

Word-Counting app to Assist Speech Therapists in Assessing Children with DLD

Sinéad O'Sullivan

Final-Year Project - BSc in Computer Science

Supervisor: Dr Ian Pitt

Department of Computer Science

University College Cork

April 2021

Abstract:

Many children suffer from Developmental Language Disorder (DLD), which is associated with poor literacy skills in later life, limited employment opportunities, and mental health problems. Effective treatment for DLD exists, but accurately assessing individual needs presents problems. The current standard test involves counting the number of words a child learns over a given period of therapy. Research suggests that other variables should be taken into account e.g. how many times a child was exposed to a word before learning its meaning.

My Word Counter is an app that will record a child speaking target words during therapeutic activities. The app is able either to count the total number of utterances of the words and alert the users when a predefined total or time limit has been reached. Data gathered by the app is stored in a database, with provision for therapists to search and view the results.

This project is being conducted in collaboration with staff from the Department of Speech and Hearing Sciences at UCC.

Declaration:

Declaration of Originality

In signing this declaration, you are conforming, in writing, that the submitted work is entirely your own original work, except where clearly attributed otherwise, and that it has not been submitted partly or wholly for any other educational award.

I hereby declare that:

- this is all my own work, unless clearly indicated otherwise, with full and proper accreditation;
- with respect to my own work: none of it has been submitted at any educational institution contributing in any way to an educational award;
- with respect to another's work: all text, diagrams, code, or ideas, whether verbatim, paraphrased or otherwise modified or adapted, have been duly attributed to the source in a scholarly manner, whether from books, papers, lecture notes or any other student's work, whether published or unpublished, electronically or in print.

Signed: .. Sinead O' Sullivan ..
Date: 21/04/2021

Acknowledgments:

I would like to begin by thanking my supervisor Dr. Ian Pitt for his continuous support and guidance throughout the year. I would like to express my gratitude to Dr. Pauline Frizelle, from the Department of Speech and Hearing Sciences, for her invaluable contributions to this project. I would like to thank my second reader, Dr. Sabin Tabrica, for his advice on the project. I would also like to acknowledge the staff in the Department of Computer Science for nurturing my love of computer science and giving me the opportunity to grow in knowledge during my studies at University College Cork.

Table of Contents

1.	Introduction	1
2.	Analysis	3
2.1.	Goals	3
2.2.	Objectives	3
2.3.	Proposed System	4
2.4.	System Users.....	4
2.5.	Requirements Analysis.....	5
3.	Design	16
3.1.	Database	16
3.2.	Mobile Application Device.....	18
3.3.	Website and App Distribution	18
3.4.	Languages and libraries	20
3.5.	Tools.....	21
3.6.	Speech Recognition	22
3.7.	Usability	25
3.8.	Accessibility.....	26
3.9.	Security	27
4.	Implementation	28
4.1.	Database	28
4.2.	Web Application	29
4.3.	Mobile Application.....	32
5.	Evaluation	42
5.1.	Testing Process	42
5.2.	Evaluation of Final System.....	45
5.3.	Remaining Issues	50
6.	Conclusion.....	52
6.1.	Summary of the Final Product	52
6.2.	If more time was available.....	52
6.3.	Evaluation of development process:	53
6.4.	What was learned	55
	Appendix	ii

1. Introduction

Developmental Language Disorder (DLD) is a type of speech, language and communication need that affects the way that children understand and use language. DLD affects 7.58% of children and usually leads to difficulties in literary skills, mental health issues, and a harder time finding a job later in life. Speech and language therapy is central to helping children and adults with communication difficulties better understand and use words. Speech therapists teach strategies to children with DLD and those around them, which aim to reduce the impact of their difficulties and develop their language abilities to their maximum potential. These treatments are effective, however, it can be difficult to determine the needs of individual patients. This project aims to create a system called *My Word Counter* that will aid speech therapists in the treatment of Developmental Language Disorder, specifically focusing on children. This project will be worked on in collaboration with the Department of Speech and Hearing Sciences at UCC.

The existing test for accessing the individual needs of children with DLD involves counting how many words a child uses during a therapy session. Research suggests that more factors should be included in the assessment. These factors include how and when the patient first encountered the new word, the number of word exposures needed before the patient began to understand the word, the age of the patient, and the clinical group that the patient belongs to. Detailed word-counting during a traditional therapy session is logistically challenging and is not part of current clinical practice. *My Word Counter* is designed to overcome these challenges and better the lives of children with DLD.

My Word Counter aims to determine the speed at which words are learned by counting how many times a word is used by a child before it is understood. The system will facilitate this through a mobile application, downloadable by parents or guardians, and will allow for remote collection of word exposure data during home-based focused learning sessions. The parent or guardian of a child will be able to create their own account, design their own sessions using the app's easy user-friendly interface, and manage their child's data. This mobile application will implement the latest Speech Learning technologies to count words spoken during a session. Additionally, this technology will allow parents and guardians to focus on interaction during a word-counting session, while simultaneously gathering word exposure information.

Word exposure data recorded during these home-based learning sessions can be viewed by a child's therapist. This data can be used by the therapist to individualize treatment plans for each child and may facilitate a reduction in treatment time for some patients. This session data will also be available (anonymously) to researchers in the speech and language field and will assist in determining the optimal number of word exposures needed to achieve understanding in patients of different ages and clinical presentation. Researchers and therapists will be able to access session word-exposure data through *My Word Counter's* online web application.

My Word Counter hopes to provide researchers and therapists with word exposure data that will transform the research landscape by improving both the quality and efficacy of Speech and Language Therapy (SLT) delivery. This project is incredibly important as it has the potential to improve the lives of people living with Developmental Language Disorder all over the globe.

2. Analysis

2.1. Goals

The main goal of this project is to aid in assessing the individual needs of children with *Developmental Language Disorder*, by taking into account the following variables:

- Word exposures: the number of times the child heard a word before starting to understand it
- Context of the intervention, for example, book reading vs. a play session
- Individual variation in the number of word exposures required for understanding
- Variation in word exposures before understanding among different age groups
- Variation in word exposures before understanding among different clinical groups

2.2. Objectives

Objective 1: To develop a user-friendly mobile application that will quantify the speed of learning by utilizing Speech Recognition technology to calculate the number of times specified words are used before it is evident that the child has achieved understanding

Objective 2: To allow therapists and parents to maintain the focus of sessions on interaction with the child while simultaneously gathering word exposure information

Objective 3: To facilitate remote collection and storage of word exposure data during home-based focused learning sessions.

Objective 4: To develop an online web application that will allow speech therapists to access word exposure data such that they can determine the individual needs of their clients

Objective 5: To develop an online web application that will allow researchers to access anonymous word exposure data so that they can analyze it to discover the optimal

number of word exposures needed to achieve understanding in patients of different ages and clinical presentation.

Objective 6: To develop mechanisms that will give the parent or guardian control over any personal data recorded about their child, and protect the personal data of the child.

2.3. Proposed System

The proposed solution is thus: *My Word Counter*, a system with two main interfaces: an application that can be downloaded on mobile, and an online web application.

The mobile application will facilitate the creation of and participation in remote sessions, during which words will be counted using Speech Recognition. Word counting will continue for either a fixed timeframe (for example, a 30-minute play or therapy session) or to a specified word count (for example, 10 incidences of word use). At this point, the device will signal to the user (by buzzing or an auditory signal) that it is no longer counting. Word counting information is stored on a remote server and can be accessed or removed by the parent/guardian of the child via the mobile app. The web application will be available online and will facilitate therapist and researcher access to session data. The focus of *My Word Counter* is on word understanding and has applications for both children and adults with communication difficulties. The initial focus of the application will be on the pediatric population.

2.4. System Users

2.4.1. Parent/Guardian

These are the parents or guardians of children with *Developmental Learning Disorder*. They will manage their sessions for the child, as well as managing the child's data.

2.4.2. Researcher

A researcher is an expert who is aiming to use *My Word Counter* to analyze word exposure data to determine the quantity and context of word exposure needed before a word is understood.

2.4.3. Speech Therapist

A speech therapist has access to the same information as researchers, but additionally, intends to use the system to determine the individual needs of their clients.

2.5. Requirements Analysis

2.5.1. Functional Requirements

2.5.1.1. Web Application:

The web application will allow therapists and researchers to log in and view session data or data related to their clients. This application will not allow parents/guardians to log in and view their account data: this account data can be viewed on the mobile application.

2.5.1.1.1. Landing Page:

- A user who is not logged in to the mobile application will be presented with a short description of the system and given the options to download the app or create an online therapist/researcher account
- **Create an account button:** The user will be brought to the applications sign-up page, where they can make a new account to start using the website
- **Download app button:** The user will be brought to a page where they can download the mobile application

2.5.1.1.2. Log-in page

- The user will be presented with a form where they can submit their email and password to log in

- **Sign-up button:** Brings a user to the sign-up page, where they can make a new account to start using the website
- **Forgotten password:** A user can request an email to allow them to reset their password

2.5.1.1.3. Sign-up page

- The user will be presented with a form that they can use to create a new account with the following fields: Name, Surname, email, password
 - The 'name' and 'surname' fields can only contain letters
 - The 'email' field must be a valid email
- **Create button:** once clicked, the user account will be created
- Once the user account is created, the user will be logged in automatically to the home page

2.5.1.1.4. Home

- A simple landing page for the logged-in user: contains links to other parts of the website
 - Clients
 - My Account
 - Sessions

2.5.1.1.5. Clients

- Allows a user who has registered clients to view data about their clients: this data cannot be modified.
- The clients are shown in a table, with an option to view more detailed information about the client
- **Add client button:** allows a user to add a new client using a specially generated code
 - When clicked, the user will be presented with an input field, where they can type the code for adding their client

- The user will have to consent to share their name, surname, and public ID with the client
- **Add button:** when clicked, this will send a message to the client's account, asking them to confirm that this is the correct therapist

2.5.1.1.6. Clients: More information

- Shows more detailed information about a specific client including the sessions participated in by that client
- **Remove client button:** allows a therapist to remove a client from their account (this does not delete the client's account)

2.5.1.1.7. Account data

- The user can view their account data
- **Edit button:** The user will be brought to a page that allows them to edit their account data
- **Delete account button:** The user can delete their account
 - When clicked, the user will be asked to confirm the decision before their account is deleted

2.5.1.1.8. Modify account data

- Modifiable fields: name, surname, email, password
 - The 'name' and 'surname' fields can only contain letters
 - The 'email' field must be a valid email
- **Save button:** once clicked, the account data is updated
- **Cancel button:** allows the user to return to the previous (account data) page

2.5.1.1.9. Sessions

- A landing page for the 'sessions' section of the website

- View all completed sessions: This allows a user to view all sessions that have ever been completed
- View sessions completed by clients: If a user has registered clients, they can view any sessions completed by that client while that client is still registered with them

2.5.1.1.10. View Sessions Completed by Clients

- If a user has registered clients, they can view any sessions completed by each client while that client is still registered with them
- The information displayed for each session will consist of:
 - Name/title of the session, the public ID of the client, recorded words and their word counts, time taken, time and date the session was recorded, age of client when the session was recorded, any additional notes

2.5.1.1.11. View All Sessions (anonymous)

- Allows a user to view all sessions that have ever been completed
- All data displayed will be completely anonymous
- The information displayed for each session will consist of:
 - recorded words and their word counts, the public ID of the client, time taken, time and date the session was recorded, age of client when the session was recorded, any additional notes

2.5.1.2. Mobile Application:

This will allow adults working with children to record sessions where they focus on using certain words

2.5.1.2.1. Log-in screen

- The user will be presented with a form where they can submit their username and password to log in

- **Sign-up button:** A user can make a new account to start using the website
- **Forgotten password:** A user can request an email to allow them to reset their password

2.5.1.2.2. Sign-up screen

- The user will be presented with a form that they can use to create a new account with the following fields: Name, Surname, email, password, date of birth of the child, a space to enter learning disabilities the client may have
 - The 'name' and 'surname' fields can only contain letters
 - The 'email' field must be a valid email
 - The date of birth must be a valid date between today and 1950
- **Create button:** once clicked, the user account will be created
- Once the user account is created, the user will be logged in automatically to the home page
- **Back button:** allows the user to return to the previous (log in) screen

2.5.1.2.3. Home screen

- **Record session button:** This allows a user to immediately begin a session from the home screen
- **Account data button:** Brings a user to the 'account data' screen where they can view/edit their information
- **View previous sessions button:** This allows a user to view previously recorded sessions
- **View session templates button:** This allows a user to view saved session templates that they have already created

2.5.1.2.4. Account data screen

- The user can view their account data
- **Edit button:** The user will be brought to a page that allows them to edit their account data

- **Reset password** button: The user can ask to reset their password
- **Delete account** button: The user can delete their account
 - When clicked, the user will be asked to confirm the decision before their account is deleted
- **Therapist section:**
 - The user can only be linked to one therapist at a time
 - If the user is not linked to a therapist, they can choose to generate a code that allows a therapist to link to them
 - The user has to be warned that their name, age, and session data will be shared with their therapist
 - This code expires after 24hrs and can only be used once
 - If the user has received a 'request' to link up with a therapist, they can see the name, surname, and ID of the therapist and can choose to 'accept' or 'decline' the request
 - If the user has a therapist:
 - The user can see the name, surname and, ID of their therapist
 - **Delete button:** The user can remove the therapist from their account (the therapist will lose access to the client's data)
- **Back button:** allows the user to return to the home screen
- **Sign out button:** allows the user to sign out of the app

2.5.1.2.5. Modify account data screen

- Modifiable fields: Name, Surname, email, date of birth of the child, a space to enter learning disabilities the client may have
 - The 'name' and 'surname' fields can only contain letters
 - The 'email' field must be a valid email
 - The date of birth must be a valid date between today and 1950
- **Save** button: once clicked, the account data is updated
- **Back button:** allows the user to return to the home screen

2.5.1.2.6. Saved session templates screen

- A session template is a previous session created by the user, that has been saved so that it can be used again
- Consists of a clickable list of session templates – when a session template is clicked on, it opens a screen with more information about the session template
- **Back button:** allows the user to return to the previous (home/record new session) screen

2.5.1.2.6.1. Ability to view each session template's data

- Displays information about the session:
 - Name/title of the session, words the app is to listen for, whether the app is ended by a time limit (in which case, specifying the time limit) or when a certain word count for each word is reached (again, specified by the user), any additional notes added by the user
- **Back button:** allows the user to return to the previous (saved templates) screen
- **Start session button:** allows the user to start a new session using this template
- **Delete button:** allows a user to delete their session template -> any sessions recorded using this template will be kept
 - When clicked, the user will be asked to confirm the decision before their session template is deleted

2.5.1.2.7. Previously recorded sessions

- Allows a user to view sessions they have previously recorded
- Consists of a clickable list of session templates – when a session is clicked on, it opens a screen with more information about the session
- **Back button:** allows the user to return to the home screen

2.5.1.2.7.1. Ability to view each session's data

- Displays information about the session:

- recorded words and their word counts, time taken, criteria that ended the session (time limit, word limit or 'stopped manually'), time and date the session was recorded, age of client when the session was recorded, any additional notes added by the user
- **Back button:** allows the user to return to the previous (recorded sessions) screen
- **Delete button:** allows a user to delete a previously recorded session. This will remove the session data from the database.
 - When clicked, the user will be asked to confirm the decision before their session template is deleted

2.5.1.2.8. Record a new session

- The user is presented with two options (buttons):
 - **Use a session template:** bring the user to the session template screen
 - **Create a new session:** bring the user to the create a session screen

2.5.1.2.9. Create a session screen

- Allows the user to create a session
- The user will be presented with a form, which they will have to fill in by typing/selecting their input.
- Input fields:
 - Name/title of the session, words the app is to listen for, whether the app is ended by a time limit (in which case, specifying the time limit) or when a certain word count for each word is reached (again, specified by the user), a space for any additional notes
 - Name/title: must only contain letters and numbers
 - Words: a list of words (separated by a space), can only contain letters
 - Time limit: must be a valid time that is less than 24hrs
 - Word count: must be an integer greater than 1
- **Save as template? button:** saves the session as a re-usable session template if checked

- This will include all of the data just inputted by the user
- This will also include a 'date created' field and a 'session template ID' field
- The user will be warned that anonymous data about their sessions will be visible to anyone who makes an account on the web application.
- **Back /cancel button:** allows the user to return to the previous (recorded new) screen. All inputted data will be lost.
- **Start recording:** Starts a recorded session, if the required input fields are filled correctly

2.5.1.2.10. Recording a session

- While recording a session, the screen will show the following:
 - Time since the session started
 - Time limit (or 'no time limit')
 - Words being counted and their current count
 - The count we are looking for, for each word (if specified in the session set up)
 - **Pause/resume session button:** This allows a user to pause a recorded session. The app will not record any data while the session is paused. A paused session can be resumed or stopped.
 - **Stop session button:** allows the user to end the session immediately, without waiting for the time limit/word count to be reached
- A recorded session will listen to words spoken near the device running the session, and identify and count the key words as specified by the session
- A session will not stop if the app is exited or closed
- A session will only stop under the following circumstances: The time limit has been reached, the word count has been reached, the user pressed the stop button, the device was powered off

2.5.1.2.11. Finishing a session and saving the data

- When a session has been completed, the user can choose to add additional notes to the session before it is saved. These notes can be inputted into a text box on the screen, by typing. These notes will not be added to any saved session template. They can then 'continue'.
- the session data will then be saved to the device and the database
 - this will include all of the data imputed by the user -> Name/title of the session, words the app is to listen for, whether the app is ended by a time limit (in which case, specifying the time limit) or when a certain word count for each word is reached (again, specified by the user), a space for any additional notes
 - this will also include: the session ID, the public ID of the client, time and date of recording, age of client when recording was taken, time taken, number of each word counted
- If the device is offline, the session data will be stored and saved whenever the device comes back online
- Once the session data has been saved, a banner will inform the user that their recorded data has been saved. They can either view the saved session (brings the user to the corresponding 'view session data' screen) or return to the home screen.

2.5.2. Non-Functional Requirements

2.5.2.1. User interface and Human Factors

- 2.5.2.1.1. The users of this system will not need any specific training
- 2.5.2.1.2. The user interface should be easily navigable and have the core elements prominently displayed.
- 2.5.2.1.3. Terms and icons should be consistent on all pages/sections.
- 2.5.2.1.4. Wherever possible, errors by the user should be kept to a minimum, this can be achieved by ensuring that forms contain error reporting

2.5.2.2. Hardware Consideration

- 2.5.2.2.1. The website application will require the user to have a system capable of connecting to the internet and running a web browser.

- 2.5.2.2.2. The mobile application will provide some offline functions (such as the recording of sessions) but will require internet access for other operations (such as logging in and updating the remote database).

2.5.2.3. Security Issues

- 2.5.2.3.1. All data gathered will be stored securely
- 2.5.2.3.2. Only authorized users have access to data relevant to them
- 2.5.2.3.3. Secure authentication of users will be enforced

2.5.2.4. Physical Environment

- 2.5.2.4.1. The online website will be hosted on a secure online server
- 2.5.2.4.2. The database and authentication data will be stored securely on a secure online server
- 2.5.2.4.3. The mobile application will be available for download from the online application

3. Design

3.1. Database

3.1.1. Choosing the best database for *My Word Counter*

Careful consideration was given to the choice of database. Based on the recommendations of past computer science students and online forums, it was determined that it will be better to use Firebase¹ rather than creating a database on a server using MySQL or some other database language.

The advantages of choosing Firebase for this project are the following:

- Firebase has premade functions for adding, updating, retrieving, and deleting data for web applications and mobile applications.
- Firebase's authentication facilitates the creation of a login system with secure authenticated user accounts.
- Firebase's website contains easy-to-follow documentation.
- Firebase supports Machine Learning
- Firebase is free for small-scale applications
- Quick and easy to set up

Firebase has two database options to choose from: Cloud Firestore and the Realtime Database

Cloud Firestore	Realtime database
Stores data as a collection of documents that are very similar to JSON	Stores data in a JSON tree
Offers offline support for complex queries on all devices	Offers offline support for iOS and Android only
Supports complex queries	Queries are indexed, but cannot be both sorted and filtered at the same time
Has a limit on write rates to individual documents or indexes.	No local limits on write rates to individual pieces of data.

¹ <https://firebase.google.com/>

Does not support the ability to record a client's connection status	Has the ability to record a client's connection status
---	--

Firebase was chosen because of its wider offline support, its support for complex queries, and its support for more complex document structures.

3.1.2. Database Structure

Cloud Firestore's data is made up of JSON-like documents, organized into collections. The database for *My Word Counter* is designed with 5 collections of documents as can be seen in figure 1 in the appendix. The *completed_sessions* collection will contain documents that store word exposure data from remote sessions and the *session_templates* collection will store sessions that a parent/guardian may want to re-use. The *users* collection will store the personal data of both users of the mobile app (clients) and users of the web app (therapists/researchers). The *temp_codes* collection will facilitate a mechanism that allows a mobile app user to specify add therapist using a one-time expirable code. The *public_ID* collection will help to ensure that no two users are assigned the same public ID.

3.1.3. Security: Authentication and Access Control

Firebase provides many secure authentication options and facilitates the control of access to data through its access control rules. Email authentication will be used for this project.

The access rules will be set to ensure that a user can only access data that is necessary to fill their basic requirements, following the principle of least principle, therefore providing maximum protection of data. These rules will include the following:

- A user will be required to authenticate themselves before reading data from any of the collections, except the *public_ID* collection which is needed during the sign-up process.
- A user can only modify documents that relate to them.
- A web user can only access the personal data of their clients but is granted access to all completed sessions.
- Session templates can only be read by the user who created them.
- All authenticated users will require access to *temp_codes*.

These rules will be defined in Firebase, and cannot be modified at the discretion of other users.

Additionally, a user must be re-authenticated before performing critical operations such as updating data related to authentication or deleting their account. This is intended to add an extra layer of security that will prevent, for example, a person opening a device where a user has been logged in and performing a critical operation.

3.2. Mobile Application Device

The Covid-19 pandemic resulted in a closure of the Western Gateway Building's Computer Science labs to students this year, and as a result, the options for developing the mobile application for *My Word Counter* were limited to technologies already owned. Developing an iOS application requires a computer running a macOS operating system, and developing an app for a FitBit would require a FitBit or a computer that is capable of running an emulator without crashing. None of these options were freely available, so it was determined that the app would be developed for an Android mobile phone device.

3.3. Website and App Distribution

3.3.1. Server

The web server for this project has been chosen carefully, taking into account security, trustworthiness, and cost, in addition to the type of hosting offered by the server. Paid servers were not considered for this project. The options considered for a web server were the following:

University College Cork's (UCC) Computer Science (CS) Servers	
Advantages	Disadvantages
Secure	Will not be available to the public beyond the end of this year
Trustworthy	

Allows dynamic hosting	
Free	

Firebase Hosting	
Advantages	Disadvantages
Secure	Only allows static hosting
Trustworthy	
Available indefinitely	
Free	

UCC's CS server will be used for this project because it is the only free and trustworthy option that supports dynamic hosting.

3.3.2. Mobile Application

The two options for publishing the mobile application are by releasing the app on a marketplace such as Google's Play Store or making it available for download from the *My Word Counter* website.

Play Store	
Advantages	Disadvantages
Easy accessibility to users	Requires paying a developer's fee
Provides a licensing service to prevent unauthorized downloads	
Provides services to track financial transactions	

My Word Counter website	
Advantages	Disadvantages

Free	Requires a user to accept downloads of apps from unknown sources
	Lacks the features provided by a marketplace

For this project, it was determined that releasing the mobile application on the project's website free of charge would suffice.

3.4. Languages and libraries

3.4.1. Web application

As discussed previously, the back-end of the web application for *My Word Counter* will be handled using Firebase. The front end of this web application will be written using a combination of HTML, JavaScript, and CSS.

Notable libraries and plug-ins that will be used are:

- **DataTables**²: a useful tool for displaying tables of information using JavaScript.
- **JQuery**³: a JavaScript library that simplifies and improves JavaScript coding.

3.4.2. Mobile Application

The two languages for coding for Android are Kotlin and Java.

Java	
Advantages	Disadvantages
Lots of online documentation and open-	Slower to compile than Kotlin

² <https://datatables.net/>

³ <https://jquery.com/>

source tools	
It is impossible for Java instructions to corrupt memory or compromise data from other applications of OS X	
Lots of online libraries	

Kotlin	
Advantages	Disadvantages
More type-safe than Java: there are no raw types and no checked exceptions	Less online documentation than Java
Faster to compile than Java	
Less verbose = fewer bugs Safe against NullPointerException	
100% interoperable with java i.e. can use Java libraries	

Kotlin was chosen because of its compilation speed and because it is more type-safe than Java.

3.5. Tools

3.5.1. Version Control

Version control is a useful tool in any software development project. It allows a developer to track changes that they make to their code, reverse any changes that harm the project, and, in the case of teamwork, allows many developers to integrate their code. Git will be used with GitHub to enforce version control for this project and ensure that all code is backed up safely, in this case in a Git repository on GitHub, if something happens to the laptop that is being used to develop the project. Additionally, using Git will simplify the process of updating the web server with new changes made to the web or mobile applications.

3.5.2. Website Application Development Environment

Multiple choices exist for free code editors that support HTML, CSS, and JavaScript.

Atom	Sublime	Visual Studio (VS) Code
Available on Windows	Available on Windows	Available on Windows
Allows a user to install packages such as auto-complete etc	Allows a user to install packages, including a package to help manage packages	Allows a user to install packages such as a debugger for chrome
Can be quite slow	Faster than Atom, similar to VS code	Faster than Atom, similar to Sublime
Git integration can be installed as a package	Git integration can be installed as a package	Git integration included

VS Code has been chosen for its debugging capabilities and version control features.

3.5.3. Mobile Application Development Environment

An integrated development environment (IDE) called Android Studio will be used to aid in developing the android app for *My Word Counter*.

Android Studio	
Advantages	Disadvantages
The official Android IDE	It is not lightweight i.e. it consumes large amounts of memory and CPU
Developed by JetBrains (creators of many popular IDEs) with Google	
Many features including auto-completion, support for version control, and a debugger	

3.6. Speech Recognition

Speech recognition involves two distinct processes: word recognition and deciphering of meaning. *My Word Counter* is more focused on the word recognition part of this process.

The most popular way to recognize spoken words is to take a waveform and split it into utterances, divided by silences. The model then tries to determine the word being spoken in each utterance, by choosing the most likely word based on its features. The model usually used for speech is called Hidden Markov Model (HMM). A speech model will use a combination of acoustic models (based on acoustic models for senones), phonetic dictionaries (which map words to phones), and language models (which restrict words based on previously recognized words e.g. the word 'the' is unlikely to be repeated twice in a row in a sentence) to help it to determine what word the utterance is most likely to be. Speech recognition models never have 100% accuracy. This is due to several factors, including accents, background noise interference, and the huge number of phonetically similar words to choose from.

The main decision that had to be made in regards to speech recognition was whether to design a new speech recognition model or use an existing model. The model must be functional without an internet connection i.e. implemented on the device itself. Additionally, the speech recognition functions must recognize words continuously i.e. to increment the counts of words as the speaker is talking, without requiring a pause between each word.

Time was spent learning how to create models using Machine Learning, however, a speech model is quite complex and requires a lot of data and training. There are a lot of factors to consider, for example, bias, where the type of accent, voice, etc used in training and testing might influence the results of the model.

Training own speech recognition model	
Advantages	Disadvantages
Full control over the data that the model is being trained on	Time-consuming
Java and Kotlin both have libraries that support Machine Learning e.g. Smile, DeepLearning4J	Accuracy will be lower than existing models
If written using the TensorFlowLite library, can be integrated into the app	Would have to design functions to apply

using Firebase's Machine Learning functions	the model to continuous listening
---	-----------------------------------

One way of increasing the accuracy of a speech recognition model would be to build a model from a pre-existing model and combine it with a model trained specifically for *My Word Counter*.

Adapting an existing Speech Recognition Model	
Advantages	Disadvantages
Java and Kotlin both have libraries that support Machine Learning e.g. Smile, DeepLearning4J, which can be used to alter the model	Would have to design functions to apply the model to continuous listening
Less time consuming than training own model	Less control over the data that the model is being trained on
Accuracy will be a lot higher than a model that is trained from scratch	
If written using the TensorFlow Lite library, can be integrated into the app using Firebase's Machine Learning functions	

After researching some online forums such as Stack Overflow, it was discovered that there are tools with pre-trained models that support continuous speech recognition. CMUSphinx⁴ has an open-source library called PocketSphinx that does this on Android.

PocketSphinx	
Advantages	Disadvantages
Less time consuming than training own model	No control over the data that the model is being trained on
Works in offline mode	Cannot easily modify the model to

⁴ <https://cmusphinx.github.io/wiki/about/>

	improve its accuracy
Accuracy will be a lot higher than a model that is trained from scratch	
Already contains functions for continuous listening on Java (also operable in Kotlin), and can be directly integrated into the mobile app	
Fast and portable	

PocketSphinx was chosen for *My Word Counter* because of its reasonably high accuracy, its support for continuous listening, and the time constraints that exist for this project.

3.7. Usability

Efforts have been made to develop a simple interface for *My Word Counter* that is pleasant and easy to use. The appearances of both the website and mobile applications have been designed with consistency in mind. A common CSS stylesheet will be used in the web application to ensure that all colors, buttons, text sizes, the appearance of tools such as the navigation bar, data tables, forms, and other layout details all use the same formats, fonts, sizes, and colours, as can be seen in figures 2, 3 and 4 in the appendix. Components of the mobile application have also been designed to match each other, using similar colours to the web application (see figures 7 and 8 in the appendix). The intention is to create an interface that is pleasant to look at, as a more polished website will improve a user's experience.

Every web page and activity in *My Word Counter* will be designed for maximum simplicity, while still performing their functions. The purpose of this is to avoid overloading the memory of its users by keeping displays simple and ensuring that all operations can be carried out in one window. The main interaction styles used are form fill-in and menu selection, which rely on recognition (as opposed to recall), thus reducing memory load. A navigation bar will be placed at the top of each web page to enable effortless navigation around the web app. All data tables contain a search bar (see figure 3 in appendix) that allows a user to search for a specific piece of information and can be sorted by column in ascending or descending order. Additionally, the mobile app's interface is orientated around its main task, i.e.

Creating and participating in a word counting session. The home screen (see figure 8 in appendix) avoids clutter as it only has four large buttons, the largest allowing a user to begin creating their session.

The interface for both applications has been devised so that everything is self-explanatory, even to a novice user. Buttons will clearly state their functions, and descriptions and instructions will be provided where necessary. Standard gestures and controls will be used as expected, for example, a back arrow in the mobile application allows a user to navigate back to the previous activity. Forms will contain hints as placeholders to help the user understand the expected input (see figures 4 and 7 in the appendix). Measures will be taken to prevent users from making mistakes where possible. This includes adding form validation to the sign-up and edit account forms which will inform a user if they have made an error in their input, and checking that a user is certain (by asking them to enter their password) before they perform sensitive actions such as deleting their account or changing their password. In addition, informative feedback will be provided once a user completes an action, for example in the case of a successful (or unsuccessful) attempt to update account details.

Shortcuts will be included for frequent users. Examples of these include the ability to navigate quickly from a session completed by a user's client to that client's page on the website, and the ability for mobile app users to quickly start a session based on a session template (i.e. a previously recorded session that the user chose to save as a template). It is also important to offer reversible actions where possible. This relieves anxiety as the user knows that errors can be undone. Mobile app activities contain 'back' arrows for backward navigation and user operations will include 'cancel' buttons where possible. Additionally, the user account data, once edited, can be edited back to its original data if the user desires.

Consideration has been given to the context in which the mobile app is used, and it has been ensured that a word counting session will continue to run in the background if a user locks their phone or opens a different application. Thus, the session does not require the screen to be on during a session. These features are important; keeping the screen on would drain the device's battery.

3.8. Accessibility

The interfaces for *My Word Counter* seek to be as accessible as possible to all users by conforming to website content accessibility guidelines.

No part of the website relies on anything other than text-based content (i.e. images and colors are purely aesthetic). All images used in both applications contain some alternative or descriptive text to aid accessibility services in determining their meaning. Colour alone is never used as the only visual means of conveying information, and care has been taken to ensure that where colour is used for text or as a background to the text, the contrast is high enough for the text to be readable to users who are colour-blind (see figures 5 and 6 in the appendix) Additionally, a large font size is used on all activity screens and web pages to increase the readability of text. Animations known to prompt seizures (i.e. repeated flashing) are not used.

The interaction styles used in the *My Word Counter* web application revolve around button selection and form filling, and are therefore all operable through a keyboard. The language of the document is specified in the HTML tag at the start of each web page so that automatic content-retrieval systems can locate documents in the required language, and so that applications such as speech-synthesizers will run more easily on the web page. Any form fields that require input contain a hint in the form of a placeholder (See figures 4 and 7 in the appendix). As mentioned above, form validation notifications inform the user if they have made any mistakes in their inputs. To maximize the website's compatibility with current and future agents (e.g. assistive tools) all elements have complete start and end tags, are nested correctly, and do not contain duplicate attributes.

3.9. Security

In addition to the security rules implemented via Firebase, measures will be taken in both the mobile application and the online application to ensure security. Every web page that appears after a user has logged in will check that the user is properly authenticated before showing any data. If the user is not properly authenticated, they will be returned to the login page. This will prevent a potential hacker from bypassing the login page. Similarly, users can be logged out from any activity in the mobile application if they are not properly authenticated.

Passwords will be required to be at least six characters. Longer passwords are more secure, as it is more time-consuming to crack a longer password.

4. Implementation

4.1. Database

Creating the database for *My Word Counter* was a straightforward process, the more complex aspects of the implementation being handled by Firebase. The only requirement for setting up a Firebase project was ownership of a Google email account. Firebase's documentation was easy to follow and included instructions on how to integrate one of Firebase's databases or Firebase's authentication services into a web application (using JavaScript) or a mobile application (using Kotlin or Java as preferred). The authentication types allowed in the project could be specified in the Firebase console, and it was possible to create email templates for emails sent to authenticated users. Examples of these email templates include an email to verify an authenticated user's email and the email sent to a user when they request to reset a forgotten password.

As previously explained in the design section (section 3) of this report, the database type used for this project was Cloud Firestore, and it consisted of collections of JSON-like documents. The structure of the database can be seen in figure 1 of the appendix. At the start of the project, a JavaScript program was used to generate mock data to populate the database. This would aid in testing the implementations of functions that retrieve and display data from the database during the early stages of the development of the mobile and web applications. If a change in requirements results in a change in database structure, it is a simple matter to modify the script and repopulate the database.

Access rules to support confidentiality, as described in section 3 of this report, were defined in the *Rules* section of the Cloud Firestore database. Again, Firebase provided ample documentation to guide this process. Below is an example of an access rule written to allow any user to access the *public_ID* collection:

```
// allow anyone to access the public IDs
match /public_ID/{public_ID}{
  allow read, write: if true
}
```

4.2. Web Application

It was determined that the best course of action would be to work on the web application for *My Word Counter* before starting the mobile application so that there would be an easy interface to use when ascertaining whether a user's session had been uploaded correctly during the development of the mobile application. Firebase handles the back end of this project, so it was only necessary to develop the client-side of the web application. Due to evolving requirements, there were several things created for the web application that did not see inclusion in the final application such as support for admin access and the ability for therapists to create word-counting sessions and assign them to their clients.

4.2.1. Data Structures

Data retrieved from *My Word Counter's* database is stored in classes using a Firebase function and can then be used by other functions involved in displaying that data on a web page. Three classes are used: Client, clientSession, and CurrentUser. The structure of these classes is observable in figure 12 in the appendix. The client and session classes have a method `toTableData()` which converts each client or session into an array that can be used to represent a row in a DataTable.

4.2.2. Display of Client and Word Exposure Data: Tables

The main function of the website application is to provide a portal through which therapists and researchers can gain access to word exposure data, and in the case of therapists, data about their clients (see figure 3 in the appendix). As previously mentioned in section 3, the DataTables JQuery plug-in was used to represent this data. The tables contained many useful features such as the ability to search the table, a pagination system, and the ability to sort the tables by a specific column. Employing pre-existing libraries (as opposed to creating JavaScript or HTML tables from scratch) is extremely beneficial because it leaves more time available for working on other parts of the implementation.

Visible below is an example of the JavaScript code needed to construct a DataTable, in this case, a table displaying the clients of a therapist. "clientDataSet" refers to a list of objects, each representing a row of client data using that client's `toTableData()` method as explained in section 4.2.1.

```
var table = $('#clientTable').DataTable( {
    data: clientDataSet,
    columns: [{ title: "First Name" }, { title: "Surname" },
        { title: "DOB" }, { title: "ID" }, {title: ""}]
});
```

4.2.3. Forms

Another vital feature of the web application for *My Word Counter* is to allow users to log in, set up accounts, and edit their account data. This is facilitated by forms, which have been designed to be as user-friendly as possible. An example of a form can be seen in figure 4 in the appendix. The inputs into these forms are retrieved once submitted and used with Firebase functions to authenticate a user or add/update user data. It was important to implement forms that were not only functional but intuitive and aesthetically pleasing to the user.

Client-side form validation was an integral part of these forms. Tasks completed by form validation included ensuring that required form fields were not empty, passwords were of the required length, and that emails used a valid email format. Regex was employed for some of these tasks due to its efficiency, including for the validation of emails seen in the example below.

```
function validateEmail(mail){ // validates if 'mail' is an email
    if (/^[a-zA-Z0-9.!#$%&'*/+=?^_`{|}~-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)*$/ .test(mail)) {
        return (true)}
    alert("You have entered an invalid email address!")
    return (false)
}
```

Although they were not included in the final implementation of the project, forms used to create word counting sessions deserve special mention here as they were perhaps one of the most challenging parts of the website implementation. The original plan was to allow a different required word count for each individual word that the user was looking for (later changed to simply use the same required word count for all words in a session). To facilitate different word counts for each user-inputted word, each word inputted into the 'words' form fields would appear in the 'required words count' section, which consisted of a table of words and corresponding required count form fields. (see figure 9 in the appendix). This was achieved by creating a loop that would call a function designed to solve this problem.

```
setInterval(updateWordCount, 1000);
```

This function was responsible for checking the current words inputted into the 'words' field, and updating the required word count section accordingly. It was possible to use JavaScript to create a table row containing the words and their accompanying input field. The full implementation of this function can be seen in figure 13 in the appendix.

Another slightly less challenging, but still significant, part of this form was the option to assign the session to a client. This required creating a dropdown menu for all the clients who had that therapist as their therapist (see figure 10 in the appendix). This was implemented by retrieving all possible clients and adding their names and IDs to a dropdown select element. The implementation of this function has also been included in the appendix as figure 14. A custom drop-down selection box was also created for specifying the time limit allowed for a session. The ability to create sessions was eventually moved from the web application to the mobile application to give parents and guardians more control over sessions being created and reduce the burden placed on a therapist.

4.2.4. Additional Operations: Creating and Deleting User Accounts

User account operations in *My Word Counter* presented some challenges, although Firebase's database and authentication functions did simplify affairs.

The creation of a new user account requires the generation of a random, unique public ID. The *public_ID* collection on Cloud Firestore is then searched. If the newly generated public ID is not already taken by another user, it can be used. Otherwise, a new public ID will be generated.

```
char_options = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";  
  
for (i=0;i<20;i++){  
  
    id = id + char_options.charAt(getRndInteger(0, (char_options.length-1)))  
  
}
```

When deleting user accounts, it is important to ensure that any related data was also modified or removed. In the case of therapists, this involves unsetting the *assigned_therapist* field in any client belonging to the therapist being deleted. Additionally, when performing sensitive operations such as the deletion of a user account or the modification of the user account's authentication information, the user is required to reauthenticate themselves, as discussed in section 3.9. This was achieved using a simple alert containing a form field.

4.2.5. Moving to the online server

The web application for My Word Counter was developed on a laptop and tested locally by running it in a browser from its local file location. The website had to be moved to the student's folder on UCC's CS server to make it publically available on the internet. This was achieved through a combination of Secure Shell Protocol (SSH), which facilitated remote access to the CS server from the local machine, and Git, which was used to pull the web application files onto the server. Some errors were encountered when the website was first moved to the server, for example, the links for images stopped working. Fortunately, these problems were simple to fix and the use of Git made it very easy to update the website application on the server whenever new changes were made.

4.3. Mobile Application

The mobile application for My Word Counter was mainly designed around its chief function of facilitating the creation of and participation in word counting sessions. During this project, the student learned much about Kotlin development and Machine Learning. Although some of this information was gained through modules undertaken this year, the student also had to learn many new things by themselves.

4.3.1. Data structures: Classes

Data retrieved from Firestore in the mobile application is stored in classes, similar to the web application. There are four classes for representing data in the mobile app, visible in figure 11 of the appendix. The current user's data is stored in *UserClass*. Session templates are stored in *TemplateClass* and completed sessions are stored in *CompletedSessionClass*. A more generic *SessionClass* exists for operations that can be applied to both types of sessions. A *SessionClass* can be converted into a *TemplateClass* or a *CompletedSessionClass*, and vice versa. An additional function of these classes is that they facilitate the easy passing of data between activities in an intent. This is implemented by making a data class *Parcelable* by extending the *Parcelable* interface in the class.

4.3.1.1. User Interface: Displaying Previous Sessions and Session Templates

Sessions (of both types) are displayed using a RecyclerView i.e. a scrollable list of items, which opens up a new activity when pressed. A RecyclerView is implemented with the help of an adaptor, which takes an ArrayList of *SessionClass* objects, creates an instance of a row, and populates it with an item from the ArrayList. The layout of each row in the RecyclerView was specified using an XML layout file. When a session in the RecyclerView is selected, a new activity opens and shows more detail about the selected session. This is achieved by passing the selected object to the activity via an intent.

```
var adapterObj = MyAdaptor(sessionList)

adapterObj.setOnItemClickListener(object: MyAdaptor.ClickListener{
    override fun onItemClick(v: View, position: Int) {
        val intent = Intent(this@SessionActivity, SessionView::class.java)
        intent.putExtra("session", sessionList[position])
        startActivity(intent)
    }
})
```

Previously completed sessions use a table to show the word count recorded for each word in the session (and the required word count, if specified). This is implemented using *TableLayout* and *TableRow* components. Session templates also use a table if

the session is to be ended by a required word count. A ScrollView component has been used to allow the activity to be scrolled when all of the information displayed does not fully fit on the screen. A user can delete a session at any time, supported by a function from Firestore. Additionally, a session can be started from a session template by sending the session template's information to the Create Session Activity using an intent.

It is possible to allow sessions to be accessible when the device is offline by utilizing a useful Cloud Firestore setting, `isPersistenceEnabled`.

```
val settings = firestoreSettings { isPersistenceEnabled = true }  
  
db.firestoreSettings = settings
```

4.3.2. User Interface: Account Data

Forms (in Android, a group of EditTexts) are used to allow users to log in, sign up, and edit their account data, an example of which can be seen in figure 7 in the appendix. Once a form is submitted, the inputted values are retrieved and used to update the Firestore using Firebase's functions. A form validation class was created to aid in the validation of each form input, in keeping with the requirements specified in section 2, to improve the user experience as discussed in section 3. A separate method has been created for the validation of each form in the mobile application. Efforts were made to maximise reusability, for example methods such as `checkValidEmail()` are called in both the `ValidateEditForm()` and the `ValidateSignUpForm()` methods. The main advantages of code reuse include saving time and reducing redundant code. Additionally, if code is reused in this fashion by avoiding copy and paste, errors and bugs are far easier to fix as they do not need to be fixed in multiple places. As in the web application, regex has been used frequently due to its efficiency, to validate form inputs, such as in `checkValidEmail()` below.

```
fun checkValidEmail(input: String): Boolean {  
  
    val EMAIL_ADDRESS_PATTERN = Pattern.compile(  
        "[a-zA-Z0-9\\+\\.\\_\\%\\-\\+]{1,256}" + "\\@" +  
        "[a-zA-Z0-9][a-zA-Z0-9\\-]{0,64}" + "(" + "\\." +  
        "[a-zA-Z0-9][a-zA-Z0-9\\-]{0,25}" + ")+"
```

```
)  
  
return EMAIL_ADDRESS_PATTERN.matcher(input).matches()  
  
}
```

Users are notified of any mistakes in their inputs via a simple Toast message on the screen. For sensitive operations such as editing a password, the user is asked to enter their login details again. This is implemented using a pop-up dialog, implemented using a Dialog class and a custom layout file, containing a small login form.

A parent or guardian can decide to set their child's therapist from the Account Data Activity. Connecting a client with their therapist is a three-step process and relies on the *temp_codes* database to store one-use randomly generated codes that expire if they have not been used in over 24 hours. A user is shown their code in a pop-up dialog if they chose to add a therapist. If a therapist chooses to add the client using their code, the client will receive a pop-up dialog asking them to accept or decline the request the next time they open the home activity.

4.3.3. Recording a session

The main function of *My Word Counter* is to facilitate word counting sessions.

4.3.3.1. User Interface

Forms, similar to those described in section 4.3.3, were used to allow a user to create a session either from scratch or using a session template as mentioned in section 4.3.2. A chronometer is used to show the time elapsed since the start of the session. The advantage of using a chronometer for this purpose is that it is relatively easy to start, stop, and pause the timer. Additionally, the timer contains a useful method that is called on each tick of the chronometer and can be used to update the UI. A toggle button is used to allow the user to pause or resume a session at any time. A table, similar to those in section 4.3.3, is used to display the current word count and is updated on each tick of the chronometer.

4.3.3.2. Service

My Word Counter requires a session to have the ability to run until completion, even if the user has opened a different application or put their phone into sleep mode. This was achieved by using a session, which would run in the foreground, using persistent notifications to inform the user as to the status of the session (i.e. time elapsed, whether the session was paused or completed). This was quite challenging, especially because the student has never developed an Android service before. Fortunately, it was possible to learn most of the necessities through a combination of online tutorials and explanations given on online forums such as Stack Overflow.

One of the most difficult parts of creating a service was binding the service to the user interface so that word counts could be displayed and so that the service would know if it was being paused or resumed. This was implemented by creating an inner class in the Service that extended `Binder()`

```
inner class RecordSessionBinder : Binder() {  
    // Allow clients to call public methods  
    fun getService(): RecordSessionService = this@RecordSessionService  
}  
  
override fun onBind(intent: Intent): IBinder {  
    return binder  
}
```

On the user interface side, the activity needs to connect to the service. Once the activity has connected, it can give the service the details of the session that is to be run, and start the chronometer. This is implemented using a `ServiceConnection` object, which connects the activity to the service and allows the activity to bind the service.

```
private val connection = object : ServiceConnection {  
    override fun onServiceConnected(className: ComponentName, service: IBinder) {  
        val binder = service as RecordSessionService.RecordSessionBinder  
        mService = binder.getService()  
    }  
    override fun onServiceDisconnected(arg0: ComponentName) {  
        mBound = false  
    }  
}
```

This `ServiceConnection` object was then called in an intent once the activity was created and used to start the service. The activity was then able to call functions from the service (e.g. when updating the word counts or pausing the session).

While getting information from the service to the activity just required binding, getting information from the activity to the service was trickier. A method was created in the service to set up the session in the service using the session object originally sent to the activity in the intent. This method is called by the activity once it has connected to the service. It was also necessary to create a timer that would run alongside the chronometer (which is a UI component) so that the service would know how much time had elapsed when the activity was closed. This was achieved by using a `Timer` to call an update loop method every second, replicating the `onChronometerTick()` method. This update loop is responsible for updating the timer on the persistent notification and checking to see if the time limit has been reached.

```
this.loopTimer = Timer()
this.loopTimer.scheduleAtFixedRate(object : TimerTask() {
    override fun run() {
        updateLoop()
    }
}, 0, 1000)
```

4.3.3.3. Notifications

Another challenging aspect of the *My Word Counter* mobile application was learning how to implement notifications (both persistent and nonpersistent), determining whether the notifications would make a noise or vibrate, and adding a mechanism to allow the notifications to resume a closed activity if pressed. As with services, knowledge of how to implement notifications was gained through various online tutorials and developer forums.

Creating notifications on an Android device requires the following: *NotificationManager*, *NotificationChannel*, and a notification *Builder*. The *NotificationManager* manages all notifications and can be used to create a notification channel.

```
notificationManager =
    getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
```

All notifications on a device must be assigned to a *NotificationChannel* from Android 8 onwards.

```
notificationChannel = NotificationChannel(channelId, description,  
                                         NotificationManager.IMPORTANCE_HIGH)  
notificationManager.createNotificationChannel(notificationChannel)
```

The notification *Builder* is then used to create a notification and set attributes such as the title of the notification.

```
currentBuilder = Notification.Builder(this, channelId).  
    .setContentTitle("MY WORD COUNTER ")  
    .setContentText("Session Service Running")
```

Notifications have different configurations for different android versions, for example, notification channels are only required from Android 8 onwards. This is catered for by implementing a different method for each version.

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O){...}
```

This notification is for a foreground service, which means that the service will inform the user what is happening during the session via notifications.

```
startForeground(123, currentBuilder.build())
```

As stated in the requirements in section 2, an audio and vibrate alert is needed when a session is finished, however, the user will not want to be overburdened by noisy notifications, so the majority of updates to the notifications (e.g. for incrementing the elapsed time or for when session is paused) will simply be updated to a persistent notification. This is achieved by setting the attribute `setOnlyAlertOnce` to true when building the notification. To allow the service to notify a user when the session was completed, `setOnlyAlertOnce` is set back to false and the `Vibrator` and `RingtoneManager` classes are used to alert the user.

```
val vibrator = baseContext.getSystemService(Context.VIBRATOR_SERVICE) as Vibrator
```

```

if (Build.VERSION.SDK_INT >= 26) {

    vibrator.vibrate(VibrationEffect.createOneShot(200,

        VibrationEffect.DEFAULT_AMPLITUDE))

} else { vibrator.vibrate(200)}

val uri = RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION)

val ringtone = RingtoneManager.getRingtone(this, uri)

ringtone.play()

```

The notifications have the additional function of allowing a user to resume the session activity after the activity has been closed. This is achieved by including a pending intent for the activity in the notification. This intent's extras are updated on each call of the update loop and include the details of the current session, the current elapsed time, and a variable that states that this session has already been started. When a session activity is started, it checks to see if this session is just starting, or if it is being resumed. The intention is to let the user believe that they are opening the same activity that they saw earlier.

4.3.4. Integrating Speech Recognition

The final major obstacle to overcome was the inclusion of Speech Recognition, the actual counting of words spoken, in *My Word Counter's* web application. This was particularly difficult because although *PocketSphinx's* library contained methods that supported continuous listening for specified keywords on an Android device, all of the existing documentation and demos were for Java and had to be understood well enough to be converted into Kotlin.

Before a session can use Speech Recognition, it needs to obtain permission to record audio from the owner of the device. This was achieved by checking if such permissions were granted before starting a session, and asking the user for permissions if not. If permissions are denied, the user is redirected to the home activity.

```

val permissionCheck: Int = ContextCompat.checkSelfPermission(

    applicationContext, Manifest.permission.RECORD_AUDIO)

if (permissionCheck != PackageManager.PERMISSION_GRANTED) {

    ActivityCompat.requestPermissions(this,

```

```

        arrayOf(Manifest.permission.RECORD_AUDIO), RECORD_REQUEST_CODE)
    } else { ifPermissionsGranted() }

```

To allow the session to continue counting words while the app itself was closed, Speech Recognition needed to be implemented in the session service. This was accomplished by extending the class *RecognitionListener* (from *PocketSphinx*) in the service. Initializing the recognizer is time-consuming, so it is initialized in an asynchronous task, which informs the recognizer to start listening once the setup has been completed.

```

SetupTask(this).execute()

```

The recognizer requires the acoustic model, dictionary, and keyword threshold to be specified on set up, as can be seen in the code below. For this project, due to time constraints, it has been decided to use the defaults that were provided in the demo.

```

@Throws(IOException::class)
private fun setupRecognizer(assetsDir: File?) {
    recognizer = SpeechRecognizerSetup.defaultSetup()
        .setAcousticModel(File(assetsDir, "en-us-ptm"))
        .setDictionary(File(assetsDir, "cmudict-en-us.dict"))
        .setKeywordThreshold(1e-20f)
        .recognizer
    recognizer?.addListener(this)
    recognizer?.addKeywordSearch(DIGITS_SEARCH, file)
}

```

PocketSphinx supports several types of search, but for this project the search required was a keyword search, using words specified in a grammar file called "my_words.gram". This file is created dynamically during the setup of a session service, using the specified words for that session. Each specified word is given its own line in the file followed by a threshold for that word, for example for the word 'cat', the line would read " cat /1e-1/" The file is deleted at the end of each session to save storage space on the device.

PocketSphinx's *RecognitionListener* includes a method called *onPartialResult()* which is called by the recognizer every time a keyword is recognized. This method can be overwritten, and it is possible within that method to retrieve a string to which a word has been written every time it is recognized. The current word count for each word can be calculated by using a method (*Collections.frequency()*) to count the number of occurrences of that word in the string. These counts are stored in an array, which is read by the activity to update the word counts on the screen. If the recorded counts

for each word exceed the required word count, the session will be stopped immediately. At the end of a session, all word exposure data recorded will be stored in the session object and passed to the next activity, which allows a user to add in any additional notes via a pop-up form dialog, and then uses Firebase's functions to save the data to the database on Cloud Firestore when the device next connects to the internet.

4.3.5. Additional Operations: Creating and Deleting User Accounts

User account operations in the mobile application for *My Word Counter* faced similar obstacles to those seen above in the web application. User public IDs were generated randomly and then checked against the public ID collection to ensure that they were unique. It is also necessary to ensure a client's sessions are deleted if they decide to delete their account. This application functions offline, however certain actions related to user authentication require access to firebase. To prevent errors, the application does a quick check to make sure that the device is connected to the internet (using the `ConnectivityManager` and `NetworkCapabilities` classes), warning the user if it is not.

4.3.6. Deployment:

Deploying the mobile application for *My Word Counter* to the website as an Android Package (APK) presented its own challenges. A separate Git branch, *deploy*, was used when modifying the application so that it could be converted properly to an APK. First of all, steps had to be taken to prepare the application for release, such as removing debugging permissions from the Android Manifest file. Next, the application was signed using a private key store. Android Studio provided tools that helped with this process. New errors in the application were encountered when generating the APK that had not previously caused bugs, such as references to non-existent IDs in the layout files. Once successfully generated, it was easy to add the APK to a page on the website, thus publishing it to the public.

5. Evaluation

5.1. Testing Process

Testing is a critical part of any software development process, and *My Word Counter* is no different. Many types of testing can be employed in a project. It should be noted that Unit Testing was not used for this project, as the code implementation is mainly focused on the front-end of the application. Instead, the testing was entirely manual and mostly took place through interaction with the user interface. Both black box and white box testing techniques were utilized during the development of the system.

To guarantee that no bugs existed and to ensure that everything worked as intended, integration tests were carried out on affected components every time a new feature was added to the system. Additionally, a full system test was executed every time the system was updated on the server. The majority of the development for each application was carried out on the same git branch, so every addition to the existing application had the potential to impact other sections of the implementation. Fortunately, the system is small so these tests were not very time-consuming.

Each component of both applications was subjected to both positive and negative tests. The test cases used were based on the requirements specified in the analysis section (section 2). An example of test cases and their results for the log-in page in the web application can be seen here.

Requirement	Test Case	Expected Result
The user will be presented with a form where they can submit their email and password to log in. Sign-up button: Brings a user to the sign-up page, where they can make a new account to start using the website Forgotten password: A user can request an email to	Open the log-in page.	A form with two fields, email and password, and a submit button will be visible to the user. A sign-up option should be available to the user A 'forgotten password' option should be visible to the user

allow them to reset their password		
The user will be presented with a form where they can submit their email and password to log in	Open the login page. Enter a valid email and password.	The user will be logged in
The user will be presented with a form where they can submit their email and password to log in	Open the login page. Enter a valid email and invalid password.	The user will not be logged in and will be asked to recheck their email and password
The user will be presented with a form where they can submit their email and password to log in	Open the login page. Enter an invalid email and valid password.	The user will not be logged in and will be asked to recheck their email and password
Sign-up button: Brings a user to the sign-up page, where they can make a new account to start using the website	Open the login page. Press the 'sign up' option.	The user should be brought to the sign-up page
Forgotten password: A user can request an email to allow them to reset their password	Open the login page. Press the 'forgotten password' option.	The user should be brought to a page that allows them to reset their password

For features such as form validation or testing for expiry dates of temporary codes, it is necessary to carry out extensive positive and negative testing on all edge cases. There are many areas where an error may occur during validation, and it is important to cover every possible path in the code. An example of this white-box testing can be seen in the test cases below. Testing different dates and times to check if a code had expired requires making modifications to the date and time fields in a generated temporary code saved to the *temp_codes* collection in the database. It is also necessary to check that an expired code has been deleted from the database.

Requirement	Test Case	Expected Result
This code (the temporary code for connecting client to their therapist) expires after 24hrs	The date that the code was generated (i.e. the date field in the document) is today	The code is accepted and a request is sent to the client
This code (the temporary code for connecting client to their therapist) expires after 24hrs	The code was generated 23 hours and 58 minutes ago	The code is accepted and a request is sent to the client
This code (the temporary	The code was	The code is not accepted:

code for connecting client to their therapist) expires after 24hrs	generated 24 hours and 1 minute ago	the user is informed that the code has expired, and the code is removed from the database
This code (the temporary code for connecting client to their therapist) expires after 24hrs	The code was generated 24 hours and 1 hour ago	The code is not accepted: the user is informed that the code has expired, and the code is removed from the database
This code (the temporary code for connecting client to their therapist) expires after 24hrs	The code was generated the day before yesterday	The code is not accepted: the user is informed that the code has expired, and the code is removed from the database
This code (the temporary code for connecting client to their therapist) expires after 24hrs	The code was generated on this day and month, but one year ago	The code is not accepted: the user is informed that the code has expired, and the code is removed from the database
This code (the temporary code for connecting client to their therapist) expires after 24hrs	The code was generated on this day and year, but one month ago	The code is not accepted: the user is informed that the code has expired, and the code is removed from the database

Some test cases require logging information to the console and reading it using debugging tools to maximize code coverage. An example of a test case that requires this can be seen below, which also tests the system used to check for expiry dates of temporary codes. This particular set of test cases cover the lines of code that determine if the 'date' specified for the code is the day before today. The code was modified so that it believed the date was 01/01/2020. The expected 'yesterday' date is then logged to the console. This test case is carried out for the last day of each month, to ensure that a valid code will not be declared 'expired'.

Requirement	Test Case	Expected Result
This code (the temporary code for connecting client to their therapist) expires after 24hrs	The function retrieving the current date is modified to return '01/01/2020' as the date	The date '31/31/2019' will be logged to the console

Due to the Covid-19 pandemic, it was not possible to test *My Word Counter* fully in its intended context i.e. with therapists or the parents/guardian of children with DLD.

Instead, the system was shared with friends and family members of the student, a group that spans ages and technological ability. These users did not know anything about the inner implementation of the system, so the testing was entirely black box. Allowing users who were unfamiliar with the system to attempt to use it was a very useful testing technique and help to highlight issues with both the basic functionality of the applications and the type of user experience they generated.

Testing of the mobile application for *My Word Counter* was carried out on a single physical device (a Samsung A40 phone) during most of the development process. The motivation for this was because the laptop used for the development of this system was incapable of running an emulator within a reasonable time frame, and frequently, without crashing completely. Towards the end of the project, the application was tested on devices owned by family members and friends to ensure compatibility across a wider range of devices. Fortunately, despite being limited to a single device for most of the testing process, the application performed well on other devices when tested.

Detailed regression tests were not designed for this project, due to the small size of the system and existing time constraints. Nonetheless, the entire system was fully tested after the final release to ensure that every page, activity, and feature complied with the requirements specified in the analysis section (section 2).

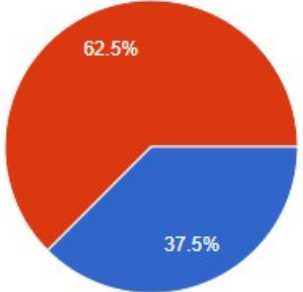
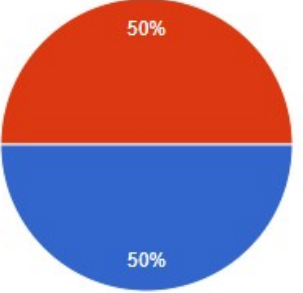
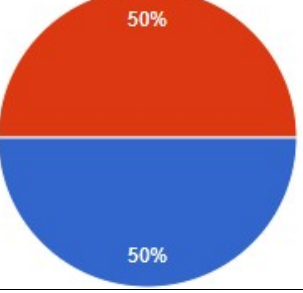
5.2. Evaluation of Final System

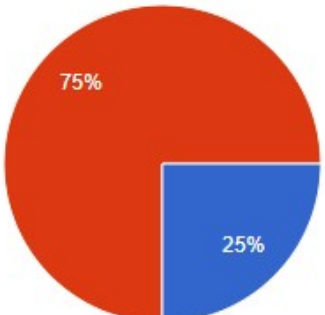
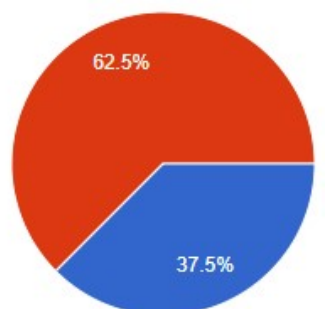
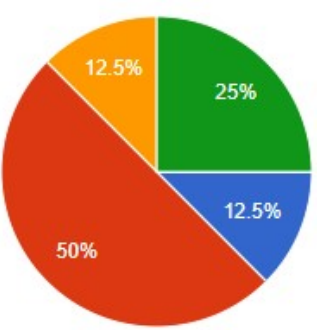
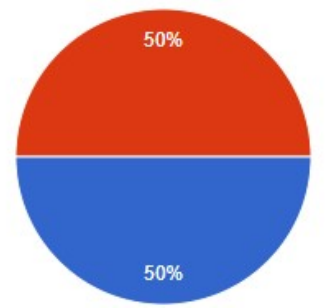
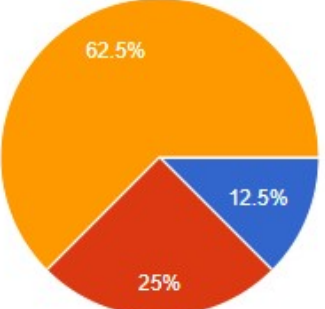
My Word Counter has been tested rigorously and it has been proven that the main objectives of this project (see section 2.2) have been met completely by the system designed. Additionally, user experience information was gathered through an anonymous Google Forms survey and was mainly positive.

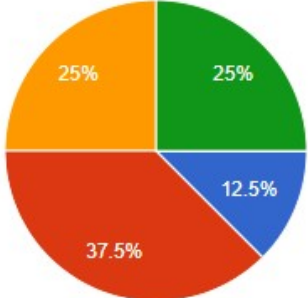
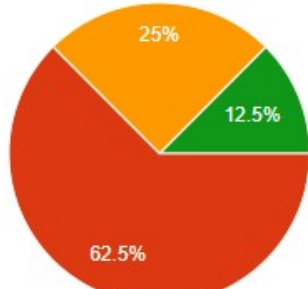
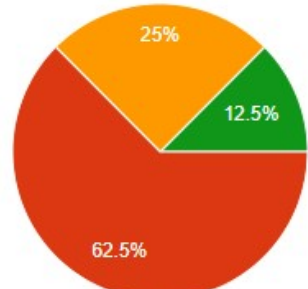
5.2.1. User Experience Survey Results

An anonymous user-experience survey was carried out towards the end of this project. A group of eight users, most of whom had never seen the application before, from a group ranging in age and technical ability, were asked to do the following: make accounts on the mobile and web applications, attempt to record a session using the mobile application, and then locate that session using the web application. They

were then asked questions about their experience. Instructions given on how to complete these tasks were kept intentionally vague, to determine how a user would navigate the app without assistance. Each question consisted of a statement, and the participant was asked to rate the statement by how much they agreed/disagreed with it (on a scale of strongly disagree to strongly agree). These statements were carefully worded to avoid introducing bias.

Survey Results							
Key: <ul style="list-style-type: none"> Strongly Disagree Disagree Neutral Agree Strongly Agree 							
Question	Response						
The website application was difficult to navigate	 <table border="1"> <thead> <tr> <th>Response</th> <th>Percentage</th> </tr> </thead> <tbody> <tr> <td>Strongly Disagree</td> <td>37.5%</td> </tr> <tr> <td>Disagree</td> <td>62.5%</td> </tr> </tbody> </table>	Response	Percentage	Strongly Disagree	37.5%	Disagree	62.5%
Response	Percentage						
Strongly Disagree	37.5%						
Disagree	62.5%						
The mobile application was difficult to navigate	 <table border="1"> <thead> <tr> <th>Response</th> <th>Percentage</th> </tr> </thead> <tbody> <tr> <td>Strongly Disagree</td> <td>50%</td> </tr> <tr> <td>Disagree</td> <td>50%</td> </tr> </tbody> </table>	Response	Percentage	Strongly Disagree	50%	Disagree	50%
Response	Percentage						
Strongly Disagree	50%						
Disagree	50%						
The text used in the mobile application interface was difficult to read	 <table border="1"> <thead> <tr> <th>Response</th> <th>Percentage</th> </tr> </thead> <tbody> <tr> <td>Strongly Disagree</td> <td>50%</td> </tr> <tr> <td>Disagree</td> <td>50%</td> </tr> </tbody> </table>	Response	Percentage	Strongly Disagree	50%	Disagree	50%
Response	Percentage						
Strongly Disagree	50%						
Disagree	50%						

<p>The sign-up process was confusing or difficult to complete in the web application</p>	 <table border="1"> <thead> <tr> <th>Category</th> <th>Percentage</th> </tr> </thead> <tbody> <tr> <td>Red</td> <td>75%</td> </tr> <tr> <td>Blue</td> <td>25%</td> </tr> </tbody> </table>	Category	Percentage	Red	75%	Blue	25%				
Category	Percentage										
Red	75%										
Blue	25%										
<p>The sign-up process was confusing or difficult to complete in the mobile application</p>	 <table border="1"> <thead> <tr> <th>Category</th> <th>Percentage</th> </tr> </thead> <tbody> <tr> <td>Red</td> <td>62.5%</td> </tr> <tr> <td>Blue</td> <td>37.5%</td> </tr> </tbody> </table>	Category	Percentage	Red	62.5%	Blue	37.5%				
Category	Percentage										
Red	62.5%										
Blue	37.5%										
<p>It was difficult to find a specific completed session in the database</p>	 <table border="1"> <thead> <tr> <th>Category</th> <th>Percentage</th> </tr> </thead> <tbody> <tr> <td>Red</td> <td>50%</td> </tr> <tr> <td>Green</td> <td>25%</td> </tr> <tr> <td>Yellow</td> <td>12.5%</td> </tr> <tr> <td>Blue</td> <td>12.5%</td> </tr> </tbody> </table>	Category	Percentage	Red	50%	Green	25%	Yellow	12.5%	Blue	12.5%
Category	Percentage										
Red	50%										
Green	25%										
Yellow	12.5%										
Blue	12.5%										
<p>It was difficult to create a word counting session</p>	 <table border="1"> <thead> <tr> <th>Category</th> <th>Percentage</th> </tr> </thead> <tbody> <tr> <td>Red</td> <td>50%</td> </tr> <tr> <td>Blue</td> <td>50%</td> </tr> </tbody> </table>	Category	Percentage	Red	50%	Blue	50%				
Category	Percentage										
Red	50%										
Blue	50%										
<p>The speech recognition used for a word counting session had low accuracy</p>	 <table border="1"> <thead> <tr> <th>Category</th> <th>Percentage</th> </tr> </thead> <tbody> <tr> <td>Orange</td> <td>62.5%</td> </tr> <tr> <td>Red</td> <td>25%</td> </tr> <tr> <td>Blue</td> <td>12.5%</td> </tr> </tbody> </table>	Category	Percentage	Orange	62.5%	Red	25%	Blue	12.5%		
Category	Percentage										
Orange	62.5%										
Red	25%										
Blue	12.5%										

<p>The speech recognition used for a word counting session is not high enough to be relied upon</p>	 <table border="1"> <caption>Speech Recognition Accuracy Data</caption> <thead> <tr> <th>Category</th> <th>Percentage</th> </tr> </thead> <tbody> <tr> <td>Very accurate</td> <td>37.5%</td> </tr> <tr> <td>It always recognised the word, however, it did fail some tests (mistook fox for box and similar)</td> <td>25%</td> </tr> <tr> <td>I have a rather unique speech pattern due to a unique accent and a slight stutter and it was able to pick up everything I said without hesitation. I was surprised at this</td> <td>25%</td> </tr> <tr> <td>Other</td> <td>12.5%</td> </tr> </tbody> </table>	Category	Percentage	Very accurate	37.5%	It always recognised the word, however, it did fail some tests (mistook fox for box and similar)	25%	I have a rather unique speech pattern due to a unique accent and a slight stutter and it was able to pick up everything I said without hesitation. I was surprised at this	25%	Other	12.5%
Category	Percentage										
Very accurate	37.5%										
It always recognised the word, however, it did fail some tests (mistook fox for box and similar)	25%										
I have a rather unique speech pattern due to a unique accent and a slight stutter and it was able to pick up everything I said without hesitation. I was surprised at this	25%										
Other	12.5%										
<p>Do you have any other comments on the accuracy of the speech recognition?</p> <p>(3 responses)</p>	<p>Very accurate It always recognised the word, however, it did fail some tests (mistook fox for box and similar) I have a rather unique speech pattern due to a unique accent and a slight stutter and it was able to pick up everything I said without hesitation. I was surprised at this</p>										
<p>It feels like the web application does not have a high level of security</p>	 <table border="1"> <caption>Web Application Security Data</caption> <thead> <tr> <th>Category</th> <th>Percentage</th> </tr> </thead> <tbody> <tr> <td>Does not have a high level of security</td> <td>62.5%</td> </tr> <tr> <td>Other</td> <td>25%</td> </tr> <tr> <td>Other</td> <td>12.5%</td> </tr> </tbody> </table>	Category	Percentage	Does not have a high level of security	62.5%	Other	25%	Other	12.5%		
Category	Percentage										
Does not have a high level of security	62.5%										
Other	25%										
Other	12.5%										
<p>It feels like the mobile application does not have a high level of security</p>	 <table border="1"> <caption>Mobile Application Security Data</caption> <thead> <tr> <th>Category</th> <th>Percentage</th> </tr> </thead> <tbody> <tr> <td>Does not have a high level of security</td> <td>62.5%</td> </tr> <tr> <td>Other</td> <td>25%</td> </tr> <tr> <td>Other</td> <td>12.5%</td> </tr> </tbody> </table>	Category	Percentage	Does not have a high level of security	62.5%	Other	25%	Other	12.5%		
Category	Percentage										
Does not have a high level of security	62.5%										
Other	25%										
Other	12.5%										
<p>Please add any other comments you have about My Word Counter</p> <p>(4 responses)</p>	<p>Could be very useful Well designed website and app, easy to manoeuvre It was difficult to exit the app, some back buttons didn't work (minor issues) It was easy for me to use and downloaded well</p>										

5.2.2. Evaluation of Original Objectives versus Final System Produced

Objective 1 requires an “application that will quantify the speed of learning by utilizing Speech Recognition technology to calculate the number of times specified words are used before it is evident that the child has achieved understanding”. This objective has been achieved by the system created for recording word exposure data using a mobile application and storing that data in a database, where it can then be accessed and analyzed by therapists.

Objective 1 also states the intention to develop an application that is “user friendly”. *My Word Counter* performed well on usability in the user experience survey, as can be seen above. All users appeared to find the system easy to navigate and agreed that the text used in both applications was very easy to read. Users also found the sign-up and session creation processes to be very intuitive. Surprisingly, some users found it difficult to find a specific session in the database without instruction. It may be necessary to include some user instructions, demo, or FAQ for the therapist interface if this system continues to be used.

Objective 2 is concerned with allowing a mobile application user to “maintain the focus of sessions on interaction with the child while simultaneously gathering word exposure information”. This is facilitated in the mobile application by ensuring that once a session has started, it will continue to collect word exposure data until it completes the stopping criteria for the session or is stopped manually by the user. The user can start a session, and as long as the device is powered on and is near the child, the user can focus on interacting with their child while the application counts words.

Objective 3 requires the “remote collection and storage” of word counting data from “home-based sessions”. By giving parents or guardians the power to create and run their own word counting sessions, they can create sessions wherever they please, including in their homes. As mentioned above, data recorded is then saved to the Cloud Firestore database, where it can be accessed by therapists and researchers from any location.

Objective 4 relates to the development of an “online web application” which must provide therapists with access to word exposure information “such that they can determine the individual needs of their clients”. This is facilitated through the system’s web application, which allows therapists to view information about their clients and every detail about each session completed by their clients. Therapists can add clients with the consent of that client’s parent/guardian.

Objective 5 also involves the web application, and requires that “anonymous word exposure data” is provided to therapists and researchers such that they can “access anonymous word exposure data” allowing them to investigate and analyze it “to discover the optimal number of word exposures needed to achieve understanding in

patients of different ages and clinical presentation". This is achieved through the *All Sessions* section of the online application, which allows users that do not have clients to access all word exposure data, but without gaining access to the personal details of the client who completed the session. This data is displayed in tables, which have features that allow the data to be searched and sorted.

Objective 6 aims to provide mechanisms such that a parent/guardian is granted *"control over the personal data"* of their child. A mobile user can view and modify their account data, or delete their account. They can also view and delete previously recorded word exposure data. Most participants in the user experience survey (62.5%) felt confident about the security of their personal data across both platforms.

5.3. Remaining Issues

The system for *My Word Counter* has successfully been designed and developed to achieve its objectives. Despite this, time constraints mean that some non-critical issues remain. In the website application, the ordering of some DataTables columns does not always function according to proper logic. The column will be ordered lexicographically, however that is not always the correct ordering system for that column. An example of this is the date which is written as dd/mm/yyyy and is sorted by the day before it is sorted by year. Another example of this is the ordering of columns such as the 'word count' column, which does not strictly fit into lexicographic order. If more time were available, it would be advantageous to a therapist or researcher's user experience to improve this ordering system.

Some participants in the survey (above) expressed a lack of trust in the security of the system. Ideally, more constraints would be placed on user passwords to increase security, and to reassure the users, for example requiring an uppercase letter and a number to be present in each password. As passwords can be reset using Firebase's password-reset template, which does not apply these constraints, these measures were considered somewhat redundant, as well as creating an inconsistency between different interfaces. If time had allowed, it would have been better to develop a custom password-reset page for the system.

There are a few outstanding issues with the mobile application. One user reported issues with backward navigation buttons in the application, as can be seen in the results above. Although these issues never presented themselves when tested on the developer's device, it is potentially a serious usability issue that would have been looked into further if time had allowed it. The 'words' field in the form that allows

users to create a session allows any input as long as it consists of letters. For usability purposes, it would be better if this field only accepted words in the Speech Recognizer's dictionary, as it may not recognize an unfamiliar word.

There have been mixed reviews as to the accuracy of the Speech Recognizer, mostly from users who are not well versed in artificial intelligence technologies. The fact is that it is currently impossible to achieve 100% accuracy. This has already been discussed in the design section of this report (section 3.6). Some users were very impressed with the performance of the application's speech recognition "Very accurate", however, comments from the user survey also include complaints, stating that the recognizer "mistook fox for box and similar". Unfortunately, errors such as these are to be expected, and the accuracy of this recognizer is still sufficient to provide researchers and therapists with the information that they need.

6. Conclusion

6.1. Summary of the Final Product

The final system for *My Word Counter* is a well-tested, robust system that is fully fit for its purpose. The result of this project is a fully functioning system that allows for the collection of word-exposure data during home-based sessions. This data subsequently collected into a database and made available such that therapists can evaluate their clients and researchers can analyze data and learn about the optimal word exposures and context necessary for a word to be learned by a child with Developmental Language Disorder. It is hoped that the information collected by this system might be of use in future research projects. The system has been designed and implemented from scratch this year. The evidence of this can be seen clearly in the preceding sections. It can be argued in hindsight that the project could have been conducted slightly differently to maximize the time available, however, the final system created was sufficient. Many things could be added to this system if more time was available, which may be implemented in the future. Additionally, much about application development (especially in Kotlin) and Speech Recognition was learned by the student this year. Overall, this project should be very useful and has the power to change the lives of the many children who suffer from DLD.

6.2. If more time was available

Unfortunately, the development of *My Word Counter* was limited by time constraints, and it was only possible to implement the most critical features needed for the application to function. Several features may have been included in the system if there was more time and resources available, and that perhaps might be implemented in the future. Some of these features, for example, a more intelligent ordering of data table columns in the website application and a custom reset-password page, have already been discussed in detail in the evaluation section (section 5.3).

In the design section (see section 3.2), the advantages and disadvantages of using different types of devices were mentioned. If the resources were available, it would be preferable to develop the app on more devices, for example adding support for iOS, FitBits, and smart watches. Smart watches would be particularly useful, as they could be worn by the child during a session. It would also be nice to make the application

available on the Google Play Store so that it would be more easily accessible to potential users.

It would be advantageous in the future to add more features that would improve the usability of both applications. For the web interface, more instructions, an FAQ/help page, or a demo for first-time users could be useful. In addition, it would be useful to develop more features that allow researchers to compare word exposure data, for example, options to view this data on a meaningful graph/chart based on certain criteria or to download the data as a spreadsheet. For the mobile application, forms could be made more intuitive, for example using a Date Picker Fragment to allow a parent/guardian to enter the date of birth of their child using a calendar during sign up. Additionally, the sessions initially allowed an individual required word count for each specified word. This was removed when session creation functionality was moved from the web application to the mobile application however it could be useful to add this feature back at a later date. Including some of these features would require extra time spent on learning more about Kotlin.

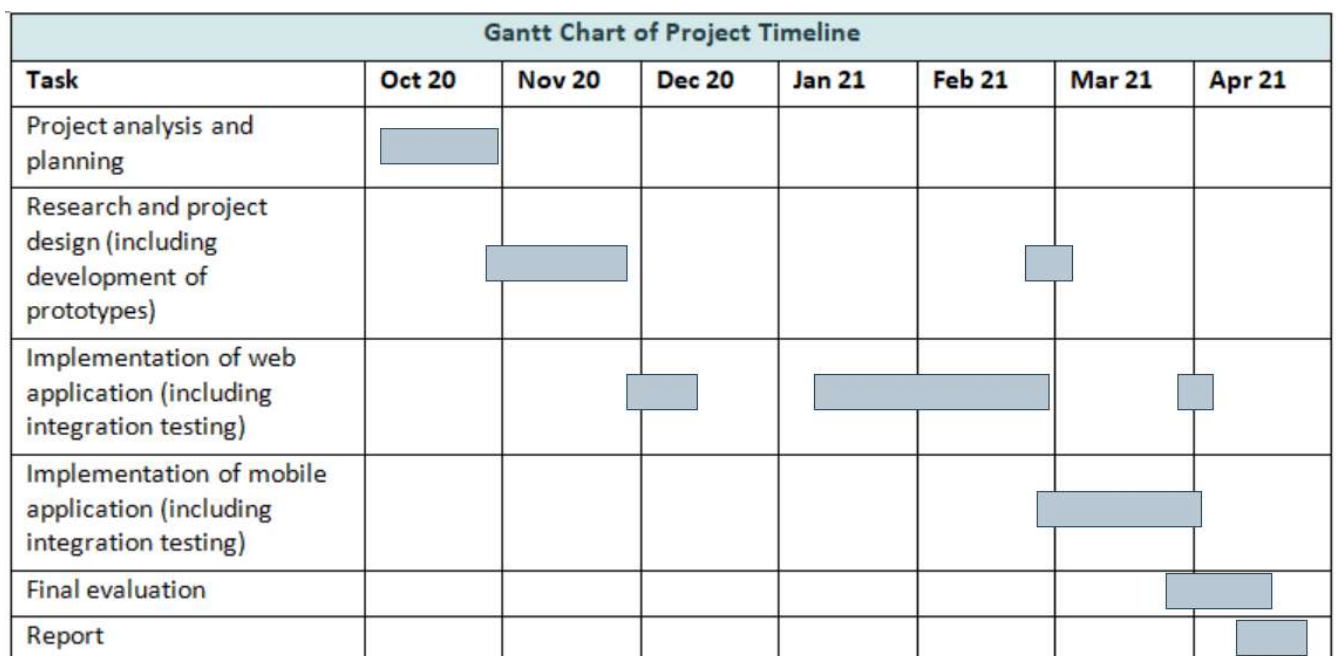
If time was available, it would be great to spend more time further improving the accuracy of the speech recognition model. This could perhaps be done by experimenting with acoustic models, dictionaries, and keyword thresholds. For example, an attempt could be made to find a dictionary better suited to an Irish accent or to suit the voice of a child. Mechanisms could also be implemented to support this across a wider range of countries by using a different acoustic model or dictionary to match the location of the child. It might also be useful to consider creating a new speech recognition model specifically for *My Word Counter*, using children's voices to train the model. This would take a lot of time, however, and issues such as the ethics of using child data such as children's voices for the project would have to be examined.

6.3. Evaluation of development process:

The software process used for this project was not strictly defined, however it followed the same concepts used in most system development. An analysis of the project and its objectives was performed at the start of the project. The system was designed based on the requirements and recommended technologies. These requirements were then implemented, following the design, and tested after the addition of each new feature. Communication between the student and supervisor was frequent, and meetings were also held with the *My Word Counter's* project

owner, a member of the Department of Speech and Hearing Sciences at UCC, to ensure that the system being developed fit with their requirements. Requirements were changed significantly during the development process, even after parts of the original requirements had been implemented. It could be speculated that the time spent implementing these features could have been spent elsewhere had the official requirements document been designed earlier and approved by the project owner, but sometimes it is necessary to diverge from the original plan of a project to better fulfill its purpose.

Time is one of the most important resources in a software project with an established deadline. Clever time management was critical for this project. This Gantt chart clearly illustrates the time spent on each aspect of the project.



When planning out the amount of time spent on each phase of the project, time also had to be allocated for the student to study for exams, as can be seen in the lack of progress achieved during the Christmas exams. It was decided that the student would undertake fewer modules during the second semester of the year, to leave more time available for project work close to the submission date. It is also vital to make sure that time is set aside to allow for setbacks such as major bugs that need to be fixed, difficulties in getting the software to do exactly as needed, and any changes that need to be made as a result of changes that are made to the project requirements. As can be observed in the chart above, more time was originally allocated to the implementation of the web application. This was due to the web application originally

handling all session and user creation, via an admin interface. Fortunately, when the plan changed, there was enough time to complete the mobile application. Good planning had ensured that any other college work required to be completed by the student would not have to be worked on during the last few weeks before the project deadline. This meant that more time each day could be dedicated to finishing *My Word Counter* to a suitable standard

6.4. What was learned

Working on *My Word Counter* was a wonderful learning experience. The student already had good knowledge of web development technologies, but everything they now know about mobile application development and Kotlin, visible in the final version of the mobile application, was learned this year. This has already been mentioned in more detail in the Implementation section (see section 4.3). It was also necessary to learn about Machine Learning, with a focus on Speech Recognition technologies.

This project emphasized the value of using existing libraries and tools (such as those chosen in the design section of this report), allowing a developer to spend their time working on features that have to be created specifically for the project at hand. The student also learned how to use Firestore's authentication and Cloud Firestore database to their full potential, and to ensure a high level of security for the system users.

Additionally, much was learned about the software development process, including the steps needed to be taken at each stage of the development process and the importance of good time management. It was also learned that checking in with the project owner of a project is important, to ensure that what is being developed is suitable for the system that the project owner had originally envisaged. Version control is another valuable part of this process, and can often help avert disaster, such as by allowing a developer to revert accidental changes that negatively affect the entire system.

Appendix

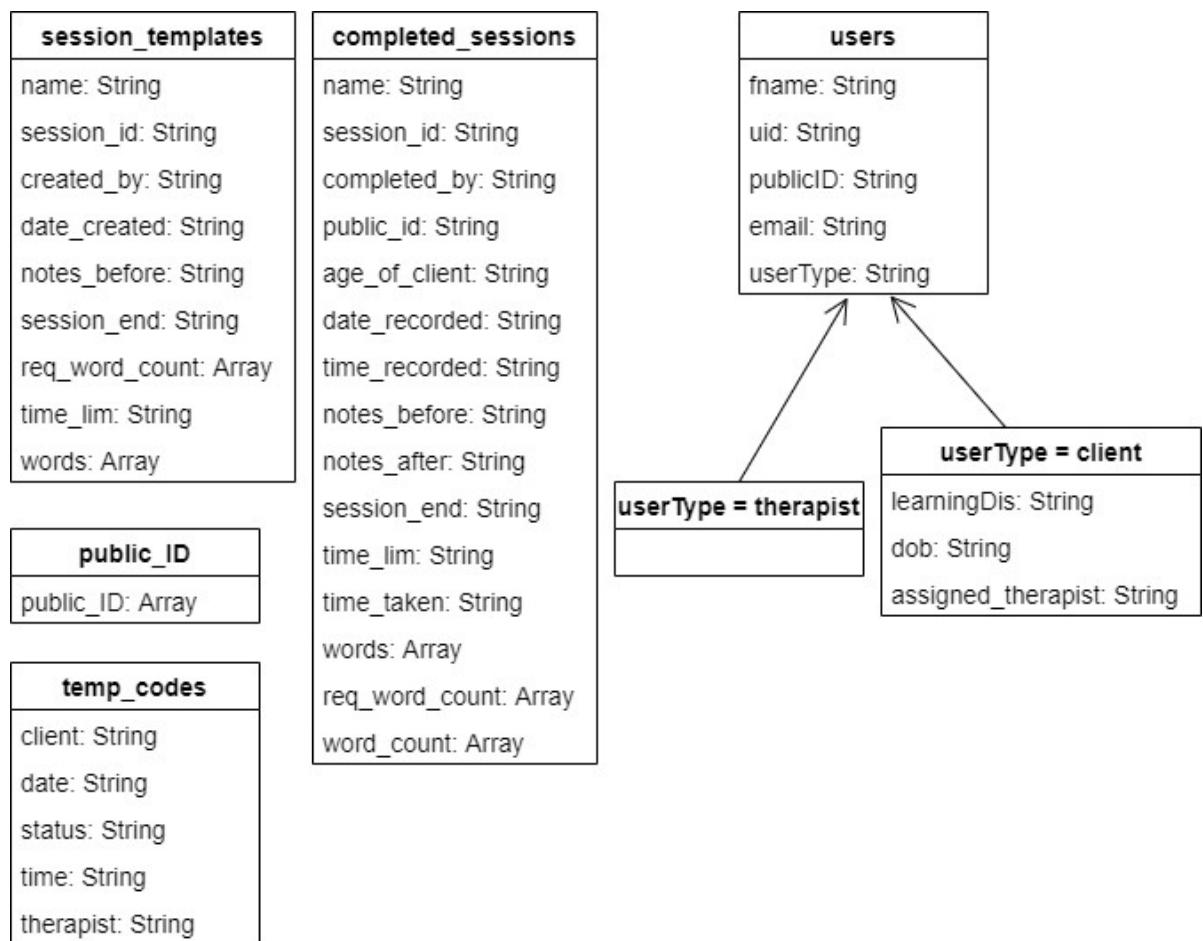


Figure 1: UML Class diagram depicting the database layout

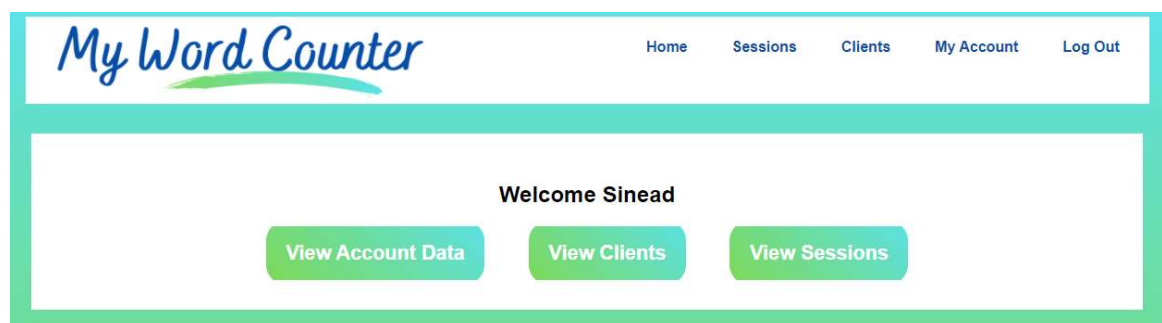


Figure 2: Home page from web app prototype

Show entries

Search:

First Name	Surname	DOB	ID	
Mary	Baker	28/7/2016	exampleC4	More Information
Mary	Drew	11/5/2016	exampleC7	More Information
Patrick	Smith	26/10/2016	exampleC21	More Information
Tom	Drew	27/5/2014	exampleC0	More Information

Showing 1 to 4 of 4 entries

Previous Next

Figure 3: Data Table displaying client information from web app prototype

Edit Account Data

First Name

Surname

Email Address

Password

[Save Changes](#) [Cancel](#)

Figure 4: Example of a form from the web app prototype


Welcome Sinead

[View Account Data](#) [View Clients](#) [View Sessions](#)

Figure 5: Buttons from the home screen, converted to black and white

[Edit](#) [Delete Account](#)

Figure 6: Account modification buttons, converted to black and white



← Edit Account Data

Email

Password

Child's Name

Child's Surname


Child's Date of Birth:

DD MM YYYY

Learning Disabilities

SAVE CHANGES

Figure 7: Example of forms from mobile app prototype



Sinead Tester

RECORD SESSION

VIEW PREVIOUS SESSIONS

VIEW SESSION TEMPLATES

VIEW ACCOUNT DATA

Figure 8: home screen of the mobile app prototype

Add New Session Template

Example

cat dog mouse

*Please enter a list of words, separated by a space. More than 10 words per session will not be accepted.

The session is ended by: Word Count ▾

Word	Count
cat:	3
dog:	4
mouse:	

Add

Figure 9: An example of dynamic forms for required word count, eventually removed from the final implementation

Mary Butler - exampleC24
John Jones - exampleC25
Jane Jones - exampleC26
Mary Baker - exampleC27
Eliza Baker - exampleC28

Assign session to: Eliza Baker - exampleC28

Figure 10: An example of dynamic 'drop-down' menu for assigning a session to a client, eventually removed from the final implementation

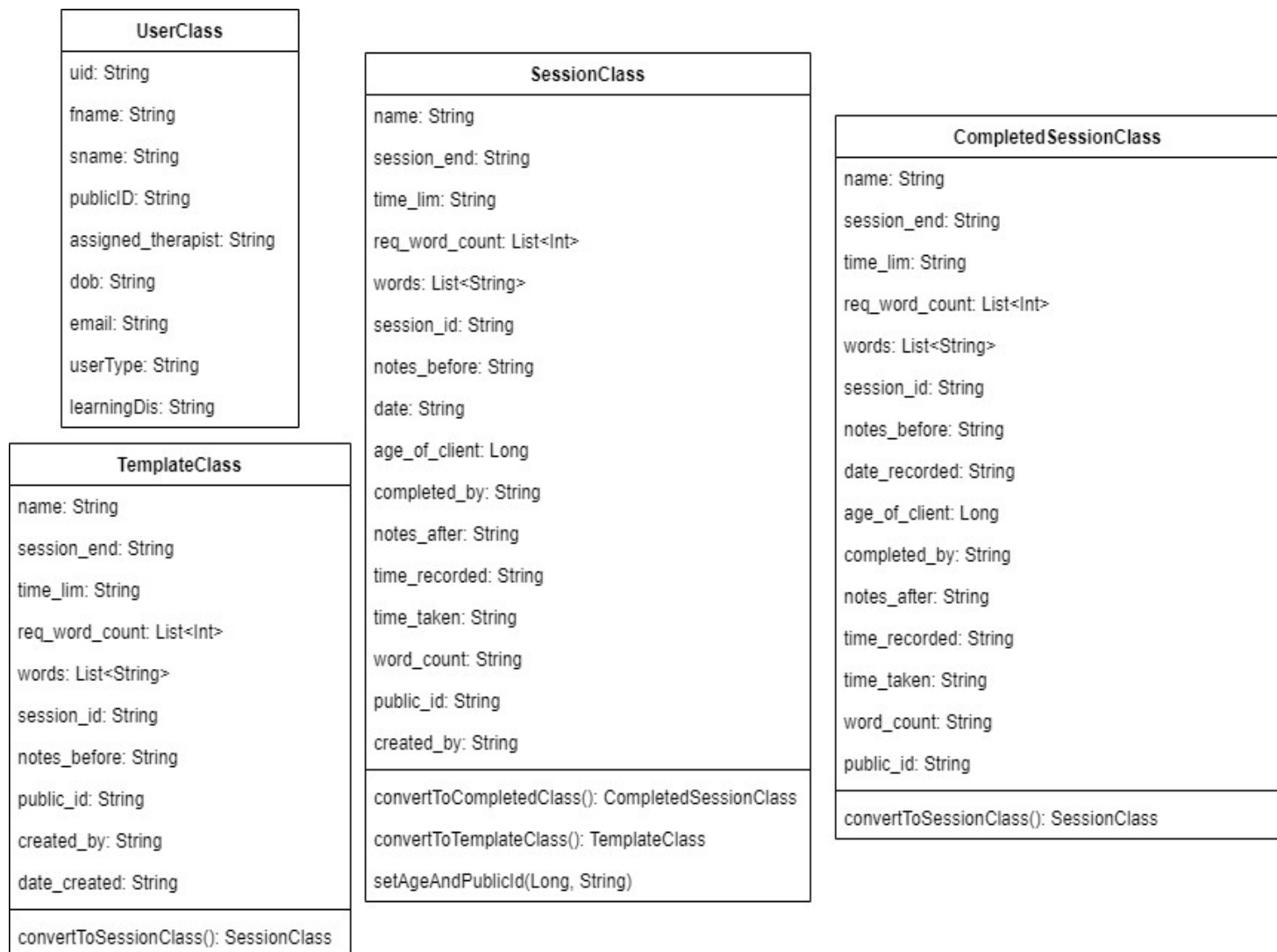


Figure 11: UML Class diagram depicting the classes used to store data in the mobile application

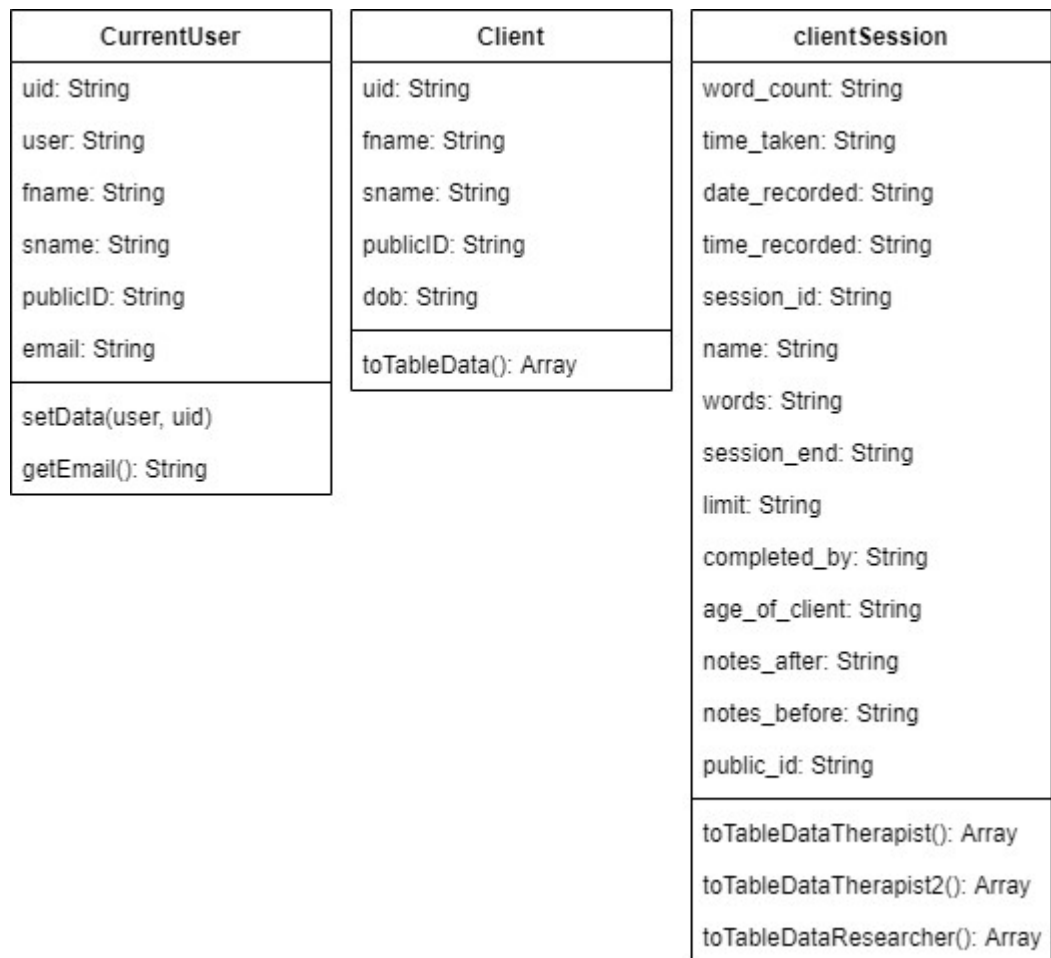


Figure 12: UML Class diagram depicting the classes used to store data in the web application

```

function updateWordCount(){
    //responsible for updating the dynamic table that allows someone to pick a word count for a word
    if (document.getElementById("word_count_selector").style.visibility != "hidden"
    || document.getElementById("word_count_selector").style.visibility != "none"){
        var words = $("#words")[0].value.split(" ");
        var table = document.getElementById("word_count_table");
        rows = table.rows;
        for (i = 0; i<words.length; i++){
            if (words[i].length > 1 && i < 10){

                // create the input for the word count
                var newInput = document.createElement("INPUT");
                newInput.setAttribute("type", "text");
                inputID = "wordCount_"+i;
                newInput.setAttribute("id", inputID);
                newInput.setAttribute("maxlength", 4);

                // if there is already a row for this i value (i.e. position in the list of words)
                if (i < rows.length-1){

                    // rows will want to start at 1 (because row[0] has the headings)
                    row = rows[i+1]
                    cell1 = row.children[0];
                    cell2 = row.children[1];
                    cell1.innerHTML = words[i]+":";

                }
                else {
                    var row = table.appendChild(document.createElement('tr'));
                    row.setAttribute("id", i);
                    var cell1 = row.appendChild(document.createElement('td'));
                    var cell2 = row.appendChild(document.createElement('td'));
                    cell1.innerHTML = words[i]+":";
                    cell2.appendChild(newInput);
                    cell2.setAttribute("class", "form-input");
                }
            }
            if (words.length < rows.length-1){
                // deletes rows of words that have been deleted
                for (y=words.length; y < rows.length-1; y++){
                    table.deleteRow(y);
                }
            }
        }
    }
}

```

Figure 13: Full implementation of the function used to update the required word count form field, later removed from the web application

```

function client_dropdown(){
    var div = document.getElementById("assigned_to");
    var label = div.appendChild(document.createElement('label'));
    var select = div.appendChild(document.createElement('select'));
    select.setAttribute("id", "client_dropdown")
    select.setAttribute("class", "custom-select")
    label.setAttribute("for", "client_dropdown");
    label.innerHTML = "Assign session to: "
    db.collection("users").where("userType", "==", "client")
        .get().then(function(querySnapshot) {
            querySnapshot.forEach(function(doc) {
                var client_string = doc.data().fname+" "+doc.data().sname+" - "+doc.data().uid;
                var option = document.createElement("option");
                option.text = client_string;
                select.add(option);
            });
        });
}

```

Figure 14: Full implementation of the function used to create a dynamic dropdown menu, later removed from the web application