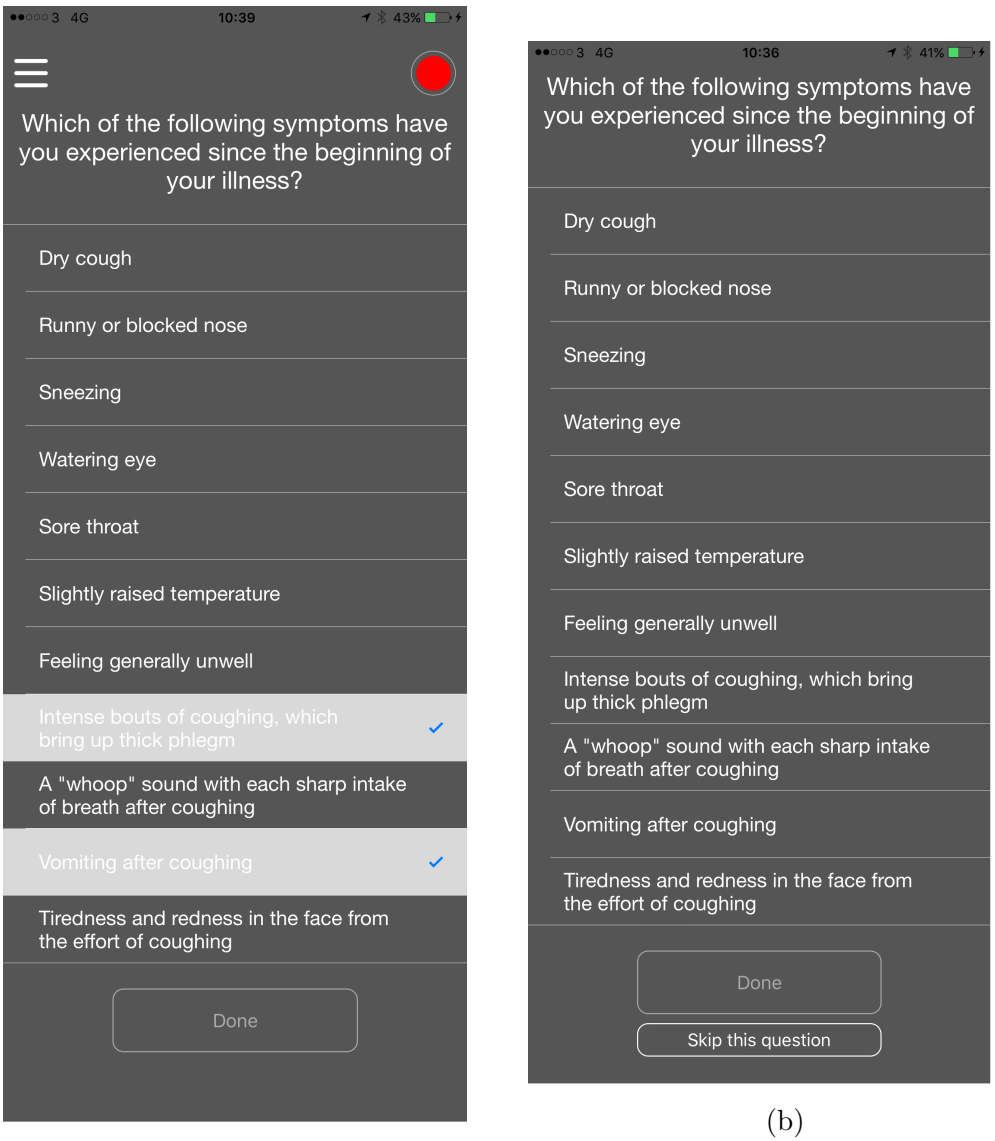


Figure 4.6: My symptoms screen in WCD

is allowed, as users may have several symptoms on the list. When a symptom is not selected, its background colour is dark grey, which is done to create an illusion of the list blending in with the background. When a symptom is selected, it is highlighted with light grey colour, and a checkmark appears on the right side of the item. If the user has previously selected their symptoms, those symptoms will be highlighted automatically, in order to save the user some time, and ensure that they can swiftly proceed to the identification stage.

This screen can be presented either from the Playback screen, or from the side menu, leading to the two distinct states. When the questionnaire is presented from the Playback screen, the navigation bar is hidden to ensure that the user follows a linear flow from the Playback screen

to My Symptoms to Result. Taking that choice away from the user should make it easier to navigate the application with clarity. Additionally, a Skip button is present to allow for the case when the user either has no matching symptoms, or their symptoms havent changed from the previous questionnaire completion. If the user presses the Skip button, they will be redirected to the Result page.

When the questionnaire is presented from the side menu, the navigation bar is presented on the top of the screen, and the skip button is hidden as theres no further action in that workflow. In this state, once the user completes the questionnaire, they have a choice of either going to the Recording screen, or opening the side menu via the navigation bar.

Done button is enabled when a change has been made to the recorded symptoms, and disabled when no change has been made, indicating that theres nothing to save. When the user clicks the Done button, the outcome depends on whether the questionnaire was presented form the Playback screen, or from the side menu. In the first scenario, changes to the symptoms selection are saved, and the user returns to the Playback screen which is blurred out by a dark overlay, with a spinner and a progress indicator, as shown in Figure 4.7. Spinner and the progress indicator aim to inform the user that the identification is in progress, to ensure that the user doesnt quit the application prematurely, and is able to estimate the remaining time for the algorithm to finish identification. The progress is displayed as a percentage, as opposed to the remaining time, as this representation is more concise. Once the identification is finished, the user is redirected to the Result screen. In the second scenario, changes to the user selection will be saved, button becomes disabled to indicate the successful save, and the user remains on the screen, which allows them to choose their next action.

4.1.6 Result

The Result screen informs user about the likelihood of their recording containing whooping cough, as depicted in Figure 4.8. It can be reached in two scenarios: analysis of the recording has just finished, or a record was selected form the My records screen. In the first scenario, details of the most recent record would be displayed, whereas in the second scenario details

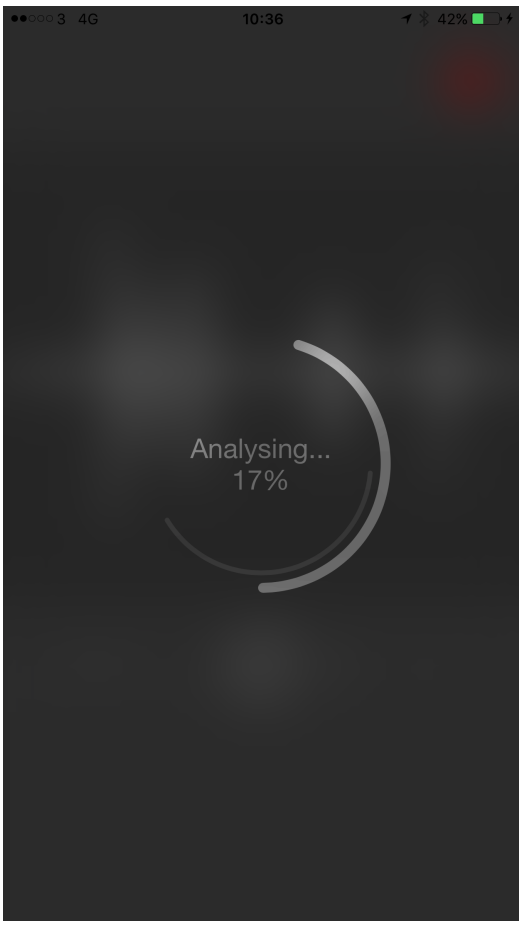
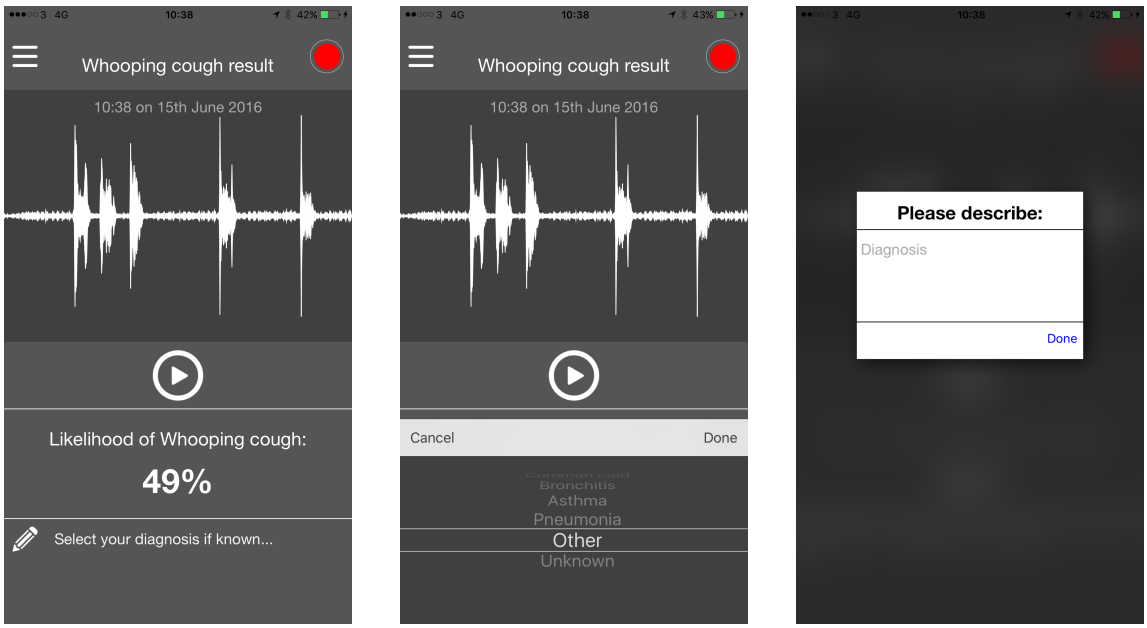


Figure 4.7: Spinner in WCD

of the selected record are shown. This screen inherits the audio visualisation and Play/Stop functionality from the Playback screen, and additional elements are added to inform the user about the identification result, and to allow the user to enter their diagnosis, if known.

The plot on this screen also displays the records date and time in the HH:mm on ddSUFFIX MMMM yyyy format. In this format, HH represents hours in a 24-hour format, mm represents double-digit minutes, dd represents the date, MMMM represents the month in prose, and yyyy represents the year. SUFFIX is dependent on the date, for example 1st, 2nd, 3rd, 4th. An example fo such formatted date would be 12:56 on 4th May 2016. The date and time are displayed in order to provide a confirmation to the user that they have selected the correct record.

Playback features is added as a further confirmation that the user has selected the correct record, and also to enable the user to monitor progression of the disease. Likelihood of whooping cough



(a) (b) (c)

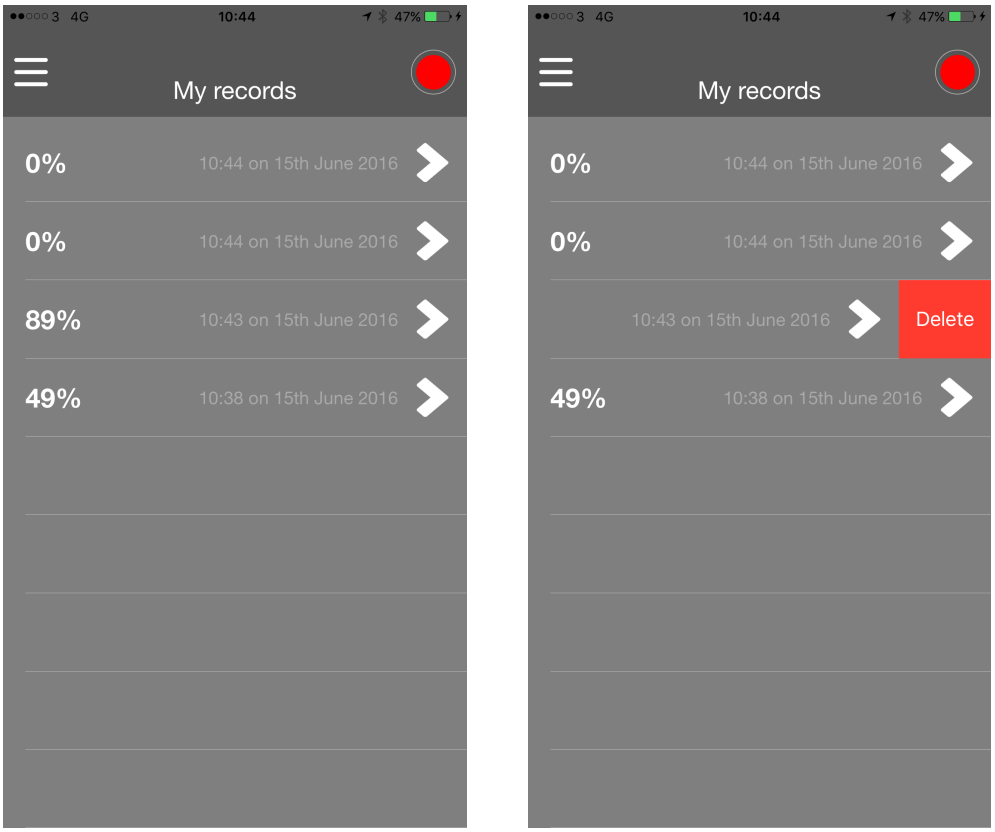
Figure 4.8: Result screen in WCD

is displayed in a bold font, as it is the most important section of the screen, and is designed to grab the users attention. It is represented by a percentage, as this is more commonly understood than a numeric index.

The diagnosis section consists of a pencil and a text field, as illustrated on Figure 4.8.a. The pencil indicates that the field can be edited, while the placeholder text provides clear instructions to the user, indicating that they are expected to select their diagnosis. Tapping on the text field reveals a picker menu, as shown on Figure 4.8.b. The picker menu allows the user to choose one of the pre-set diagnoses, indicate that their diagnosis is not on the list by selecting Other, or indicate that they dont know their diagnosis by selecting Unknown. If the user selects Other, the screen becomes blurred, and a pop-up is shown as depicted on Figure 4.8.c. A combination of blurred screen, and a white pop-up colour is designed to attract the users attention. The pop-up allows the user to enter their diagnosis, if it wasnt on the list. This should result in a more accurate information being recorded by the WCD application, which would assist researchers to evaluate the performance of the whooping cough identification algorithm. Pressing Done button, hides the pop-up, removes the blur effect, and populates the diagnosis field.

4.1.7 My records

This screen displays a list of records previously created by the user, as shown on Figure 4.9. It can only be reached from the side menu, once at least one record is available. Records are displayed in a scrollable list, and sorted by date and time with the newest record being on top of the list. Such layout can accommodate for a large number of records.



(a) (b)

Figure 4.9: My records screen in WCD

Likelihood of whooping cough is indicated in a bold font, as this is the most important piece of information. By the time user reaches this screen, they would have seen the Result page, hence they would have no doubt about the meaning of likelihood percentage. The records date and time are displayed in the same format as the result page, which allows the user to choose a record according to its creation time. Chevron to the right of each record indicated that each element is clickable, so to encourage the user to view their records in more detail, which would improve the likelihood of the user updating their diagnosis. If the user taps on a record, they are taken to the Result page, which is populated according to the selected record. Swiping left

on a record allows the user to delete that record from their phone, as illustrated in Figure 4.9.b, but this does not affect the record stored on our research server. This implementation enables the user to eliminate records which they dont want to have on their device for the cleanliness or privacy reasons, without disrupting the researchers ability to evaluate performance of the algorithm.

4.1.8 About

This screen displays non-critical information about the application, as depicted in Figure 4.11. It contains the apps logo, name, version number, and a number of legal disclaimers and licenses. The Imperial College London disclaimer informs the user about the universitys involvement in the project. The license notices are required to be displayed by the third-party libraries and graphic elements used in the application. The About screen is not part of the desired workflow, thus it is only accessible from the bottom of the side menu if the user is interested.

4.1.9 Notifications

If the user leaves the Result screen without providing their diagnosis, a notification is scheduled to be displayed a week after the recording, as illustrated in Figure **??**. The notifications intended purpose is to encourage the users to inform us about their diagnosis, which would improve the quality of data stored on the research server.

The notification follows Apples standard notification format, and displays the applications logo, as well as a polite question, asking where the user has received their diagnosis. The message ends with a call to action, prompting the user to inform us about their diagnosis. Swiping this notification opens the WCD application on the relevant Result page.

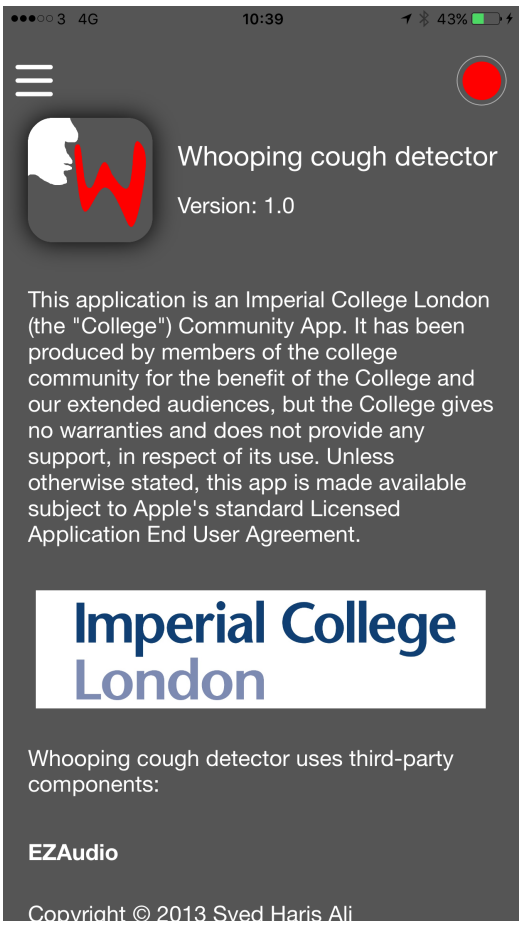


Figure 4.10: About screen in WCD

4.2 Whoop Research

4.2.1 Overview

The WR application is based largely on the WCD application, in order to simplify the development and reuse some of the WCDs design decisions. This applications intended purpose is to collect data for research, thus it sticks to the linear workflow, to encourage the user to record their cough sound, and answer questions, which would be used for research. Unlike the WCD application, the WR application does not have a side menu, as the user is expected to adhere to a very simple linear workflow. In order to enable the user to view the Help screen, the side menu button in the navigation bar is replaced by a Help button. Additionally, WR does not perform whooping cough identification, thus the My records and Result screens were removed. Additionally, the WR application features a separate Diagnosis screen, which is necessary due

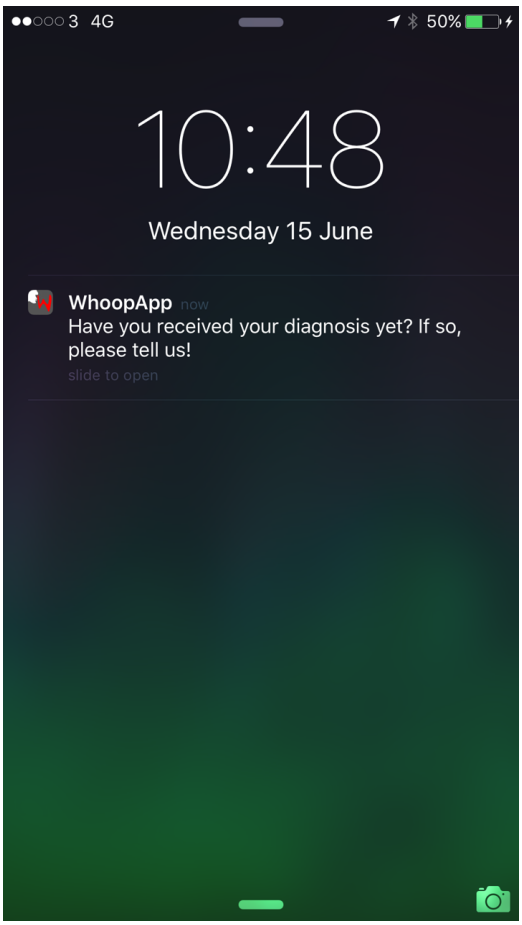


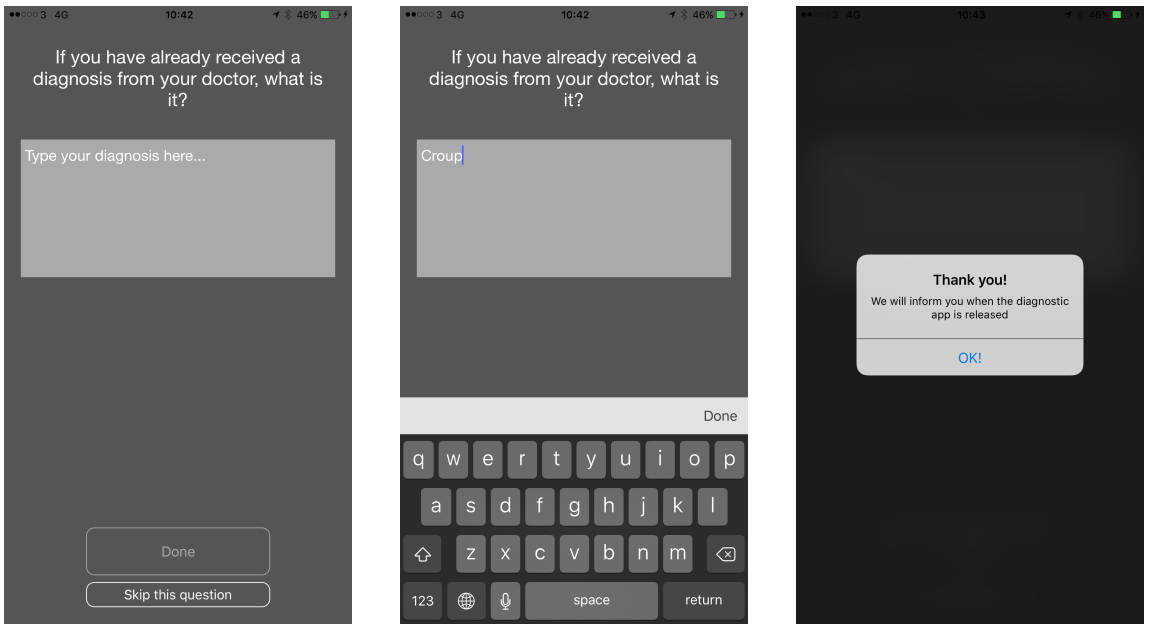
Figure 4.11: Notifications triggered by WCD

to the Result screen becoming obsolete. The My symptoms screen is still present, although it can only be accessed from the Playback screen, as per the desired linear workflow. Absence of the side-menu also enabled me to eliminate the corresponding state of the My symptoms screen, which was shown in Figure 4.6.a. This also means that when the user clicks on the Done or Skip buttons on this screen, they are directed to the new Diagnosis screen.

4.2.2 Diagnosis

The Diagnosis screen represents the last item in the WRs workflow, which prompts the user to enter their diagnosis, if known. It follows the dark scheme, akin to the other screens in the WCD and WR applications, as illustrated in Figure 4.12.

A question is presented on top of the screen in a large font, which is design to attract users attention, when they first reach this page. Below this question, ample space if provided for the



(a) (b) (c)

Figure 4.12: Diagnosis screen in WR

user to describe their diagnosis, which is pre-populated with the previous diagnosis if the user has entered it previously.

Tapping on the Done button saves the updated diagnosis, and shows a pop-up, as shown in Figure 4.12.c. Tapping on the Skip button discards the users changes if any, and shows the same pop-up. The pop-up thanks the user for their participation in the study, and makes a promise to inform the user when WCD application becomes available on the App Store. Once the user clicks OK!, they are redirected back to the recording page, in order to encourage them to make more recordings, which would expand the research teams dataset.

4.3 Administration Panel

4.3.1 Overview

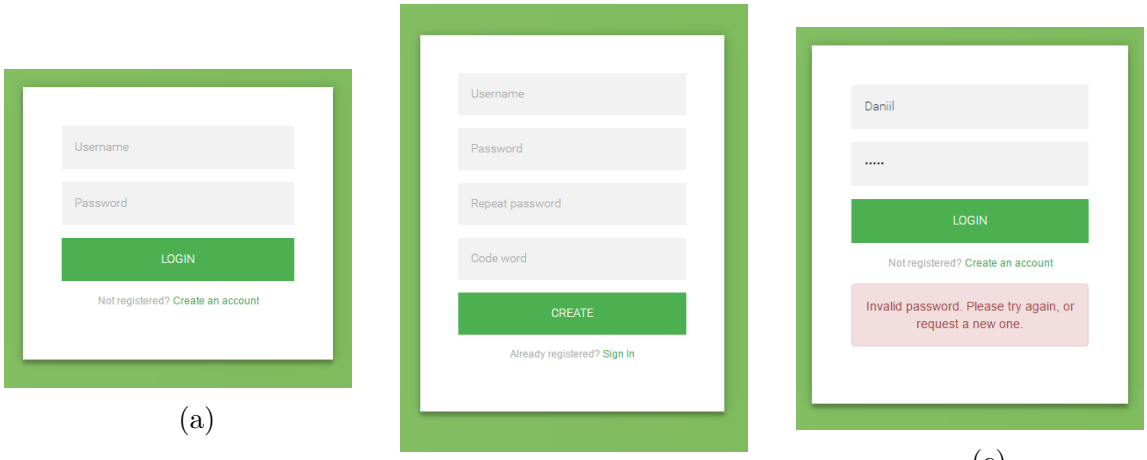
The administration panel is a responsive web application, which can be accessed by the researchers from a multitude of platforms and devices with various screen sizes. This allows researchers to access the panel from Windows, Mac and Linux desktops, as well as from tablets

and mobile devices, which preserves the functionality regardless of the chosen hardware. The AP follows a simple design, allowing the research team to focus on its contents, while ensuring that all of the functionality is visible at glance to ensure that all features are easily accessible.

This application consists of a Login screen, as well as the WCD and WR panels. The Login screen serves as a tool for authentication and account creation, while the WCD and WR panels display the data collected from the respective mobile applications.

4.3.2 Login

This is the first screen which is displayed when a researcher accesses the AP. It contains login and signup forms, as illustrated in Figure 4.13. This screen is designed as a clean non-descript login page with no content description present, as this page will only be used by the research team, who are familiar with the applications purpose, which prevents unwanted attention from any malicious parties.



(a) (b) (c)

Figure 4.13: Login/Signup screen in the AP

Bright page background contrasts with the plain white login/signup form, which is designed to attract the users attention to the form. Placeholders are present to indicate the purpose of each field. Clicking on the link at the bottom of the form in Figure 4.13.a animates the form to add additional fields to enable account creation, as shown in Figure 4.13.b.

The signup form allows the prospective user to choose a username and password. The code

word, which is known only by the research team, prevents malicious parties from creating an account.

If an error occurs during login/signup, an error message field is displayed, as illustrated in Figure 4.13.c. Such error could occur if the user enters invalid credentials, unavailable username, non-matching passwords, or an invalid code word. Successful login or signup redirects the user to the WCD panel.

4.3.3 WCD panel

The WCD panel follows standard Bootstrap design with navigation bar, as illustrated in Figure 4.14. It is used for displaying data collected from the WCD application, such as recordings, symptoms, diagnosis and identification results.

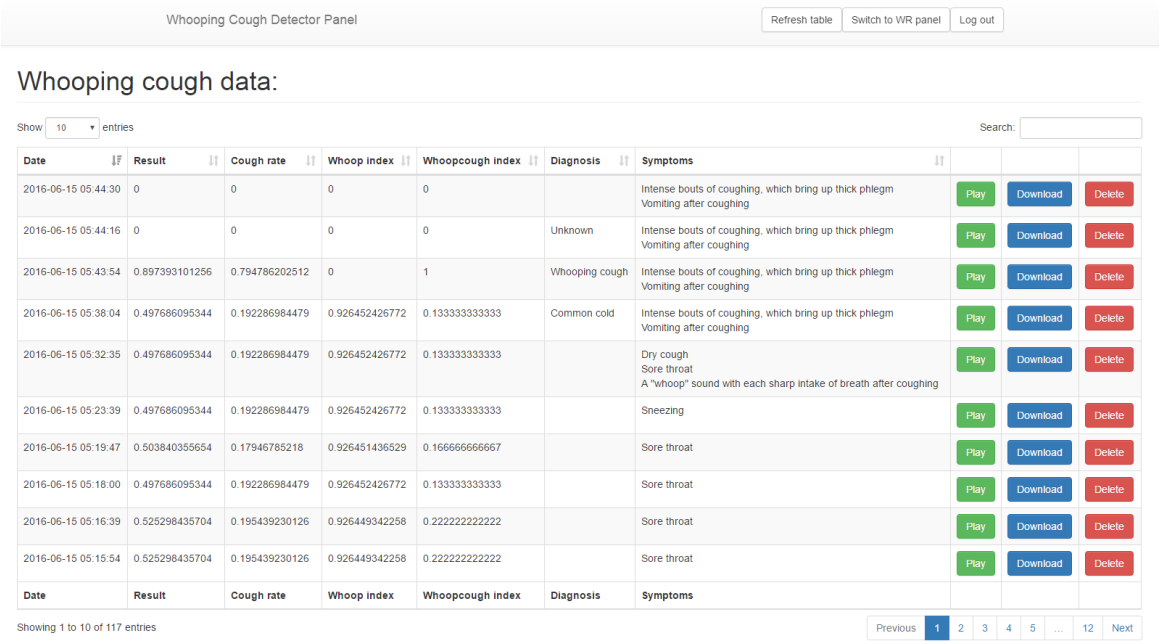


Figure 4.14: WCD panel

Navigation bar contains three buttons: Refresh table, Switch panel, Log out. Refresh table button is used when the user is expecting new data to be added to the table from the WCD application. Switch panel button allows the user to navigate to the WR panel. Log out button allows the user to end their session once they finish using the AP, which is needed to prevent unauthorised access.

The WCD panel contains a table, displaying the data collected from the WCD application, as well as the date and time on which each record was uploaded. The table is sorted by date, with the newest records being on the top of the table. Upload time is displayed, as opposed to the creation time, as the records can be uploaded several days after its creation, thus they might have gone unnoticed by the user if they werent added at the top of the table. This table can also be sorted by other columns, should the user wish to do so. Search functionality is implemented, which allows the user to filter the table according to desired values, for example all records containing sore throat. Pagination on the table ensures that the user is presented with a manageable amount of data per screen, with the default number of entries per page set to 10, which can be increased to 100 entries per page.

Delete buttons next to each entry allow the user to delete the record if it is deemed to be not useful to the study, which is usually the case if the record is too short, too noisy, or does not contain any audible cough sounds. When this button is clicked, a confirmation dialogue appears to warn the user that the deletion is not reversible, encouraging them to think twice before deleting the record, as illustrated in Figure 4.15.

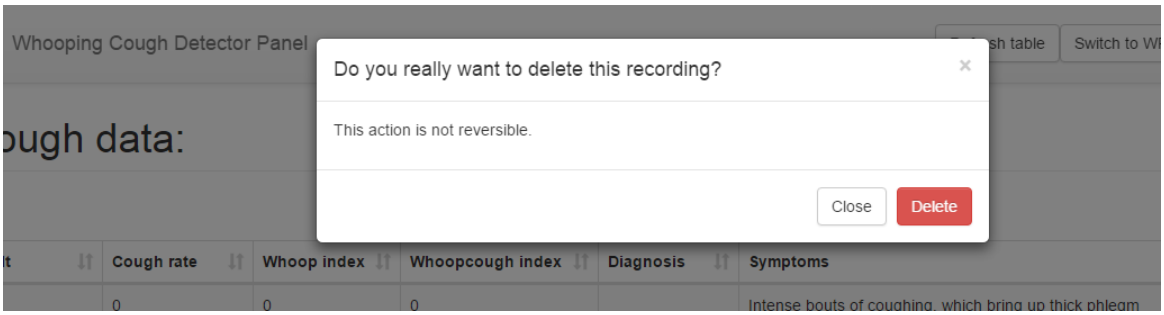


Figure 4.15: Deletion dialogue in AP

Download buttons next to each entry allow the user to download the audio recording, in order to use it in further development and evaluation of the WCD algorithm.

Play buttons next to each entry are styled in bright green colour to attract attention to one of the key features of this panel. When clicked, a spinner overlay is displayed to indicate that the content is rendering.. Once the content finishes rendering, the playback screen is displayed, as per Figure 4.16.

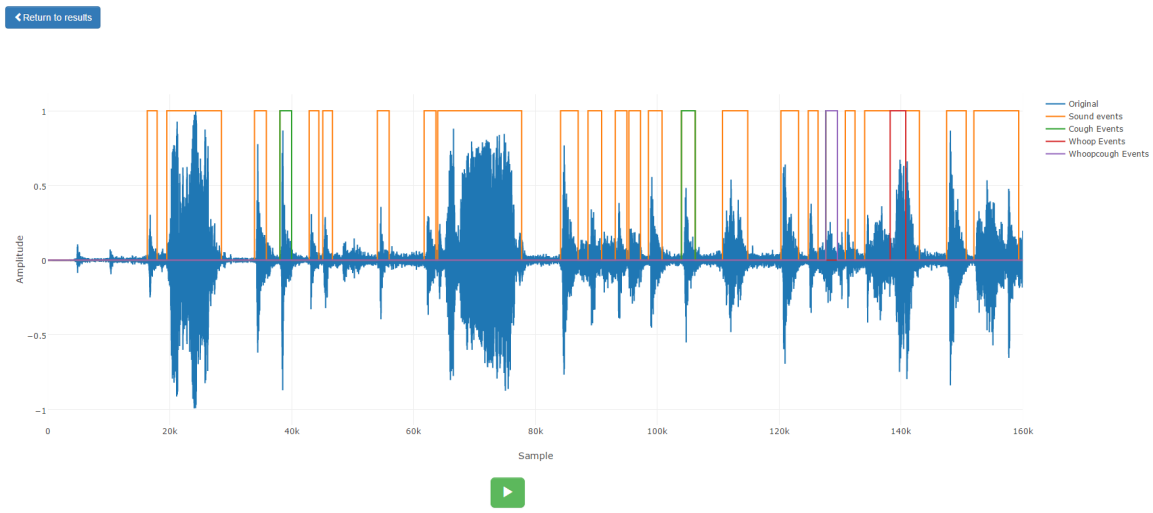


Figure 4.16: Playback screen in AP

The two main features of the playback screen are the audio visualisation and the Play/Stop button. The visualisation occupies most of the screen, in order to allow for better visibility of its details. It contains an audio plot, where the horizontal axis represents sample number, and the vertical axis represents the normalised magnitude. This plot allows the researchers to visually examine various features of the signal. Additionally, this visualisation contains several rectangular overlays, representing detected sound, cough, whooping cough and whooping sound events. These overlays enable the researchers to evaluate individual components of the algorithm, for example they may notice that some cough events went undetected, allowing them to focus on improving sensitivity of the cough detection module. The overlays can be toggled to allow for better visibility for overlapping events. For example, a whooping cough event occurs simultaneously with a cough event, which is also a sound event, thus the researcher may choose to hide the whooping cough overlay, in order to focus on detected cough events. A legend to the right of the plot indicates the significance of each overlay via colour-coding. The visualisation also allows for zooming and panning, which enables the researchers to gain a better insight into the algorithm performance and the underlying waveform features.

Play/Stop button is located right underneath the plot, and provides a similar functionality to the Play/Stop button on the WCDs Playback screen. It uses the commonly understood icons for playing and stopping of the recording, as well as adopting green and red colours. Playback of the recording allows the researchers to manually identify different events, and compare them

to the algorithms output.

4.3.4 WR panel

WR panel follows the same styling as the results panel, to ensure seamless transition between the two panels. This panel displays various data collected from the WR application, as depicted in Figure 4.17.

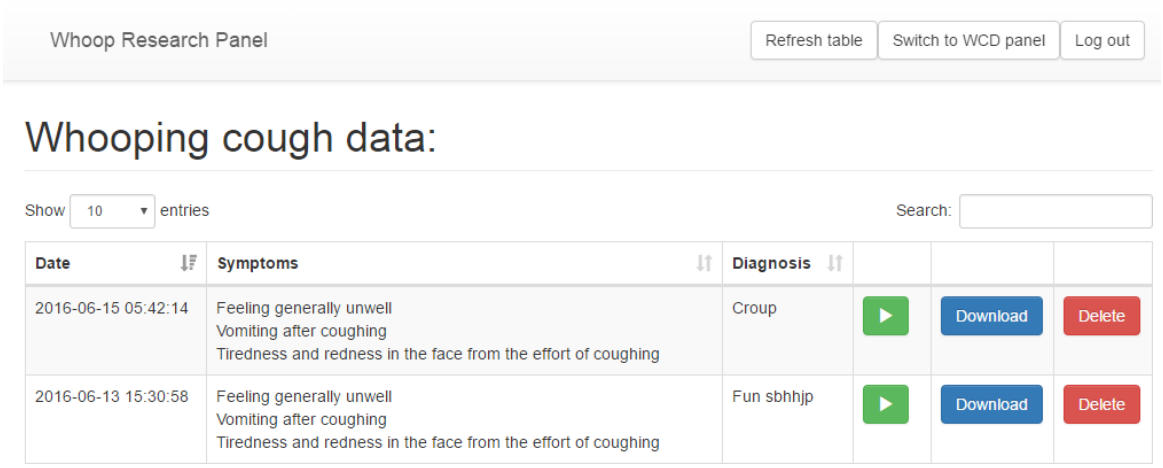


Figure 4.17: WR panel

The main user interface difference between the WR and WCD panels stems from the fact that the WR application does not perform whooping cough identification, thus the identification results arent displayed, and the Play button is replaced by a Play/Stop button, which inherits its behaviour and functionality from the Play/Stop button on the WCD panels playback screen.

Chapter 5

System design

This chapter will detail and justify architecture for each application in the software suite, and describe the technical design of each component. The previous chapter described the system from the users perspective, whereas this chapter will be more focused on data flows within the system.

5.1 Whooping Cough Detector

5.1.1 Architecture

The WCD application follows a Smart client, dumb server model, where intensive computations are performed on the device, white the back-end server only handles the database operations. This design paradigm enables seamless offline operation without relying on network connectivity and servers responses. Majority of whooping cough cases occur in the developing countries, thus it is expected that the users will have limited internet connectivity, which makes this design most suitable for the intended audience. Additionally, this model allows for a virtually unlimited scalability, as the growing number of users will result in a negligible increase in servers load, while most computations are performed on the users devices.

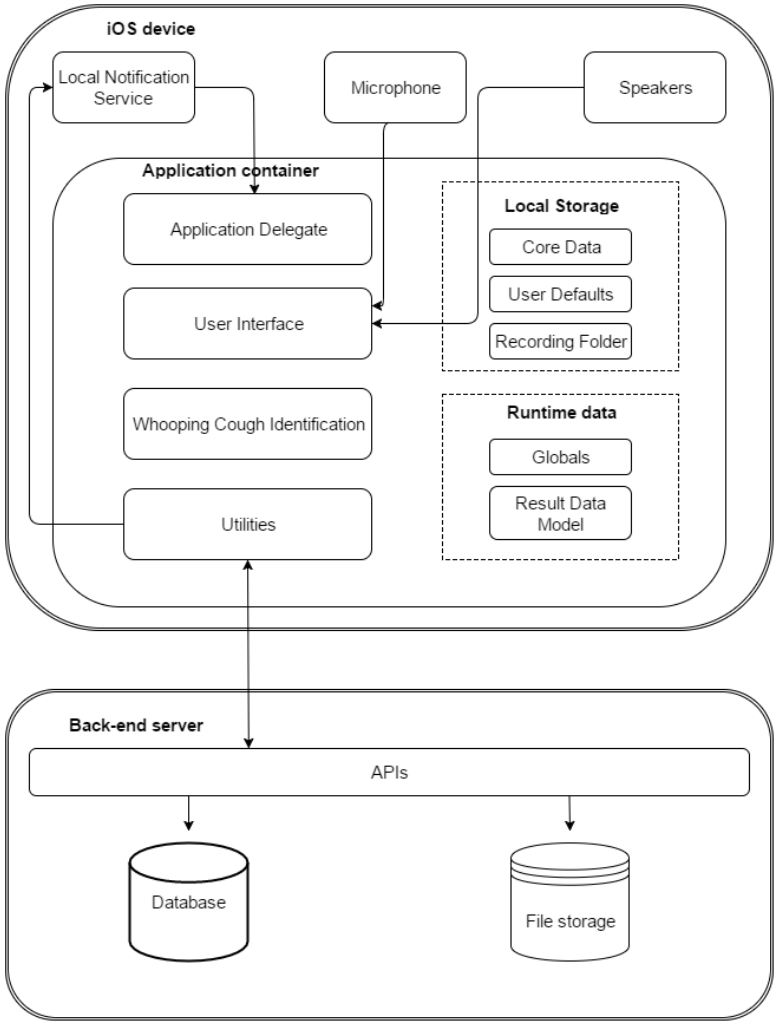


Figure 5.1: WCD architecture

The WCD architecture can be separated on the top level into Application Container, Device Services, and Back-end server, as illustrated in Figure 5.1. Application Container includes software modules and data storage implemented by the WCD application, while the Device Services include software and hardware provided by the iOS, which can be accessed from the Application Container via the iOS APIs. Back-end server resides in the cloud, and exposes APIs to handle WCDs requests via HTTP.

Application Delegate is responsible for handling iOS events which are outside of the applications control, such as application start, or receiving a local notification. When the application starts, the Application Delegate makes a request to Utilities module in order to check if any records are stored within the Core Data. If this is the case, it sets the corresponding flag in the Globals module, and ensures that the Recording screen is displayed first, as opposed to the Help screen,

when no records are stored. Additionally, if at least one record is stored in Core Data, the Application Delegate makes a request to the Utilities module to check if any records present on the device are missing from the server. If this is the case, it uploads the missing data onto the server, according to the logic described in the system design section. If the application is launched by the user swiping a notification, the Application Delegate will ensure that the Result screen is populated with a record which caused the notification. When the application is running in background, and a local notification is received from the Local Notifications Service, the Application Delegate will preload the appropriate Result screen, to enable the user to swiftly update their diagnosis if they move the WCD application to the foreground. High-level functionality of the Application Delegate is illustrated in Figure 5.2.

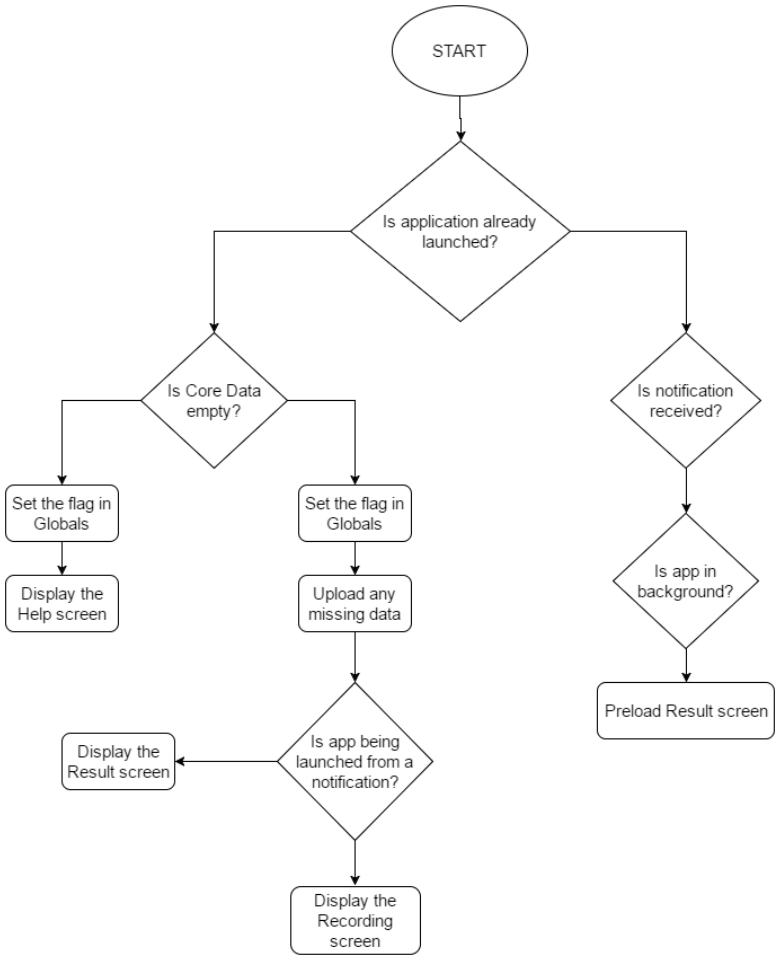


Figure 5.2: Application Delegate Logic

User Interface module contains logic necessary for rendering any screen displayed to the user. It communicates with the Local Storage via the Utilities module, and communicates directly with the Runtime Data, in order to populate the screens with correct content. It is also responsible

for communicating with the devices microphone and speakers for recording and playback of the cough sounds.

The whooping cough identification module provides one interface for the UI module to begin the analysis, and also communicates with the Result data model to store the results in runtime, locally and upload results onto the server.

Result data model is a structure which is used to represent a specific record, containing a link to the recording file, symptoms, diagnosis and the identification results.

Utilities module contains a collection of state-independent reusable functions, which can be called from any other part of the application. These functions perform various tasks, such as mathematical calculations, networking and data storage.

Core Data is an internal database, which is used to locally persist instances of the Result data model, corresponding to the users previous records. User Defaults is a dictionary, which is used to locally persist simple individual objects, such as symptoms. The recordings folder is persisted within the application, and contains all of the users audio recordings. Globals module contains a collection of application-wide non-persistent variables, such as classification coefficients, application state flags, and intermediate results.

For the purposes of WCD application, the iOS device provides three services: microphone, speaker and Local Notifications Service. Microphone and speaker communicate directly with the WCDs UI module, whereas the Local Notification Service receives scheduling instructions from the Utilities module, displays triggered notifications on the device, as well as any paired devices, such as Apple Watch, and sends appropriate notifications to the Application Delegate.

Back-end server exposes RESTful APIs which enable the WCD application to interact with the servers file storage and database. The file storage is used for storing audio recording files and JSON filed with visualisation data. Database is used to store symptoms, diagnoses and identification results from all application instances.

It is expected that the application will be used in many different countries, whose primary language is not English. Therefore, WCD needs to be developed in a way which enables

localisation by non-technical translators.

5.1.2 Whooping cough identification

This section will detail the operation of the whooping cough identification module. The identification is performed on the users device, and provides an analyse interface for the UI module to trigger the analysis. The analyse interface takes the recordings local URL as an input, and provides a completion handler to inform the UI that the identification has been completed. Such design ensures the applications modularity, which is necessary to enforce separation of concerns, where this module is solely responsible for the whole identification process, and other modules do not become bloated by this process.

This module is initiated by the Playback controller, once the user has pressed the Use button, and completed the symptoms questionnaire. The identification is performed according to Pramonos algorithm, as outlined in Figure 5.3.

Pre-processing step ensures that the data is normalised, in order to account for different recording environments, such as background noise and distance between the user and the device.

The sound event detection step is used to extract useful segments from the recording, such as cough sounds, whooping sounds and speech. This is designed to eliminate noise and silent segments, which improves performance and the processing time.

The features extraction step computes audio features, as outlined in Table 2.1. These features will later be used to perform cough, whooping cough and whooping sound classification.

Classification steps are performed by utilising Logistic Regression Model (LRM), and result in an index ranging from 0 to 1. Whooping cough classification is only performed after cough sound classification has been completed, as it is expected that the whooping cough events can only occur where the cough sound is present. Cough rate is also computed once the cough sound classification has been performed.

In the last step, the three indices are weighted, in order to obtain the final result index, also

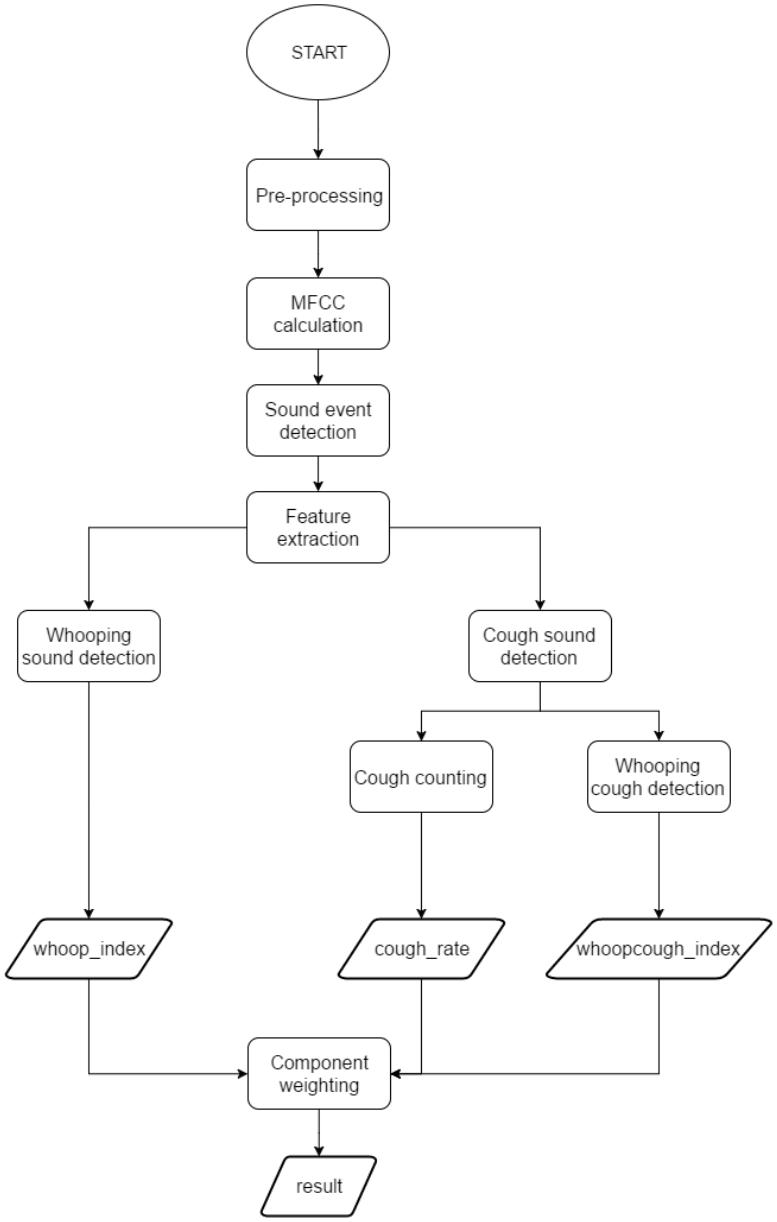


Figure 5.3: Whooping cough identification algorithm

ranging from 0 to 1.

Once the result is computed, it is stored in a global instance of the Results object, along with the three contributing indices, normalised waveform in an array format, as well as cough, whooping cough and whooping sound events in a Boolean array format, where 0 represents the absence of corresponding event, and 1 represents presence of the corresponding event.

Global instance of the Result object is only stored in the runtime, thus it is not persistent. Hence, the objects store method is also invoked to ensure that the record is stored both locally in Core Data, and remotely on the research server.

5.1.3 Class model

Persisted data from the WCD application takes a shape of a Result class model. Class modelling is a widely used design pattern in Object Oriented Programming, as it allows to keep related variables in one place, thereby providing separation of concerns, and resulting in a generally cleaner code. This concept also becomes useful when dealing with persistent data in local and remote databases, as each row in a database could be stored as a class, and a collection of classes may represent the whole database.

The Result class model contains properties, as listed in Table 5.1

Property	Type
local_id	Int
id	int
filePath	NSURL
dateTime	NSDate
whoop_index	Double
cough_rate	Double
whoopcough_index	Double
result	Double
origSound	[Double]
soundEvents	[Bool]
coughEvents	[Bool]
whoopCoughEvents	[Bool]
Diagnosis	String

Table 5.1: Properties of the Result class.

localID is a record identifier, which is unique to each device, but not the overall system, as the latter would require internet connectivity, which may be scarce among the target audience. It is used within the WCD application to differentiate between records, and also to inform the Result screen about the record to be displayed.

Id is a record identifier, which is unique to the overall system. It serves as a primary key in the back-end servers database, which allows the APIs to find individual records in order to update them if needed. The default value of id is -1, which signifies that the identifier has not yet been created. This identifier is generated on the back-end server, thus requires internet connectivity to be set.

filePath is a local URL which specifies the path to a corresponding audio recording.

dateTime represents the date and time of the objects creation, which is then used by WCD to sort records on the My records screen, and by the user to differentiate between individual records.

Whoop_index, whoopcough_index, cough_rate and result are indices generated by the whooping cough identification module.

origSound, soundEvents, coughEvents and whoopCoughEvents are the arrays representing normalised audio waveform, and detected audio events, as described in the previous section.

The Result class contains one public method: store(). It is used to store the current instance of Result both locally on the device, and remotely on the back-end server.

This class also contains one private method: storeLocal(), which is used to store the current instance locally in CoreData, and is called from the public method store(). This represents a Facade design pattern, whereby only one unified API is exposed by a class, which is then able to handle any internal logic.

The Result class is used throughout the WCD application. When the application is launched, a global instance of the class is created, which is then used during the Recording Playback Result flow. During the initialisation of My records page, an array of Results is used to store all records from CoreData, with each instance of Result representing a row in CoreData. Each row on the My records page is populated using this class result and dateTime fields. During the initialisation of Result screen, an instance of Result model is created by fetching the appropriate record from CoreData by the local_id field, and the screen is then populated using filePath, dateTime, result and diagnosis fields.

5.1.4 Local data persistence

In order to enable seamless offline operation, and account for poor internet connectivity, it is necessary to persist collected data locally. WCD application achieves this using three technolo-

gies: Core Data, User Defaults and local storage.

Core Data is Apples framework, which uses a SQLite database as a persistent store of managed objects, representing entities. Entities have attributes, which are akin to properties in data classes that were described in the previous section. In the WCD applications design, one Result entity exists, which can directly mapped to the Result class model. This configuration enables WCD to fetch the entire database as an array of Result objects, as discussed in the previous section.

However, attributes in Core Data are unable to store non-primitive objects, which makes it non-trivial to store the filePath property and properties which are arrays. In the case of filePath, it is necessary to convert the property into a string containing the files URL during storage, and vice versa for data retrieval. Similarly, the arrays, such as origSound need to be converted into binary data for storage, and then back to arrays for retrieval.

NSUserDefaults is a dictionary which is persisted in the local memory. It is generally used to store simple single objects, such as integers and strings, as it avoids the overhead from using Core Data. The WCD application uses NSUserDefaults to store the users symptoms. However, each symptom is represented by an integer, thus in order to enable storage of multiple symptoms, an array of integers is converted into a dash-delimited string. For example, [1, 2, 3] is converted to 1-2-3. In this case, if symptoms were stored in Core Data, it would always contain just one instance of the symptoms entity.

Finally, local storage is leveraged to store the audio recordings, as this allows the recordings to be stored directly as WAVE files, which are then used directly to visualise, play and upload these recordings. Using this method, the files are stored within the Documents directory of the WCDs application container, and references to these files is maintained via Results filePath property. However, the application container identifier changes with each application launch, thus the file references from the previous launches become invalid. Thus, the Utilities module implements a helper method, which updates these references with the current container identifier. In order to ensure the file uniqueness, all files are named according their creation date and time, which never repeats.

5.1.5 Networking

In the context of WCD application, networking refers to any interaction between the application and the back-end server. It is used for uploading data onto the server and checking whether any data is missing from the server.

Hypertext Transfer Protocol (HTTP) is used for communication over the internet between a client and a server. In our case, WCD app is the client, which initiates communication, while the back-end server responds. Thus, a request from the client, and a response from the server are interchanged. HTTP messages contain a header and a body. Header contains metadata, such as description of its contents, authentication and expected response type. Body would then contain any data that needs to be transmitted over the network. HTTP also defines a set of request methods, such as GET, POST, PUT, DELETE, and many more.

The WCD application will be utilising POST requests to transmit data to the server, and the server will be replying with responses that will contain a status code, as well as any additional data. In order to achieve this communication, the back-end server implements RESTful APIs. Automatic Programming Interface (API) is a set of publically facing functions which allows third parties to interact with the application. In our case, the APIs reside on the back-end server, and allow the clients to upload records, and check for missing data. Representational State Transfer (REST) is a widely used architecture for constructing APIs. It is lightweight, which reduces bandwidth requirements, and decoupled, which allows for communication between applications utilising different technologies and languages.

In the context of WCD application, once the identification has been performed, Results store method is triggered, which is used to store the latest record both locally and remotely. The uploading process is outlined in Figure 5.4.

Each record is uploaded in three parts: primitive data, large arrays, and audio file. Separation of audio file upload into a separate request was necessary, because once data has been encrypted, it is transmitted as a stream of bytes, thus it wouldnt be possible to differentiate where JSON data ends, and audio file begins. Arrays are also transmitted separately from the primitive data,

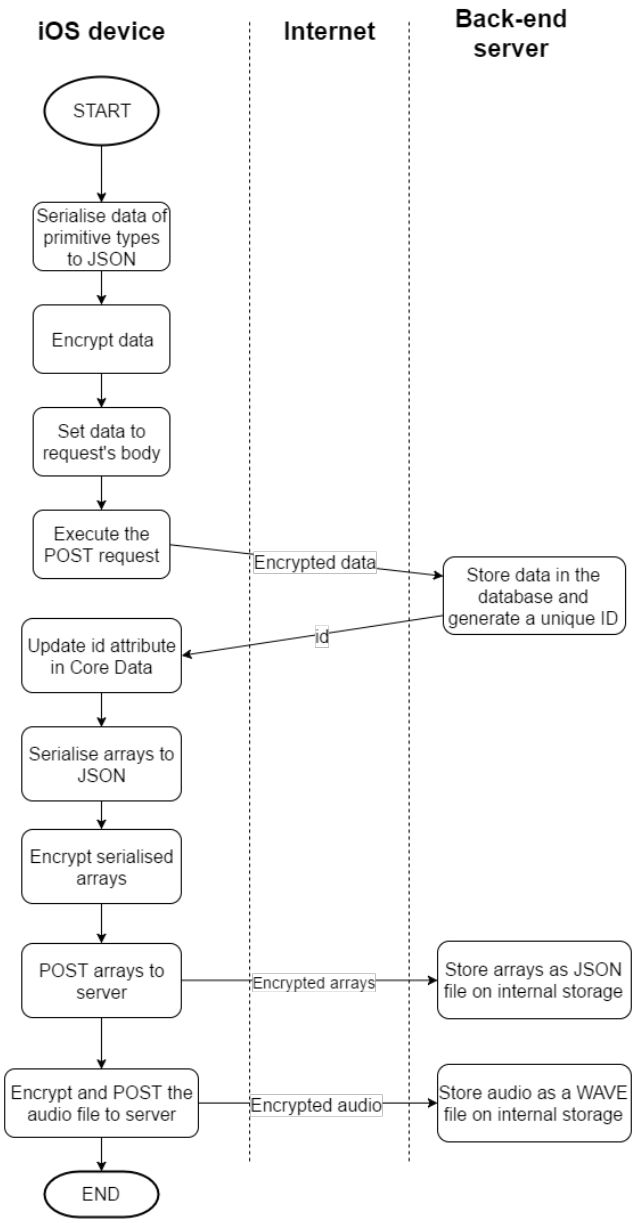


Figure 5.4: Process for uploading a record from WCD to the server.

despite being encoded in JSON, because these arrays could reach very large sizes, depending on the recordings length, thus the servers memory resources would be exhausted if it attempted to decode such arrays. Therefore, arrays are transmitted as encrypted JSON data, decrypted on the server, and then stored as JSON files on the server, without parsing. The primitive data is also transmitted as encrypted JSON, but its size does not increase with the recording length, thus it is encrypted and parsed on the server, and then stored in a database.

5.2 Whoop Research

5.2.1 Architecture

The WR application is designed on the basis of WCD application with several key differences. The WR application does not perform whooping cough identification, this it doesnt need to store and display users past records. It also does not schedule notifications, as the user is prompted to enter their diagnosis within the linear workflow. The simplified design is illustrated in Figure 5.5.

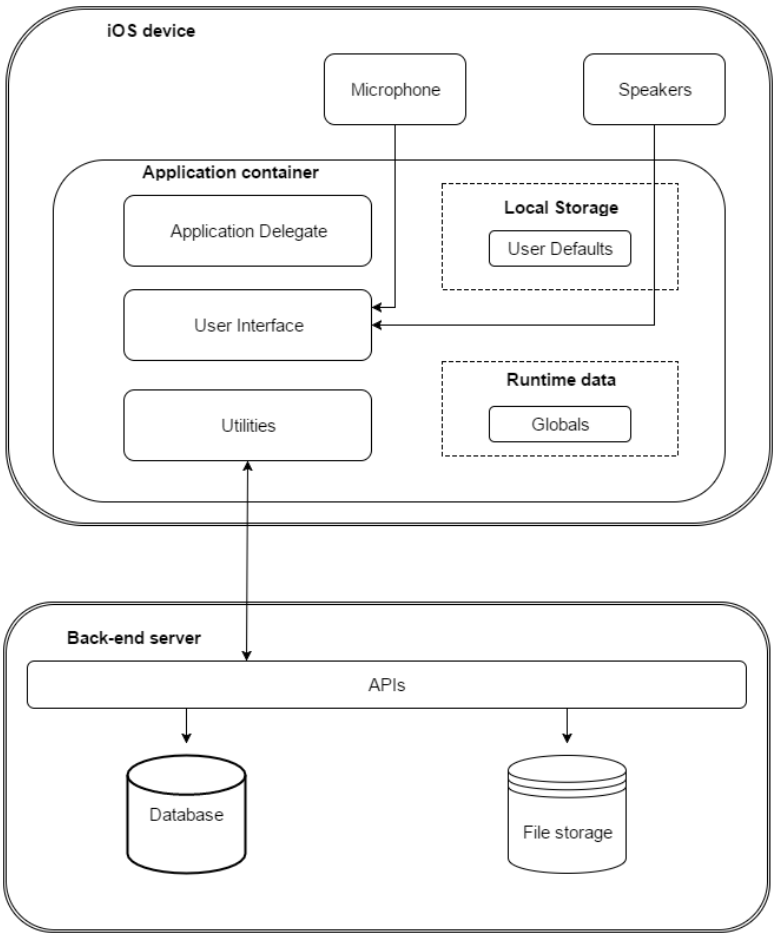


Figure 5.5: WR architecture.

Limited local data persistence requirements allowed me to reduce the local storage technologies just to User Defaults, as well as to eliminate the Result class model.

In the WR application, data is uploaded onto the server after each step, so recording is uploaded first, symptoms second, and diagnosis third. Such design also means that the screens have to

be presented in the order of importance, in order to maximise the usefulness of collected data, should the user quit WR application prematurely.

The Globals module contains a global reference to the recordings location, a flag indicating whether the user has launched the WR application previously, and encryption keys.

WR will also need to be localised, thus localisation-friendly implementation will need to be considered.

5.2.2 Data persistence

Three pieces of data are persisted within this application:

Name	Type
isNotFirst	Boolean
Symptoms	String
Diagnosis	String

Table 5.2: Variables persisted in the WR application.

It can be seen from Table 5.2, that all of the persisted data consists of single primitive values, which makes NSUserDefaults an ideal technology for persistence.

Audio recordings are still created within the applications Documents directory, however they are deleted post upload, thus references to their locations do not need to be persisted between the applications launches.

5.2.3 Networking

Similar to the WCD application, WR communicates with the back-end server via HTTP POST requests, sent to its own set of APIs. In this case, audio recording, symptom and diagnosis are all uploaded separately. Once the audio recording is uploaded, the server responds with an identifier, which is unique to the whole system, and this identifier is then used to store symptoms and diagnosis in the correct row in servers database.

5.3 Administration Panel

5.3.1 Architecture

The AP is designed as a web application, which is composed of the three modules: Authentication, WCD Panel, and WR Panel. The Authentication module follows a thin client design by providing HTML forms for entry of credentials on the front-end, but performing authentication and account creation entirely on the server. The WCD and WR panels follow the smart client, dumb server paradigm, whereby all logic and rendering is performed on the front-end, and the back-end is only used for data retrieval and deletion.

5.3.2 User management

A database on the back-end server contains a table which stores usernames, hashed passwords and salts. During the account creation, the user enters details, which are then submitted to the server. The server verifies that the code word is correct, and that the username is not taken. If that is the case, it encrypts the password and creates a new row in the database with the users details. Once the new row is created, a user session is initialised, and the user is redirected to the WCD panel. Both WCD and WR panels check that the session is active before loading any contents. If they determine that the session does not exist, the user would be redirected back to the login page. If code word is incorrect, or the username is taken, the server returns an error, informing the users about a mistake in the form.

During the login, the user enters their details, which are submitted to the server. The server attempts to find an entry in the database corresponding to the provided username, and compares it to the password stored in the database. If both credentials match, the user session is initiated, and the user is redirected to the WCD panel. If credentials do not match those stored in the database, the user receives an error.

When the user clicks on the Log out button, the user session is terminated, and the user is redirected back to the login page.

5.4 Back-end server

5.4.1 Database

Back-end server is responsible for data storage, and user management. As such, it contains a MySQL database with three tables: users, results, research. The MySQL database has been chosen, as it is fairly simple to use for data storage and retrieval.

In order to enable the three applications to interact with the database, an API layer has been created, which allows client applications to create, read, update and delete records from the database, representing a CRUD design pattern.

However, MySQL database is not suitable for storing very large arrays, such as those containing waveforms and identification events, thus these arrays were stored in a JSON format on the servers internal storage. Additionally, the audio recording files were also stored outside the database.

In order to maintain link between the database records and the corresponding JSON and audio files, these files were named according the records unique identifier. For example, in case of data collected from the WCD app, there would be a record in the results table where id is 29, and the audio file named 29.wav, and a JSON file, named 29.json.

Additionally, the back-end server provides API for file download, which is used to retrieve the corresponding audio file from the internal storage.

5.5 Security

5.5.1 Network security

Any data sent from the device to the server is encrypted via AES-128 cipher, which ensures that it cannot be eavesdropped by a malicious party. Once the encrypted data is received on

the back-end server, it is decrypted using the private key.

In the initial stages of development, asymmetric RSA encryption was considered for achieving secure data transmission, as this scheme uses a public key for encryption, and private key for decryption, therefore we dont need to worry about the encryption key falling into the wrong hands. However, it is only possible to encrypt data up to 245 bytes, which is insufficient for our purposes. Advanced Encryption Standard (AES) is a symmetric key symmetric block cipher, which can be used to encrypt large amounts of data, and commonly comes in three flavours: 128-bit, 192-bit, and 256-bit. For example, AES-128 encrypts the data in blocks of 128 bits using a 128-bit key, by applying 10 rounds of encryption. AES-192 uses 12 rounds of encryption, and AES-256 uses 14 rounds. All three sizes are deemed sufficient to ensure secure data transmission, with any exhaustive search attack exceeding the current possibilities. However, AES-128 cipher is about 40% faster than AES-256, and requires more memory resources for decryption, which were not accessible during this project. Thus, AES-128 cipher will be used for encryption throughout this project.

In order to ensure secure communication, both the client and server need to know a private key, thus currently the same key is stored both on the device and the server. This encryption scheme not only protects the data from eavesdropping during the transmission, but also protects the back-end server from impersonation attacks. The server exposes several public APIs which are designed to allow the three applications to interact with the database. If encryption wasnt used, malicious parties could potentially view and modify the database. Usage of this encryption scheme ensures that if the data arrives, which is not encrypted using the correct key, the request would be aborted.

The second line of defence used for preventing the impersonation attacks, is the usage of an API key, which is included in every request made to the server. Once the API receives a request, it decrypts the data, and verifies the API key. If the API key does not match the one stored on the server, the request will be aborted.

Another server-side security consideration is the prevention of SQL injection attacks. SQL injection is the type of attack, where a malicious user suspects that a string will be stored in

a database, thus they input a malicious string through the client application. For example, during the account creation, it is common for back-end servers to check if the username already exists, which is accomplished using am SQL query like "SELECT * FROM users WHERE username = " + $_POST[username]. In this case, $_POST[username] would be the string from the username field on the signup page. An attacker could input myUsername or 1=1 into username field, and they would receive all details stored within the users table, as 1 is always equal to 1.

In order to prevent such attacks, any user-supplied strings are escaped, which inserts backslashes in front of potentially dangerous characters. This prevention method ensures that the user-supplied string is always treated as a single string, as opposed to a string and an SQL statement.

5.5.2 Authentication security

Many internet users tend to reuse their passwords between different services, for example they could use the same password for Facebook and online banking. This may lead to a situation where their Facebook password is compromised, and is then used to log into their online banking system.

Thus, it is essential to protect the APs users, should the service be compromised. This is sometimes achieved by hashing the user-supplied passwords. Hashing uses Secure Hash Algorithm (SHA) to encrypt passwords, which convert plain text passwords to cipher text. Cipher text cannot be converted back into plain text algorithmically, it is vulnerable to dictionary attacks. In a dictionary attack, malicious parties create a dictionary of hashes for every possible combination of characters, which allows them to find the hashed password in the dictionary, and determine the corresponding plain text password.

In order to prevent this vulnerability, a technique called salting is used. According to this technique, a random string (salt) is generated, appended to the user-supplied password and hashed. The resultant hash is then stored in the database, along with the salt which was used to generate the hash. This severely restricts any possible dictionary attack, since the number

of possible permutations of salts and passwords is too large.

When the user attempts to log in, their salt is retrieved from the database, appended to the plain text password, hashed, and then compared to the stored hash. If the hashes match, the user would be allowed to log in.

5.6 Privacy

In both WCD and WR application, no personal identifiable information is collected, thus the users privacy would not be at risk, in the event that the server is compromised.

Chapter 6

Implementation

The previous chapter provided a high-level overview of the system architecture. This chapter will be discussing the most interesting low-level implementation details in this project.

6.1 Languages

6.1.1 iOS

Both WCD and WR are written in Swift 2.2, which is the new open-source language developed by Apple. It is designed to be safer, faster and more expressive than its predecessor. Previously, iOS applications were coded in Objective-C, however since Swifts release in December 2015, many iOS libraries and open-source projects started transitioning to Swift. However, Swift remains compatible with Objective-C, thus older libraries can still be imported into the project using a bridging header. Additionally, every effort has been made to ensure that the code will remain compatible with Swift 3.0, which will be released later in 2016. For instance, the code avoids using c-style for loops, and increment operators.

6.1.2 Back-end

PHP is used as a server-side language. PHP is the most popular language on the web, thus it is widely supported on the shared hosting plans and well documented. Moreover, I have some existing knowledge in this language, thus using PHP ensured that this project fits the given timeframe.

6.1.3 Front-end

The Administration panels user interface is created using HTML, CSS and JavaScript. These three languages represent the most common front-end stack, with HTML being used for creating elements on the web pages, CSS being used for styling these elements, and JS used for client-side business logic, such as handling click events, rendering visualisations, etcetera.

6.2 iOS applications

6.2.1 User Interface

View controllers (VC) are used to create UI elements for each screen programmatically. This ensured that the element bounds and font sizes could be calculated programmatically in each VC, according to the screen height. The WCD and WR applications are designed to support iPhone 4s of newer, which means that we need to support only four different screen sizes listed in Table 6.1

Device model	Screen size in points
iPhone 4 / 4s	320 x 480
iPhone 5 / 5s / 5c / 5se	320 x 568
iPhone 6 / 6s	375 x 667
iPhone 6 Plus / 6s Plus	414 x 736

Table 6.1: Screen sizes of various iPhone models.

Each VC is associated with a View in the iOS Storyboard, thus when the VC is initialised, it loads the associated View, following which several methods are triggered: viewDidLoad, viewWillAppear and viewDidAppear.

In order to create elements within that View programmatically, they need to be initialised within the viewDidLoad() method, and added to the View as Subviews. In our applications, we also wish to have different sizes for each element, thus it is necessary to calculate the screen height in points, and use a switch statement to set different element and font sizes for each height, corresponding to different devices.

iOS devices use points instead of pixels, with each point representing a coordinate on the screen. So (0, 0) would correspond to the top left corner, and (320, 480) would correspond to the bottom right corner on iPhone 4s. Points are preferable to pixels, as some devices use Retina screens which use high pixel density to display sharper UI elements.

Programmatic creation of the UI elements was chosen instead of visual creation, as this allowed me to create reusable elements by changing only some of their properties, and tailor their sizes and positions according to each screen height. Additionally, this allowed me to declare colours within the code, which made it easier to change colours during the development, to find the most visually-appealing palette. Both WCD and WR applications are expected to be used in a variety of developing nations, thus is important to consider localisation throughout the development. Programmatic creation of UI elements meant that all the text was declared in code, which could be easily localised.

Visual creation of UI elements is performed by positioning these element in the Storyboards, and setting their properties. I used this methodology to implement the My symptoms and Diagnosis screens, as their layout was quite simple, which allowed me to take advantage of Auto Layout. Auto Layout is a layout system which positions and resizes UI elements, based on the constraints defined by the developer. Constraints are essentially relationships between elements and screen bounds. An example of a constraint would be to specify that a text label should be positioned 30 points below the recording button.

Side menu is facilitated using the IOS-Slide-Menu library, and is initialised within the Application Delegate by creating a VC for the menu, and adding it to the WCD applications navigation bar. Throughout the application, the librarys APIs are used to show and hide the side menu, as well as to detect when the menu is closed.

It can also be noted that the Help and Result screens feature a blur effect, which are achieved by creating a UIVisualEffectView using UIBlureffect of the same size as the screen, and adding it to the View contained within the VC. zPosition is then used to adjust the order of displaying elements. For example, on the Help screen the red circle is positioned at zPoisiton of -2, the blur effect is at zPositon of 0, and the text is at zPosition of 1.

In case of the WCD application, a navigation bar is used, which is implemented using the navigation controller. The navigation controller contains a hierarchy of views, also referred to as a navigation stack, that determines which view should be presented currently, and which views were presented previously. This navigation stack is often used for applications that implement a Back button for users to navigate to the previous screen without that screen losing its state. However, in our applications, the Back button is intentionally not implemented, thus we experiencing no benefit from keeping the previous views in the hierarchy, as they occupy memory, and require us to manage their state post transition. Thus, in order to facilitate transitions between VCs, a helper goTo() function was implemented in the Utilities module, which takes the name of the desired controller as an argument, pops all VCs from the current navigation stack, and pushes the desired VC onto the stack. In effect, this deletes all instances of the previous VCs, and transitions the user to the next VC.

In case of the WR application, a navigation bar is not used, thus the helper goTo method works by replacing the first within the application window (current VC) by the desired VC.

6.2.2 Recording

Recording is performed using Apples AV Foundation framework, which provides services to work with audio-visual media. AV Foundation provides an AVAudioRecorder class, which can

be used for recording audio.

In the Recording VC, an instance of AVAudioRecorder class is initialised with an empty WAVE file, and dictionary of settings. This dictionary is configured to ensure that the recording is performed with settings suitable for the WAVE format, thus the recording uses linear PCM format with 2 channels, with the bit depth of 16, and sample rate of 16 kHz. This sample rate was chosen, because the important information in the recording was did not exceed 8 kHz, making the Nyquist frequency equal to 16 kHz.

When the user touches the recording button, a record method is called on that instance, and a timer is scheduled to be triggered every 0.05 seconds. This ensures that the progress circle around the button is updated periodically. When the 30 second time limit is reached, or the user touches the recording button again, the timer is invalidated and stop method is called on the AVAudioRecorder instance.

During the development, it was discovered that the audio visualisations on the Playback screen do not work with very short recordings. Thus, when the user attempts to stop the recording, the AVAudioRecorders currentTime property is read, and if it was found to be less than 1 second, then the stop method would be called after a 0.5 second delay. This was achieved by implementing a delay method, which accepts the duration in seconds, and a closure as arguments. Swift closures are blocks of functionality, which can be declared as variables, and passed to other functions as arguments. Thus, I have defined a stop closure, which is used to stop the recording, save the file reference, and transition to the Playback screen.

6.2.3 Playback and visualisation

Visualisation is created using EZAudio librarys EZAudioPlot class. This class was created specifically to deal with audio visualisation, thus it contains necessary logic to visualise and interpolate the audio waveforms, while utilising GPU for compositing frames. This is more suitable for our applications than the general-purpose charting libraries, as they are not designed to deal with a large number of samples, while a typical recording would contain about a million

samples.

During the VC load, the EZAudioPlots buffer is set to contain all samples from the newly recorded audio file, which enables the visualisation. Depending on the recording conditions, the waveforms amplitude may vary significantly, thus it was necessary to apply gain to the visualisation, in order to ensure that the waveform fills the plot without features being clipped. This gain was calculated by Equation 6.1.

$$Gain = \frac{1}{A_{max}} \tag{6.1}$$

Audio playback is performed using EZAudios EZAudioPlayer class. This was chosen over AV Foundations AVAudioPlayer class, as it provides a delegate, which is triggered whenever a sample is played. This functionality becomes useful during real-time visualisation. Delegation design pattern entails creating a delegate protocol in the class, implementing the delegate methods within that class, and calling these methods ad hoc. In our case, EZAudioPlayer class implements the protocol and methods, while the Playback VC adheres to the protocol by implementing the same delegate methods, which allows the VC to be notified whenever a sample is played.

During audio playback, real-time visualisation is displayed by receiving each sample from the delegate method, and appending it onto the plots buffer. In order to ensure that the plot fills horizontally, the EZAudioPlots rolling history length needs to be set according to Equation 6.2.

$$Rolling\ history\ length = \frac{44100}{1024 * audio\_duration} \tag{6.2}$$

In this case, 44100 Hz is the playbacks sampling frequency, 1024 is the number of frames per delegate callback, and audio duration is the duration of the audio in seconds, extracted from the recorded WAVE file.

Use buttons behaviour differs between the WCD and WR applications. In the WCD application, the user is presented with a symptoms questionnaire screen, and then return to the Playback

screen, which is when the identification begins. In order achieve this, the My symptoms screen has to be presented modally, and then dismissed once the user completes the questionnaire. Therefore, the playback VC has to be notified when the questionnaire has been dismissed, in order to begin identification automatically. This is achieved by utilising the delegation design pattern described earlier. In case of the WR application, the user is simply transitioned to the questionnaire screen, using the goTo() helper method.

Prior to launching the identification process, a spinner has to be created, in order to notify the user about the identification progress. The spinner is created and displayed by using the SwiftSpinner class. Identification progress is updated by scheduling a timer which fires every 1 second, and updates the progress as shown in Equation 6.3.

$$Progress = min(\frac{100 * time\_elapsed}{expected\_duration}, 100) \tag{6.3}$$

Here, time_elapsed is the time since the identification process has started in seconds, and expected duration of the identification process given in seconds for each device. The expected identification duration is calculated by Equation 6.4.

$$Expected\_duration = audio\_duration * m \tag{6.4}$$

Here, m is a multiplier that estimates the expected duration from the audio duration. For example, on iPhone 6 Plus, the identification process usually takes 3.5 times as much time as the audio duration.

6.2.4 Whooping cough identification

Implementation of the whooping cough identification module relied on the results of Pramonos Matlab implementation. Throughout the development, outputs of each stage of the algorithm were compared against the Matlabs output to ensure consistency, which represents Test-Driven

Development (TDD).

TDD is a programming technique, which can be described by a series of steps. First step is to separate the application into small functional components that can be run independently of each other. Second step is to write a unit test for each component, which specifies an input, and compares the functions output to the expected output. These tests should fail at first, as the functions are not yet implemented. Third step is to implement each function until all the tests pass. This process is summarised in Figure 6.1.

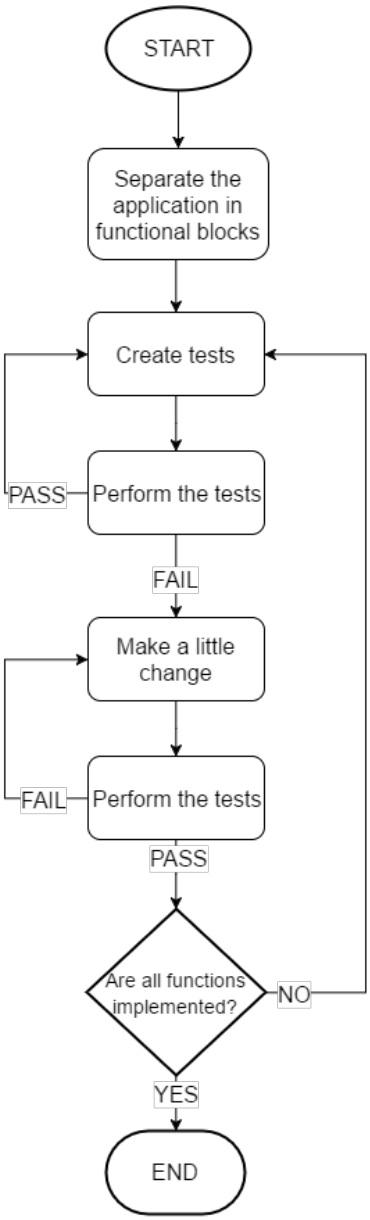


Figure 6.1: TDD process.

In iOS development, tests are usually constructed, using the XCode Unit Testing framework.

However, in the interest of time I conducted these tests by separating the Matlab implementation into the same functional blocks and providing the same input both to Matlab and WCD, and comparing their results manually.

On high-level, whooping cough identification algorithm was outlined in Figure 5.3. This section will focus on the low-level implementation details.

Pre-processing

The first step is pre-processing, which is performed by standardizing and normalising the data.

Recordings need to be standardised, as their means may vary, depending on the recording conditions. Thus standardising modifies the recording to have zero mean, which will come useful when we reach the classification steps, as the same LRM coefficients need to work for all recordings. This is performed by Equation 6.5.

$$z = \frac{x - \mu}{\sigma} \tag{6.5}$$

Additionally, the recordings need to be normalised in order to ensure that all recordings fit within the same range. This will also necessary to achieve the correct results during classification. Normalisation process is described in Equation 6.6.

$$norm = \frac{x}{X_{max}} \tag{6.6}$$

MFCC calculation

Mel-Frequency Cepstral Coefficients (MFCCs) are widely used in the automatic speech recognition applications, as they are able to accurately represent the energy envelope of the signal. Such accurate representation is achieved by projecting the signals spectrum onto Mel-scale, which is designed to mimic operation of cochlea by utilising the Mel filterbank, where the

frequency bins grow larger with higher frequencies. This is inspired by the observation that humans can recognise changes in lower frequencies better than those and higher frequencies.

WCDs classification steps will require 39 MFCC features, consisting of 12 MFCCs, 1 zeroth-order coefficient, as well as first and second derivatives of the MFCCs and coefficients.

Pramonos implementation relied on a Matlab toolbox called Voicebox, which was created by Brookes. Voicebox contains a function melcespt, which can be used to calculate the desired MFCC features. However, an equivalent toolbox does not exist for iOS, thus I implemented the melcespt function in Swift, inspired by Brookes implementation. This functions operation is detailed in Figure 6.2.

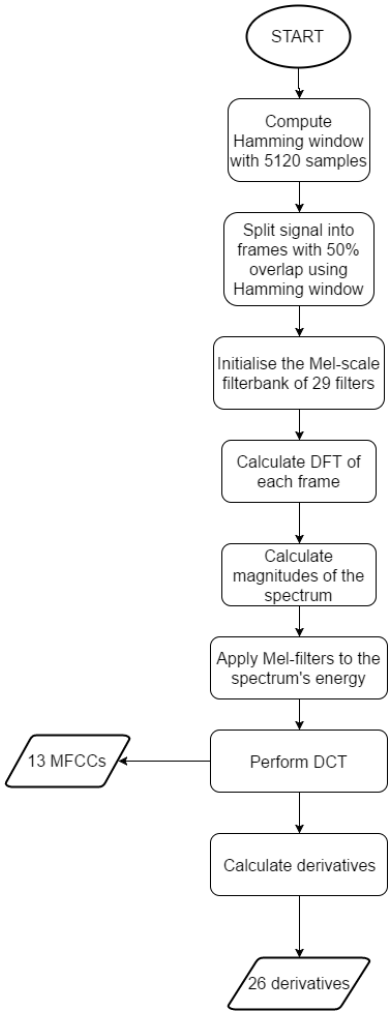


Figure 6.2: TDD process.

First, the recording is split into 320ms frames with 50% overlap using Hamming window. Framing is needed to obtain reliable spectral estimates. Overlapping is used to smooth values

on the frame bounds, in order to prevent sudden jumps in the spectral estimate.

Next, the Mel-scale filterbank is computed. Each Mel-filter is a triangular function, thus the filterbank represents a comb of filters spaced according to the Mel scale from 0 to 8000 Hz. The filterbanks implementation was inspired by Brookes melbankm implementation.

Discrete Fourier Transform is usually computed using a Fast Fourier Transform implementation. Apple provides Accelerate framework, which includes a native FFT implementation, which is accelerated using GPU. However, this native implementation only works with powers of 2, while our frame size is 5120. Therefore, the DFT was computed using Chirp-Z Transform (CZT).

CZT is a generalisation of DFT, which was first introduced by Rabiner et al. in [13] as an extension of Bluesteins algorithm. Consider the forward DFT formula in Equation 6.7:

$$X_k = \sum_{n=0}^{N-1} x_n (e^{-2\pi i/N})^{kn}, \qquad k = 0, ..., N-1 \tag{6.7}$$

Now, consider an algebraic identity in Equation 6.8:

$$kn = \frac{1}{2}(k^2 + n^2 - (k-n)^2) \tag{6.8}$$

Substituting this identity into the DFT formula, results in Equation 6.9:

$$X_k = ((e^{-2\pi i/N})^{k^2/2}) \sum_{n=0}^{N-1} (x_n (e^{-2\pi i/N})^{n^2/2} (e^{-2\pi i/N})^{-\frac{(k-n)^2}{2}}) \tag{6.9}$$

This assumes the form of convolution with an added $(e^{-2\pi i/N})^{k^2/2}$ factor, called chirp. My CZT implementation is illustrated in Figure 6.3.

Since the signal is padded to the nearest power of 2, FFTs and IFFT are implemented using Apples native implementation. This results in a complex spectrum, however Swift does not natively support complex numbers, hence Kogais swift-complex library was used to work with complex numbers.

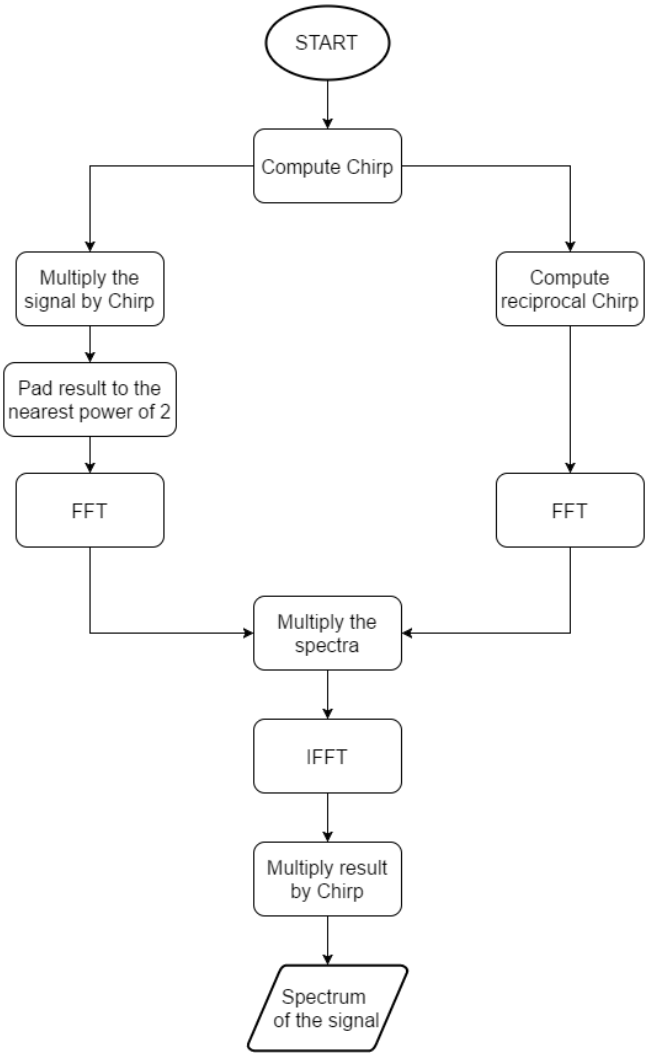


Figure 6.3: Chirp-Z transform.

Next, magnitudes of the spectrum were computed using the abs method from swift-complex. It works by multiplying each complex number by its conjugate.

In order to apply the Mel-filters, the melcespt function performed pairwise multiplication between the filter values and spectrum magnitudes, followed by computing a logarithm of the result, in order to ensure that the Mel-filters are applied to the spectrums energy.

Discrete Cosine Transform (DCT) of the result was then computed, in order to decorrelate the filterbank energies, which will be needed for the classification steps. This implementation was based on Brookes rdct function. This resulted in 12 MFCCs and 1 zeroth-order coefficient for each frame.

First order derivatives for each frame were then computed by applying a 1-Dimensional Finite

Impulse Response (FIR) filter to the previous result. Second order derivatives were computed by applying the 1-D FIR filter to the first order derivatives.

Sound event detection

Prior to performing feature extraction, sound event detection was performed to ensure that features were extracted only for useful segments. This acts to improve the speed of feature extraction, and performance of the classification stages by eliminating the potential false positives.

In order to perform sound event detection, the recording was split into 20ms frames, standard deviations for each frame were calculated, and mean standard deviation of all frames was computed. Next, the frames with higher standard deviation than the mean were classified as sound events, while those frames with lower or equal standard deviation than the mean were classified as silent frames.

Such sound event detection method is quite accurate, however it sometimes results in short silent segments within the voiced segments. In order to eliminate these false negatives, close consecutive events were grouped together. Events were considered to be close, if they were spaced three frames apart, or closer.

Feature extraction

Following the sound event detection, 12 frequency domain features and 2 time domain features were extracted.

In order to compute the frequency domain features, Power Spectral Density (PSD) had to first be estimated. Following Pramonos implementation, a one-sided PSD estimate was calculated using Welchs modified periodogram method. Welchs method is advantageous, as it reduces noise in the estimated PSD at the expense of a lower frequency resolution. This method works by splitting the signal into overlapping frames, applying a window function to these frames, computing DFT and averaging the resultant periodograms.

In my implementation of Welchs method, a 256-sample Blackman window is used, and 50% overlap is applied. This is illustrated in Figure 6.4.

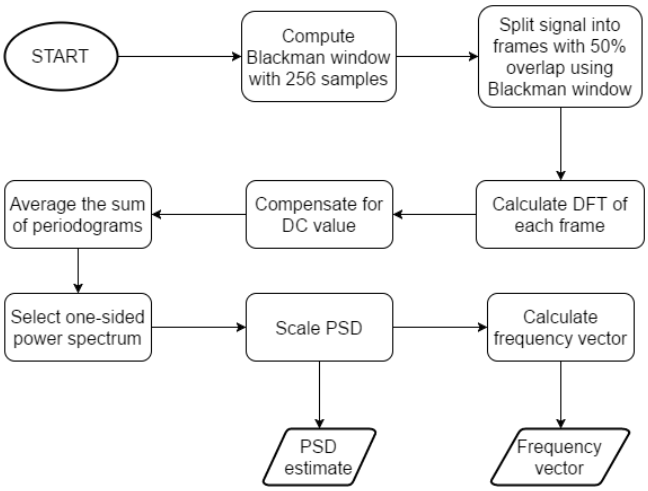


Figure 6.4: Estimating PSD using Welch's method.

Periodic symmetric Blackman window is calculated by equation 6.10. Periodicity is achieved by extending the window by one sample, computing the symmetrical window, and removing the last sample.

$$w(n) = 0.42 - 0.5\cos(\frac{2\pi n}{N-1}) + 0.8\cos(\frac{4\pi n}{N-1}), \quad 0 \leq n \leq M-1 \tag{6.10}$$

Here, $M = (N+1)/2$, and $N = 256$. Second half of the window is obtained by flipping the first half around $N/2$.

In this case, DFT was performed using Apples standard implementation, as the periodogram length was always fixed at 256 samples.

During the PSD estimation, the Blackman window was convolved with every power spectrum peak, thus it was necessary to compensate the estimate for the DC value. This was achieved by dividing the PSD estimate by the sum on squared window.

One-sided PSD was selected by extracting the first 129 values from the PSD estimate. It was then scaled by sampling frequency, and all but first and last values were doubled, as these two values are unique Nyquist points.

Frequency vector was then generated using Equation 6.11.

$$w_i = \frac{f_s * i}{256}, \quad 0 \leq i < 129 \tag{6.11}$$

Some frequency domain features need to be calculated by utilising Pxxlog, which is 10 times the logarithm of the PSD.

Dominant frequency is the first frequency domain frequency to be extracted. It was calculated by finding an index of the maximum element in Pxxlog array, and selecting the corresponding element from the frequency vector.

Spectral skewness, also known as the Spectral Asymmetric Coefficient was extracted by implementing Equation 6.12. This is the measure of data asymmetricity about its mean.

$$s_1 = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x})^3}{(\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x})^2})^3} \tag{6.12}$$

Spectral kurtosis is the measure of peaks in power spectrum. Kurtosis of more than 3 means that the spectrum has more peaks than the normal distribution, while kurtosis of less than 3 meant that is has less peaks. Kurtosis is implemented using Equation 6.13.

$$k_1 = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x})^4}{(\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x})^2})^2} \tag{6.13}$$

Next, spectral standard deviation was computed.

Band power was computed by using the rectangle method to integrate the PSD estimate, as illustrated in Figure 6.5.

Width of the rectangle is determined by calculating the differences between successive elements in the frequency vector. Missing width is calculated in Code block 1:

```
let missingWidth = (w.last! - w.first!) / (Double(w.count) - 1.0)
```

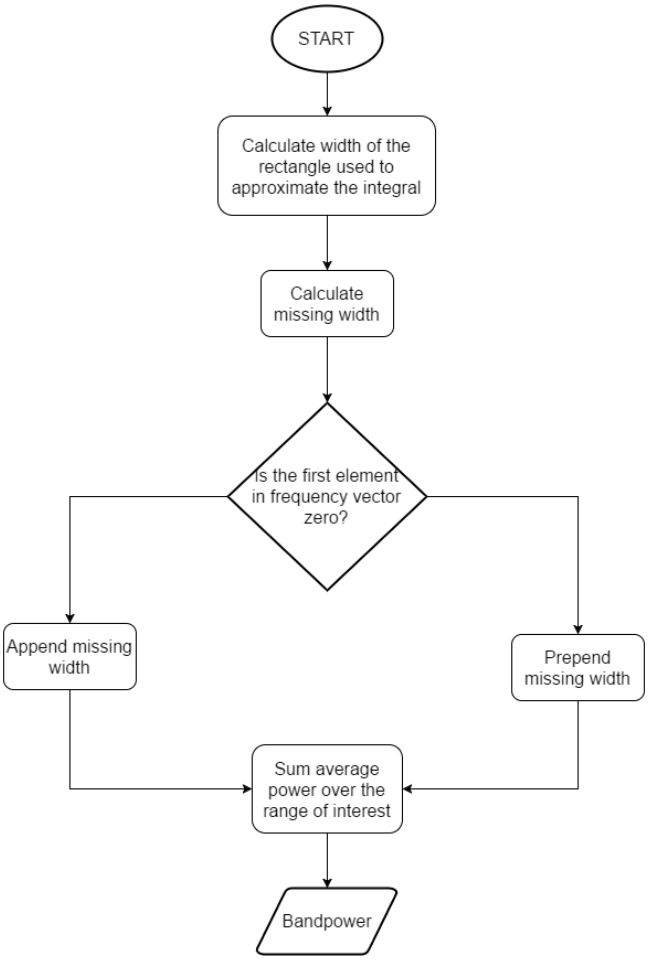


Figure 6.5: Calculating bandpower.

Missing width occurs, as the frequency samples do not cover the whole frequency range due to periodicity, therefore the missing frequency range has to be compensated in the integral.

Finally, the average power is summed as shown in Code 2:

```
// Sum the average power over the range of interest.
var pwr = 0.0
//let count = 0
for i in idx1...idx2 {
    pwr += width[i] * x[i]
}
```

Spectral Roll-Off consists of 5 elements: first quartile frequency, central frequency, third quartile frequency, interquartile frequency and 90th percentile cumulative frequency. The first 3

elements are calculated, as shown in Figure 6.6.

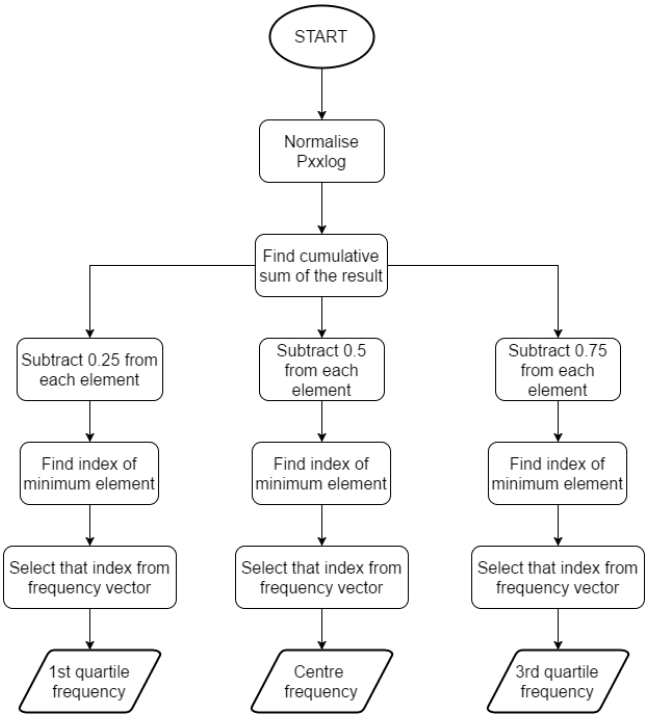


Figure 6.6: Calculating quartile frequencies.

Pxxlog is normalised by dividing each element by the sum of all elements. Interquartile range is found by subtracting the quartile frequency from the 3rd quartile frequency. In order to calculate the 90th percentile cumulative frequency, several steps have been applied. First, the total energy in the sound event has been found by normalising the signal, finding DFT of the first 256 elements, and summing the magnitudes of the first 128 spectral samples. Then, an index of the first sample exceeding $total_energy * 0.9$ was determined. Finally, the frequency has been found using Equation 6.14.

$$f = \frac{index * f_s}{256} \tag{6.14}$$

Spectral centroid represents the center of mass of a spectrum. It is computed as a weighted mean of the spectrum, by using frequency magnitudes as weights.Spectral spread is a measure of the bandwidth of the spectrum. Spectral decrease averages the set of slopes between frequencies f_k and f_1. Spectral flatness is a measure of noisiness of the spectrum, which is defined as a ratio between geometric and arithmetic means. The last frequency-domain feature to be extracted

is spectral slope is a measure of the spectrums decreasing slope. It is computed using linear regression over the spectral amplitude values. [11]

Next, the function extracted two temporal features: zero-crossing rate and crest factor. Zero-crossing rate (ZCR) measures the number of sign changes of the signal. In order to compute ZCR, the function was implemented that multiplies each sample by its successor, and then the negative products are counted, which represents the number of sign changes. Finally, ZCR is determined by dividing this count by one less than number of samples in the signal.

Crest factor is the ratio between peak amplitude of the waveform, and its Root Mean Squared (RMS) value.

Cough sound detection

Once the audio features have been extracted, they can be used in the subsequent classification steps, namely cough sound classification, whooping cough classification, and whooping sound classification. Pramonos implementation used binomial Logistic Regression Model (LRM) in order to perform classification. LRM is a form of regression analysis that is able to compute probabilities of a random variable belonging to a certain category. The LRM function consists of dependent variables, independent variables, and coefficients. Dependent variables can be thought of as prediction outcomes ranging from 0 to 1, independent variables are the factors influencing these outcomes, and coefficients describe how independent variables affect the outcomes.

Mathematically, LRM is defined by the sigmoid function in Equation 6.15:

$$g(z) = \frac{1}{1 = e^{-z}} \tag{6.15}$$

Here, z is a linear predictor, which is computed by Equation 6.16:

$$t = \beta_0 + \beta_1 x_1 + ... + \beta_m x_m \tag{6.16}$$

Here, Beta variables represent model coefficients computed during the model training. Binomial LRMs contain two dependent variables, also referred to as categories. In our case, these are cough and not cough, whooping cough and not whooping, whooping sound and not whooping sound. The audio features extracted in the previous section are considered to be independent variables for the purpose of our model. Pramono in [12] performed feature selection for each classification step, and computed the LRM Beta coefficients. The WCD implementation will be using these same features and coefficients, in order to achieve the same outcome as Pramonos implementation, for the purpose of adhering to the TDD development framework.

The coefficients for each classification stage are stored within the Globals module, to avoid polluting the implementation with lengthy variable declarations. For the purpose of cough detection, LRM classification has been implemented as function, which accepts a vector of complex-valued coefficients and a matrix of real-valued features as an input, and outputs a matrix of real-valued probabilities ranging from 0 to 1. Each column in the features matrix represents an observation, which is a set of the 29 features (predictors) for each sound event, as determined by the sound event detector. The probabilities matrix consists of two rows, with the first row representing probability of the event being non-cough, and second row representing probability of the event being cough. For example, if we supply a feature matrix with 4 columns, we will receive a 2x4 matrix, representing the probabilities. Implementation of the classification function is detailed in Figure 6.7.

Here, linear predictors are computed according to Equation 6.16. In practice, this was achieved by performing matrix dot multiplication between the predictor matrix and the coefficients vector. Natural predictions for the non-cough category were computed according to Equation 6.15.

Once the predictions have been calculated, threshold was applied to determine whether the event contains cough. It was assumed that the event contains cough if the probability of cough category exceeds 0.6.

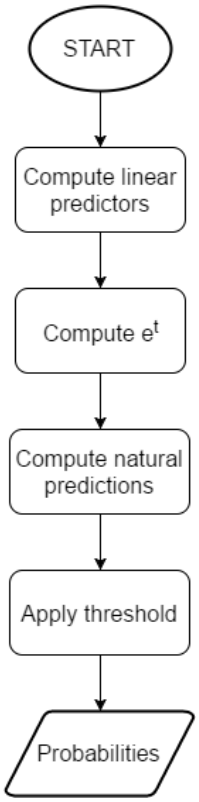


Figure 6.7: LRM implementation.

Whooping cough detection

This step focuses on examining the previously detected cough events, and determining whether that cough is characteristic of pertussis or another disease, such as croup or bronchitis. Figure **??** outlines the implementation of this step.

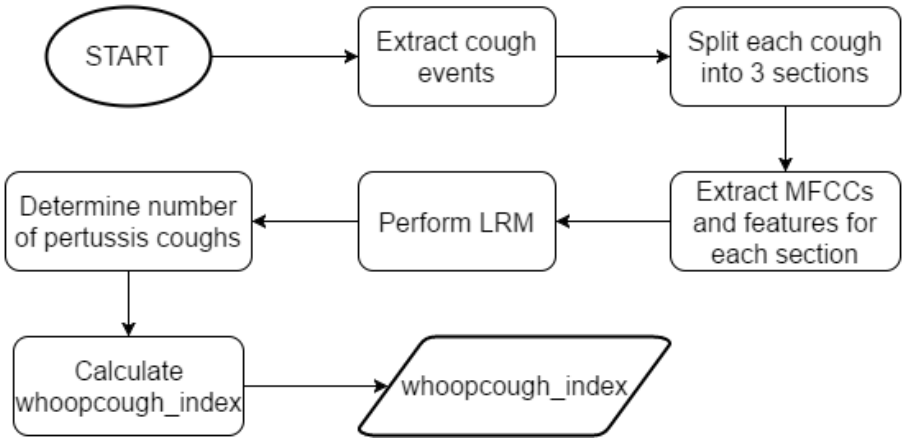


Figure 6.8: Whooping cough detection implementation.

First, the cough events were extracted from the signal, using the identification results from the cough detection step. This involves joining closely located cough events, and removing

events which are shorter than 2000 samples, or longer than 9000 samples, as these are not characteristic of a typical cough.

Next, each cough event was separated into three equal length sections, in an attempt to identify the three stages that are typically present in the pertussis cough.

MFCCs and audio features were then extracted from each section of each cough event, and used in the multinomial LRM function to perform classification. The LRM function implemented in the cough detection stage was used for this purpose. This resulted in a matrix consisting of 4 rows, where the first row represents the probability of pertussis, second row represents the probability of croup, third row represents bronchitis, and the fourth row represents unknown cough type.

Once the LRM classification has been performed, the number of pertussis coughs was counted, and the whooping cough index was calculated by finding the ratio of pertussis coughs to the total number of coughs. Due to the high false positive rate of this classification stage, if the index came to be less than 0.75, it was divided by 1.5 in order to minimise its weighting in the final result calculation.

Whooping sound detection

The last classification step is aimed at detecting whooping sounds, which typically occur in the pertussis cases after a bout of cough, as a patient gasps for air. It is performed as illustrated in Figure 6.9.

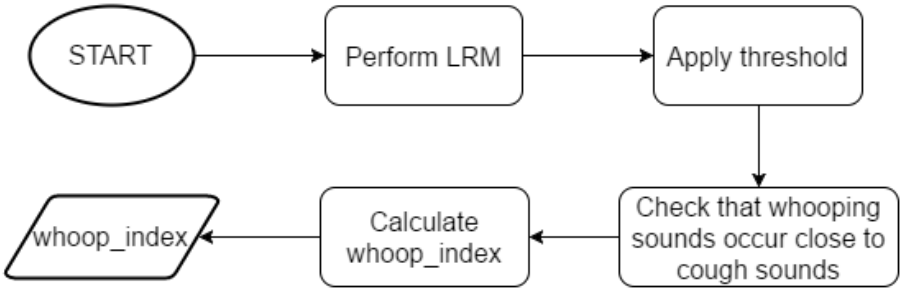


Figure 6.9: Whooping sound detection implementation.

First, binomial LRM classification was performed using the coefficients and features computed

by Pramono. This resulted in a matrix which consists of two rows. The first row represents the probability of the sound event not containing whooping sound, and the second row contains probabilities of the whooping sound being present in that sound event.

Next, a threshold was applied, so that only the sound events that have a probability of whooping sound of higher than 0.8575 are classified as whooping sounds.

Its worth noting that the whooping sounds should occur in close temporal proximity to the cough sounds, due to their clinical nature. Thus, we need to check that the whooping sounds occur no more than 40 sound events apart from the cough sounds.

If any sound events remain classified as whooping sounds after the validation, whoop index is determined to be the highest probability among the valid whooping sounds.

6.2.5 Cough counting

Pramono suggested that the cough rate could be useful to determine whether the user has whooping cough. This is because rapid bouts of cough are characteristic of pertussis. Cough rate was calculated, according to Equation 6.17.

$$cough_rate = number\ of\ cough\ events/(number\ of\ sound\ events * 0.4194) \tag{6.17}$$

The 0.4194 factor was determined empirically to improve the accuracy of whooping cough identification.

6.2.6 identification result

The final step is the whooping cough identification process, is to compute the identification result, ranging from 0 to 1, which makes it convenient to represent the result as a percentage likelihood of whooping cough. If the whooping sound index is higher than 0.5, this result is calculated according to Equation 6.18, otherwise Equation 6.19 is used.

$$result = 0.4 * whoop_index + 0.3 * cough_rate + 0.3 * whoopcough_index \tag{6.18}$$

$$result = 0.5 * cough_rate + 0.5 * whoopcough_index \tag{6.19}$$

This concludes the section of this project, which followed the TDD framework in order to implement Pramonos algorithm.

Storing the result

Once the final identification result has been calculated, it is assigned to the global instance of the Result class, together with the intermediate results. In order to enable visualisation of the identification process on the Administration Panel, Boolean arrays of sound, cough, whooping sound and whooping cough events have been constructed and assigned to the appropriate properties in the Result class. An array of audio samples was also assigned to the corresponding attribute in that instance. These arrays are of the same length as the signal, where presence of the respective events is marked by true samples, and absence of such events is marked by false samples. At this stage, the store method is invoked in order to store the results locally and upload them onto the research server. Finally, a completion handler is invoked, that signals to the Playback controller, that the identification has finished.

6.2.7 Local data storage

Throughout the WCD application, user-generated data is stored locally within the Core Data database. Specifically, this occurs whenever the store method is triggered in the Result class, or diagnosis is updated. The Result class store method makes a call to a private method storeLocal, which is illustrated in Figure 6.10.

First, a unique local identifier is assigned to the record. This is achieved by retrieving the value

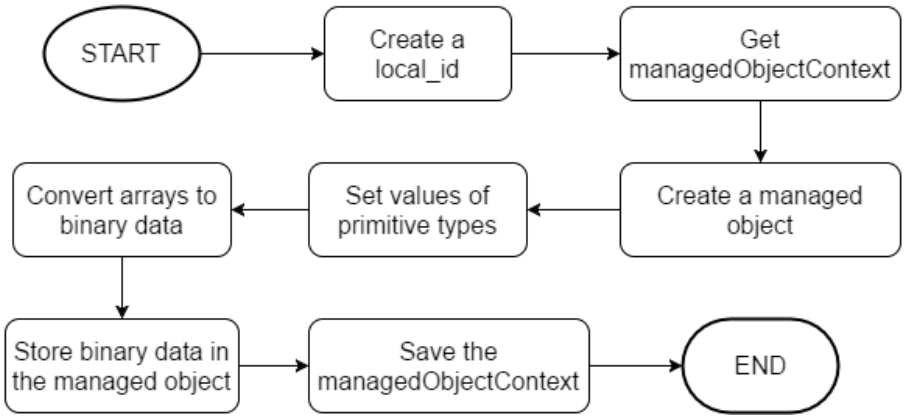


Figure 6.10: Local data storage implementation.

for localID from the User Defaults, incrementing it, saving it back to the User Defaults, and assigning it to the object that needs to be saved. This way, User Defaults always keeps track of the highest local id, which ensures its uniqueness.

Next, the managed object context needs to be retrieved from the Application Delegate. This represents a single object space of the application, which is used to interact with the locally persisted objects.

Managed object is then created and inserted into the context. This will be used to persist the record about to be stored.

This managed object can be thought of as a dictionary with a set of keys representing the attributes of the Result class. Primitive parameters are assigned to the managed object directly, however this cannot be performed for the arrays. Instead, the arrays are converted into binary data, which can be saved directly in the managed object.

Finally, the managed object context is saved by calling its save method.

Data stored within the Core Data needs to be retrieved when the application delegate is checking for the presence of existing records on application launch, or when the user navigates to the My records or Result screens. This is performed using two functions: fetchResultsLocal, and fetchResultLocalByID. The former returns all records stored in Core Data as a collection of Result instances, while the latter uses the local ID to return a specific record as an instance of Result.

fetchResultsLocal works by creating a fetch request for the Result entity, and executing it on the global instance of the managed object context. This returns a vector of managed objects, which are then mapped onto the Result class. Arrays were originally saved as binary data, so in order to retrieve them, the function converts binary data into arrays of Doubles or Booleans by splitting the data into chunks of bytes, according to the types size. For example, elements of type Double occupy 8 bytes, thus the binary data would be split into chunks of 8 bytes, which are then converted into an array of Doubles.

fetchResultLocalByID works by getting all records from the Core Data database using fetchResultsLocal, and iterating through the array to find the record with a matching local ID.

6.2.8 Cloud data storage

Throughout both WCD and WR applications, data needs to be uploaded. This is performed via HTTP POST requests, as discussed in the System Design chapter. The HTTP requests are constructed using NSMutableURLRequest, by initialising the request with the APIs URL, setting the HTTP method to POST, setting Content Type to JSON, and setting the request body to binary data. This is shown in the following code block:

```
// Prepare request
let request = NSMutableURLRequest(URL: NSURL(string: "http://tarakanov.me/Wh
request.HTTPMethod = "POST"
request.setValue("application/json", forHTTPHeaderField: "Content-Type")

// Encode parameters in JSON and encrypt them
request.HTTPBody = dictToEncryptedJSONData(rsaKey, parameters: parameters)
```

In order to execute the requests, and obtain responses, I am using the Alamofire library. This library provides a multitude of tools for HTTP networking, which allowed me to speed up the development process. Code below illustrates an example of executing an HTTP request, and getting a response:

```
// POST the data to server
Alamofire.request(request)
    .responseJSON { response in
        if let JSON = response.result.value {
            if let myID = JSON as? Int {
                // If the object didn't exist in global DB, update the ID
                if myID != tempResult.id {
                    updateIDLocal(tempResult.localID, id: myID)
                }
            }
        }
}
```

6.2.9 Multi-threading

In Swift, code is executed one line at a time, which is known as synchronously. This is a safe way of coding, which works for most applications as most code relies on the results of previous commands. However, the WCD and WR applications contain a multitude of lengthy operations, such as whooping cough identification and data upload, therefore performing these operations synchronously would mean that the whole application freezes until these operations are completed. This would result in laggy applications that are not typically liked by the users.

In order to avoid this problem, it is necessary to understand multi-threading and concurrency. A thread is a series of tasks that are run serially. Multi-threading allows the same process to execute concurrently on multiple threads. Concurrency means that two or more tasks are able to run in overlapping time periods. On single core devices, this results in the two tasks running for short periods of time each. On multi core devices, such as the latest iPhones, concurrency is achieved via parallelism, which results in both tasks running simultaneously on different cores within the processor.

In Swift, concurrency is managed by Grand Central Dispatch (GCD), which is Apples library that provides dispatch queues to manage tasks that are provided by the developer. Dispatch queues execute tasks in a first-in, first-out order, which can be performed either serially or concurrently. GCD provides several global queues, as well as an ability to create our own queue. In this project, we will be using Main, Default and custom queues. Main queue is always responsible for any user interaction and display of elements. Default and custom queues are performed in the background without the user noticing. [1]

Whooping cough identification in the WCD application should start as soon as the user has completed the My symptoms questionnaire, and the user should be redirected to the Result page, as soon as identification has been completed. However, this cannot be performed on the main queue, as we also want to display an animated spinner and progress updates. Therefore, the identification task is scheduled asynchronously onto the default queue, as demonstrated in the following code:

```
dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0),
    let analysisController = AnalysisController()
    analysisController.analyseAudio(global_result.filePath, completion: {
        dispatch_async(dispatch_get_main_queue(), {
            SwiftSpinner.hide()
            self.progressTimer.invalidate()
            // Indicate that the current result is to be shown
            idToShow = -1
            // Update coreData flag
            isCoreDataEmpty = false
            goTo("ResultDetailViewController")
        })
    })
})
```

As can be seen from code above, the analyseAudio method takes two arguments: filePath, and

completion. The former is used to inform the function about the location of the audio file that needs to be processed, while the latter is a closure, also known as a completion handler. This closure gets called by the identification module once the processing is complete. Within that closure, we are hiding the spinner, invalidating the timer, and navigating to the Result screen. These tasks would result in effects that are clearly visible to the user, thus they need to be performed on the Main queue, which is achieved by the first line within the completion closure.

However, this implementation resulted in the Result screen becoming unresponsive for several seconds after it was loaded. As you may recall, the identification module invokes Results store method, which uploads the primitive data first, and uploads arrays with audio file once it has received the response from the server. Even though the first upload is performed in the background, the lengthy arrays and file uploads are performed on the main queue, which causes the application to become unresponsive. This is caused due to the Alamofires implementation of response handling. In order to counteract this, we must explicitly force Alamofire to perform response handling in the background. This was achieved by creating a custom queue, and passing it as an argument to Alamofires response function

6.2.10 Encryption

Whenever data is uploaded onto the research server from the WCD and WR applications, it needs to be encrypted to prevent eavesdropping and impersonation. This is achieved by AES-128 encryption scheme with Cipher Block Chaining (CBC) mode, a random Initialisation Vector (IV), and a static private key, which is stored on the devices and the server. Mode of operation is an algorithm that allows us to encrypt data of more than 16 bytes by splitting the data into 16-byte blocks encrypting each block, and chaining it. If chaining was not applied, an attacker would be able to decrypt the data by finding patterns in the ciphertext. CBC is the most widely used mode of operation. It works by XOR-ing each block of plaintext with the previous block of ciphertext before encrypting it. However, this is still prone to an attack. If each message begins with the same plaintext, then the ciphertext would also be the same, which leaks information to a potential attacker. In order to overcome this, a random IV, which is a

16-byte string, must be generated with every transmission, and prepended to the ciphertext. This will ensure that the ciphertext is different during every transmission, even when using the same private key. [2]

In order to implement the encryption cipher, CryptoSwift library was used. The two applications upload two types of data: dictionaries and audio files. Thus, a helper function for each type was implemented in the Utilities module.

In the case of dictionaries, the helper method used to prepare the data for transmission is outlined in Figure 6.11.

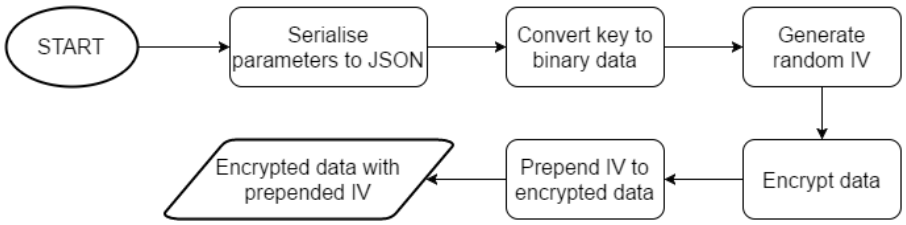


Figure 6.11: Encryption implementation.

First, the dictionary needs to be serialised into a JSON format, as the server APIs are designed to only parse the JSON data. This is performed using Apples NSJSONSerialization class, which outputs JSON in binary data. Next, the private key, which is stored as a string in Globals module is converted into binary data. Random IV is then generated in a binary format using CryptoSwifts randomIV method. The JSON binary data is then encrypted using CryptoSwifts encrypt method, and passing key and IV data as arguments. Finally, plain IV data is prepended to the resultant block of data, in order to enable the server to decrypt the message.

Audio files are encrypted in a similar way to dictionaries, however they are converted into the binary data without using JSON serialisation.

6.2.11 Localisation

It is expected that both WCD and WR applications will be used extensively in non-English speaking countries, as the majority of pertussis cases occur in developing nations. As such, it is important to ensure that both applications are localisable, which is why majority of the

UI elements were created programmatically, while the visually-created UI elements implement referencing outlets that allow their text to be modified programmatically.

iOS has built-in localisation support, which was enabled in the project settings. Localisation relies on the presence of Localizable.strings files in various languages in the project directory. An excerpt from such file is displayed below:

```
"Touch to record" = "                                                    ";
"Recording" = "                          ";
"Touch to stop recording" = "
```

A string on the left hand side is the key, and the string on the right hand side is the localised value. In this project, I named the keys after their English translations for clarity. These files can be distributed among non-technical translators for localisation, and then imported into the project, once translated. Throughout this project, any strings displayed to the user are created using Apples NSLocalizedString class, which requires the key for initialisation. This ensures that the two iOS applications will display localised text to its users, depending on their devices language setting.

6.2.12 Third-party libraries

CocoaPods is a popular dependency manager, which is used within both applications. It allows easily install and update third-party libraries using the command line.

EZAudio is an audio visualisation framework, which is used for creating plots and playing audio on the Playback screens.

PermissionScope is a framework for requesting permissions from the user. It was used to obtain permissions for the microphone and notifications.

Alamofire is an HTTP networking library, which is used to communicate between the applications and the server.

CryptoSwift is a library of cryptographic functions, which was used to encrypt data prior to transmission.

iOS-Slide-Menu was used to create the side menu in the WCD application.

SwiftSpinner was used to display the spinner and progress updates during the whooping cough identification in the WCD application.

RecordButton was used to create the recording button in both applications. Its behaviour and appearance was modified to fit the purpose of these applications.

6.3 Back-end server

The back-end server has been implemented as a Linux, Apache, MySQL, PHP (LAMP) stack. This is the most widely used server-side set-up, and it is abundant among hosting providers. It has been chosen due to my previous experience with this stack, in order to speed-up the development.

APIs have been implemented as PHP scripts in order to allow the three applications to interact with the database.

WCD and WR applications communicate with APIs to create records in the database. These APIs are implemented in a similar manner, as outlined in Figure 6.12.

First, the encrypted binary data is received from the POST request using $file\_get\_contents(php://input)$. Next, the data is decrypted, as detailed in Figure 6.13.

First, it is necessary to convert the binary data into a string by decoding it as a base64 encoded string, which is achieved using PHPs native $base64\_decode$ function. Next, we extract the IV which was prepended to the data during the encryption. Decryption is then performed using PHPs native $mcypt\_decrypt$ function.

Finally, we strip any padding that has been added to the data during encryption. Padding might have been necessary to ensure that the data can be split into blocks of 16 bytes.

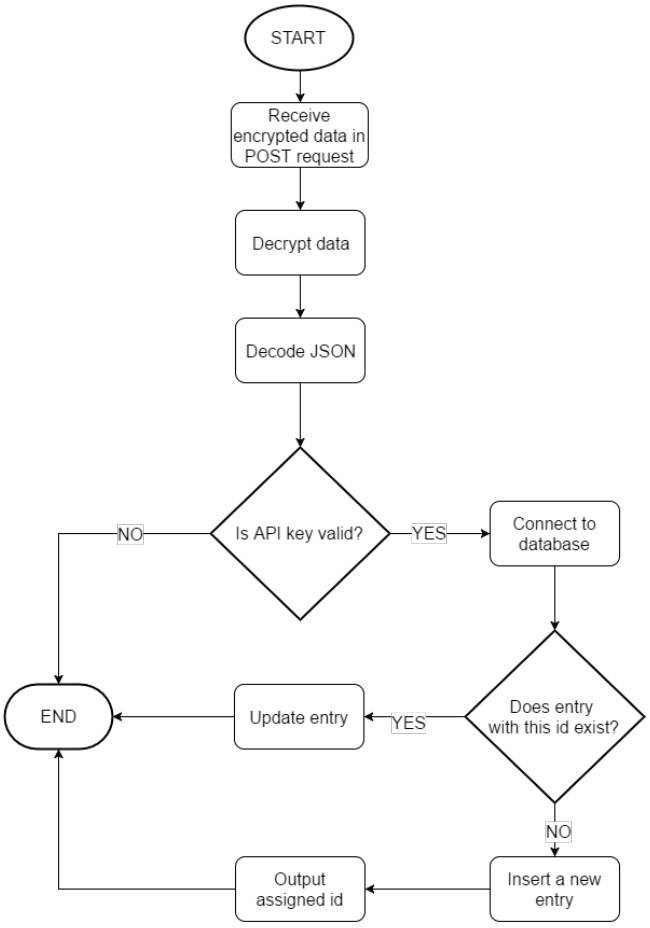


Figure 6.12: API implementation.

Once the data has been decrypted, we have a JSON object in a string format. Thus, it is necessary to parse the JSON string in order to access the encoded data as a native PHP object. This was achieved using the $json_decode$ function. However, if the received data was not encrypted using the private key, we are left with a random string that cannot be decoded, thus the script will exit. This ensures that the API is protected from the malicious requests.

Second line of defence against impersonation is the API key, thus the received API key is compared to the expected value. If the keys match, a connection to the database is established using PHPs mysqli class.

Next, the API checks whether the record already exists by attempting to select a record with the id provided through the request. If the selection returns a result, it is determined that the record exists, thus its contents are updated and the API exits. If the selection returns empty, the user-supplied data is inserted as a new row in the database, its unique system-wide

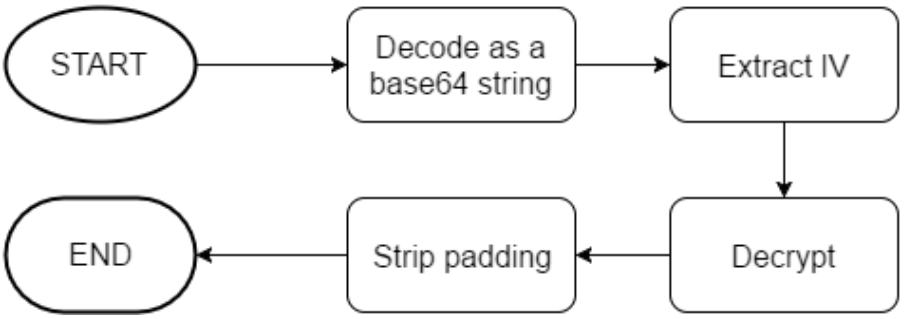


Figure 6.13: Decryption on the server.

identifier is noted and returned to the user as a response in JSON format.

Back-end server also must receive visualisation arrays and audio files from the iOS applications. These requests contain encrypted data in the body of POST request, and simultaneously a plaintext identifier and API key in the GET request. The API checks the key, gets data from POST, and id from GET, and decrypts data. Once the data is decrypted, it is saved in a JSON or WAVE format on the servers internal storage. During the applications launch, WCD checks if any data is missing from the server by sending a request to the checkDB API. This API receives the POST request, decrypts data to find the id and API key, and checks that the API key is valid. It then checks the database to determine whether the record exists, and checks the internal storage to determine if JSON and WAVE files exist. This API outputs a status code, which is either 0, 1, 2 or 3 in a JSON format. Status code 0 signals the application not to do anything about this record, which occurs when all data is present, or when the records doesnt exist. The latter scenario signifies that the record has been uploaded by the device, but it has since been deleted by the researcher. Status code 1 means that audio is missing, 2 means that JSON file is missing, and 3 means that both audio and JSON files are missing.

The Administration Panel allows its users to delete, download and visualise records, thus APIs to accomplish this are necessary. First, the deletion API checks whether the user is logged in, so to prevent malicious parties from deleting data on the server. This is accomplished by checking the corresponding session variable. Next, the API receives an id from the POST request, which represents the record to be deleted. This id is then used to delete the corresponding row from the database, and delete any JSON and WAVE files with that name using PHPs unlink function.

In order to enable the user to download audio files, the download API receives record id from the GET request, and then initiates a download stream. This is accomplished by setting the HTTP response header Content-Type to application/octet-stream, Content-Disposition to attachment, and filename to the id of the requested file. Finally, the file is read into the response stream using PHPs readfile function. The stream will be closed by the users browser once the download completes.

The back-end server also provides an API for the Administration Panel to download the JSON file containing identification arrays, which will be used to render visualisations. This is accomplished by receiving the record id from POST request, fetching the JSON file from internal storage using $file\_get\_contents$, and sending a response containing that file.

6.4 Administration Panel

The AP consists of three modules: authentication, WCD panel and WR panel. Authentication contains the login and signup screen, WCD panel displays data collected from the WCD application, and the WR panel displays data collected from the WR application.

Functionality and operational logic have been discussed extensively in the System Design chapter, thus this section will be dedicated to the most interesting implementation details.

The record tables within the WCD and WR panels have been implemented using the DataTables library, which provides the styling, pagination, sorting and search functionalities. Content of these tables is generated automatically during the page load. This has been accomplished by inserting a PHP script in the table body, which acts to retrieve data from the database and generate rows.

As you may recall, the symptoms were stored as dash-separated integers, as opposed to strings in order to minimise the bandwidth requirements, and facilitate localisation. The symptoms questionnaire is presented in the users language within the iOS application, thus storing symptoms as integers allows us to list these symptoms in English on the AP. This is achieved by

converting the dash-separated string into an array of integers using PHPs function explode, and retrieving each symptom from an array, using the integer as an index. Each row is generated by outputting the corresponding HTML from the PHP script, while substituting records from the database into each cell.

Audio playback is achieved by using the HTML5s audio tag, and setting the src attribute to the files location.

WCD panel allows the user to visualise the waveform and overlays from whooping cough identification. These waveforms are stored as a JSON file on the servers internal storage, which bears the records id as its name. In order to retrieve this file, a JavaScript function sends an AJAX request to the getData API on the server. The record id was passed to the server as a GET request, which is achieved by appending question mark and the id to APIs URL. For example, if we want to retrieve the JSON file for record with id of 29, we would send a request to ./getData.php?29. A completion handler is specified with the AJAX request, which is triggered once a response is received. Response status is then checked to determine whether the request has been successful. Request might be unsuccessful if the user has not yet uploaded their visualisation data due to poor internet connectivity, therefore the AP would warn its user that the visualisation data is missing. If the response has been successful, the chart is rendered.

Plotly.js library was used to render the visualisation, as it was specifically designed for visualisation of very large data sets. My previous attempts at using other libraries, such as Charts.js were unsuccessful, as they are unable to support data sets as large as the recording waveform. JavaScript supports JSON natively, thus individual arrays can be retrieved from the received JSON by their names. A trace for each dataset was then created, and the chart was rendered using Plotlys newPlot function.

Downloading in both panels was performed by redirecting the user to the downloading API, while passing the records id as a GET parameter. In effect, the user stays on the same page, but the downloading stream opens, and the user is able to download the audio file.

Chapter 7

Evaluation

7.1 Field usability test

Functionality of each component in the application suite was tested throughout the implementation, following the TDD framework, which ensured that every feature is ready for deployment before implementing other features. Therefore, the functional correctness of each application was determined by the end of implementation stage.

The primary purpose of field usability test was to heuristically find out how users, who were not involved in development of these applications, are able to navigate the user interface, and whether their interactions produce expected results. It was not possible to conduct these tests with pertussis patients, as this would require an ethics approval, and the incidence rate of whooping cough in London is very low.

7.1.1 Setup

I performed a series of field usability tests for the iOS applications with 15 subjects. 7 subjects were undergraduate engineering students, 5 subjects were non-technical people under 40 years of age, and 3 subjects were non-technical people over 40 years age. In this case, non-technical

subjects were those who did not undertake advanced studies in the fields of science, technology, engineering or medicine. Such mix was necessary to understand how users from different backgrounds interact with the applications. The 7 undergraduate engineering students are well-versed in technology, both from the users and developers point of view. The 5 non-technical people under 40 years of age also use technology in their everyday lives, and are interested in the newest consumer electronics trends. The 3 non-technical people over 40 years of age use technology in their everyday lives, albeit they do not follow the latest trends, and begin using new technologies once they has been proven successful.

For each application, I provided a brief overview of the applications purpose in order to simulate the experience of the user discovering the iOS application in the AppStore. I have also asked each user to imagine that they have a suspicious cough.

Field usability tests for the Administration Panel were conducted only with the 7 undergraduate engineering students, as their background most closely matched that of the hypothetical researcher. For each user, I provided an overview of the AP, which attempted to simulate the experience of a researcher being briefed by their supervisor.

7.1.2 Metrics

These field usability tests were not timed, nor time limited, although most users spent approximately 3 minutes on each iOS application, and 5 minutes on the AP.

During the Whooping Cough Detector and Whoop Research usability tests I recorded the subjects interactions with the applications. This was performed to determine whether the users have followed the envisaged flow, whether they entered their symptoms and diagnoses, how long were the recordings, did the subject use the playback feature, and what steps did the users take after the envisaged flow.

For the WCD, the envisaged flow consisted of starting with the Help screen, and navigating to Recording, Playback, Symptoms and Result screens. For the WR, the flow consisted of

starting with the Help screen, navigating to Recording screen, then to Playback, Symptoms and Diagnosis screens, and returning back to the Recording screen.

During the AP usability tests, the users interactions were also recorded to determine whether they were able to sign up successfully, and which features they have discovered.

7.1.3 Results

As expected, the linear flow design of the application meant that all subjects followed the envisaged flow. Most subjects have selected one or more symptoms on the symptoms questionnaire, which means that wording of the question was clear, and the multiple-choice list of symptoms was intuitive. This could also be attributed to the disabled Done button at the bottom of the questionnaire, which prompted users to select a symptom. Those users who didnt select any symptoms and pressed Skip instead, explained that they didnt want to select any symptoms, but were aware that they could.

Interestingly, only one technical user has selected a diagnosis. The other subjects explained that they werent aware that they were expected to select a diagnosis. This suggests that the placement or styling of the diagnosis section is not suitable to elicit diagnoses from the users.

5 of the 15 subjects used the playback feature. The rest explained that they were aware that the feature exists, but did not have any desire to play their recordings.

Judging from the recording lengths, it is clear most recordings tended to be approximately 8 seconds long, however there exists another peak at approximately 13 seconds. Subjects who created recordings of longer than 10 seconds explained that they didnt notice that they should touch the recording button to stop recording. Some expected the recording to stop sooner automatically.

The users action after the envisaged flow was recorded to explore their behaviour with regards to this application. Perhaps unsurprisingly, 6 out of 7 students performed a further action after seeing their result, as they were keen to explore the other features in WCD. Only 2 non-technical

subjects under 40 years of age have taken another action before exiting the application, and none of the subjects over 40 years of age was interested in exploring the application further.

In case of the WR application, the linear flow within the application resulted in all subjects following the envisaged flow.

On this occasion less subjects have selected their symptoms. This appears to be caused by the fact that those same subjects have already seen the WCD application, and entered their symptoms there, thus they were uninterested in repeating the exercise.

Only 4 out of 15 subjects have failed to enter their diagnosis using the WR application. Those who didnt explained that they found typing too tiresome, and preferred to press Skip instead. This change in behaviour could be explained by the fact the WR application features a separate Diagnosis page, as opposed to a section within another page.

None of the subjects used the playback feature in the WR application. This could also be explained by the fact that some subjects already used that feature in the WCD application, and were not interested in trying it again.

During the AP usability test, all subjects were able to create an account for this application. This is expected, as the sign-up screen is intuitive, and all subjects have technical backgrounds.

All subjects decided to open the Playback screen. This could be attributed to the simple design of the AP, and the bright colour of the Play button.

On the playback screen, all subjects opted to play the recordings and interact with the chart. The subjects explained that they were interested to see how the audio correlates with the waveform, and how individual events were identified.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

This project is mainly concerned with design and implementation of a software suite aimed at automatic identification of whooping cough and facilitation of further research in this area. The application suite consists of the Whooping Cough Detector iOS application, Whoop Research iOS application and a web-based Administration Panel.

The Whooping Cough Detector is able to record cough, perform automatic whooping cough identification, request users to provide their symptoms and diagnoses, and upload the collected data onto the research server even with periods of limited internet connectivity. Field usability tests showed that this application is intuitive to use, as every test subject was able to use it to record their cough and receive the identification result. This application is much faster than the laboratory tests, doesnt require specialists involvement, and is virtually free, provided that the user has access to a suitable device.

The Whoop Research application is able to record cough, request users to answer symptoms and diagnosis questionnaire, and upload the collected data onto the research server. Field usability tests indicated that this application is successful at eliciting recordings, symptoms and diagnoses from its users. Traditionally it is quite difficult to gather data for research, as it involves

distributing fliers and inviting participants to record their cough and answer questionnaires at the research facility. The WR application alleviates these problems by enabling researchers to recruit participants electronically via AppStore, and enabling the participants to donate data from their smartphones.

The Administration Panel enables researchers to visualise the data collected from both WCD and WR applications, play and download recordings. Field usability tests showed that panel is intuitive to use, as all subjects were able to sign up and discover the key features. Traditionally, research teams would receive data on paper forms that had to be manually entered into the database, and evaluation of any diagnostic tool made available to the public relied on explicit feedback. The AP is able to display data, play recordings and visualise performance of the whooping cough identification automatically, as soon as this data is recorded on the users device.

8.2 Limitations and Future Work

Field usability tests showed that the current layout is not successful at eliciting diagnoses from the users of WCD application. This could be improved by requesting the diagnosis using a separate screen, as implemented in the WR application.

Some improvements could be made to speed up the whooping cough identification algorithm, as currently the users may spend up to 5 minutes waiting for their identification result. For instance, currently the MFCCs are computed prior to sound event detection, which results in many unnecessary calculations. Therefore, performing the MFCC computation after sound event detection would result in a faster identification. Additionally, the current implementation uses frames of 5120 samples for MFCC calculation and feature extraction, which is not a power of 2, resulting is a need to perform a more time-consuming Chirp-Z transform in place of FFTs. Modifying the frame size to become a power of 2, could potentially result in a three-fold performance improvement.

Some security limitation need to be addressed prior to a full-scale deployment of this system.

Currently the authentication on AP is performed over unsecure connection, thus it would be necessary to use SSL for any communications with the server. The server is currently hosted on a shared hosting, which has its own security limitations, thus it would be worthwhile to host the back-end server on a secure network.

Currently, the WR records might not be uploaded if the internet connection is poor. This problem could be fixed by storing the records locally, and attempting to upload them when the internet connection is more reliable, as implemented in the WCD application.

Appendix A

Appendix

In order to obtain the full code listing, please contact Dr Esther Rodriguez-Villegas at Imperial College London.

Bibliography

[1] Apple. Concurrency and Application Design. https://developer.apple.com/library/mac/documentation/General/Conceptual/ConcurrencyProgrammingGuide/ConcurrencyandApplicationDesign/ConcurrencyandApplicationDesign.html, 2012. [Online; accessed 13-June-2016].

[2] M. Bellare, J. Kilian, and P. Rogaway. The security of the cipher block chaining message authentication code. *Journal of Computer and System Sciences*, 61(3):362–399, 2000.

[3] N. S. Crowcroft and R. G. Pebody. Recent developments in pertussis. *The Lancet*, 367(9526):1926–1936, 2006.

[4] J. Hamborsky, A. Kroger, et al. *Epidemiology and Prevention of Vaccine-Preventable Diseases, E-Book: The Pink Book*. Public Health Foundation, 2015.

[5] NHS. Treatment for whooping cough. http://www.nhs.uk/Conditions/Whooping-cough/Pages/Introduction.aspx#treatment, 2016. [Online; accessed 13-June-2016].

[6] A. of Public Health Laboratories. Treatment for whooping cough. http://www.aphl.org/AboutAPHL/publications/Documents/ID_2010May_Pertussis-Diagnostics-Brochure.pdf, 2010. [Online; accessed 13-June-2016].

[7] W. H. Organisation. World - By cause. http://apps.who.int/gho/data/node.main.CODWORLD?lang=en, 2015. [Online; accessed 13-June-2016].

[8] W. H. Organisation. World - By cause. http://apps.who.int/gho/data/node.main.PROJNUMWORLD?lang=en, 2015. [Online; accessed 13-June-2016].

[9] W. H. Organisation. World - Pertussis. http://apps.who.int/gho/data/view.main.CM100WORLD-CH4?lang=en, 2015. [Online; accessed 13-June-2016].

[10] D. Parker, J. Picone, A. Harati, S. Lu, M. H. Jenkyns, and P. M. Polgreen. Detecting paroxysmal coughing from pertussis cases using voice recognition technology. *PloS one*, 8(12):e82971, 2013.

[11] G. Peeters, B. L. Giordano, P. Susini, N. Misdariis, and S. McAdams. The timbre toolbox: Extracting audio descriptors from musical signals. *The Journal of the Acoustical Society of America*, 130(5):2902–2916, 2011.

[12] R. X. A. Pramono. System for Remote Identification of Whooping Cough. MSc. Imperial College London, 2015.

[13] L. R. Rabiner, R. W. Schafer, and C. M. Rader. The chirp z-transform algorithm. *Audio and Electroacoustics, IEEE Transactions on*, 17(2):86–92, 1969.

[14] A. M. Salim, Y. Liang, and P. E. Kilgore. Protecting newborns against pertussis: Treatment and prevention strategies. *Pediatric Drugs*, 17(6):425–441, 2015.

[15] K. M. Scanlon, C. Skerry, and N. H. Carbonetti. Novel therapies for the treatment of pertussis disease. *Pathogens and disease*, 73(8):ftv074, 2015.

[16] S. Walke and V. Thool. Differentiating nature of cough sounds in time domain analysis. In *Industrial Instrumentation and Control (ICIC), 2015 International Conference on*, pages 1022–1026. IEEE, 2015.

[17] K. Winter, J. Zipprich, K. Harriman, E. L. Murray, J. Gornbein, S. J. Hammer, N. Yeganeh, K. Adachi, and J. D. Cherry. Risk factors associated with infant deaths from pertussis: a case-control study. *Clinical Infectious Diseases*, 61(7):1099–1106, 2015.

[18] N. Wood and P. McIntyre. Pertussis: review of epidemiology, diagnosis, management and prevention. *Paediatric respiratory reviews*, 9(3):201–212, 2008.