National College of Ireland

BSHC Computing

Software Development

2022/2023

Eoghan Feighery

x19413886

x19413886@student.ncirl.ie

MedsUpApp

Technical Report

Contents

Executive Summary

The purpose of the report is to detail how the MedsUp application will work in terms of functionality, data saves and what requirements I set out to achieve.

The app is designed to cater to the general public, elderly people who may struggle with remembering when to take their medication, parents who may want to help keep track for their kids' medicine. It can even be used by young and middle-aged adults who are swamped with work and other life commitments and may forget about their medication (varies on the person) requirements.

1.0 Introduction

1.1. Background

I chose to create the application because since we've reached the end of the Covid-19 pandemic, we've all gotten somewhat weary about our state of health and that can even be extended to elderly relatives and to the younger set.

And now that everyone is back in school/work and/or college, we can be so caught up in that busy mindset and for anyone with prescribed medication, it can be easy to forget about when we need to take them.

On top of that, while there are many medical reminder applications, very few of them have the option to scan prescriptions to get the desired text needed and add it to a calendar.

As such, my application sets out to address these two concerns so that users and their relatives can be informed with one another about when they follow through with their medication.

For example, Medisafe Pill & Med Reminder (Google Play, 2023) has features such as medicine trackers and reminders for refills and health measurement and medicine trackers. It even contains a manager and calendar system for doctor appointments. It does contain a lot of features and does have a medication reminder functionality, but it doesn't seem to mention any microphone recorder options for the reminder. MyTherapy Pill Reminder (MyTherapy, 2023) is similar to Medisafe, except the former has a health journal for a user's doses, tablets and state of mind which also contributes to a pill diary for taking necessary doses.

CareZone (Medicare.org, 2023) is a multi-user app that lets families handle medications and appointments with pill reminders and an incrementor for taken doses plus a camera function to import details regarding a prescription. Family members can co-ordinate with each other especially in an emergency. MangoHealth takes the concept of managing healthcare and medication rounds and makes it more fun. MangoHealth (Treichler, 2022) lets users organise a schedule with medication reminders and health habit customisation options. On top of that, the app records medication data so users can see how a medication can coincide with other medications and food items. Finally, it has a note taker for tracker user medication progress and results whilst also letting them share the information with their GP along with a reward system.

From these 4 examples, while the idea of a medication reminder isn't the most original one, it did help me to confirm that a speech-to-text recogniser hasn't been attempted as frequently compared to the features I mentioned in the other applications.

1.2. Aims

One aim of my project is to create a text recogniser functionality for the application that can find text from prescriptions and medication box information. Users can activate a microphone and say out loud their medication instructions to theoretically produce the desired text they need. Next, a button will also copy the details and adds them to the clipboard.

Another aim would be to save user details when they register an account to a user profile that can display various details such as gender (if they feel comfortable in doing so) age, and email address. Plus, I also plan to let users add contacts so that an email or a text message can be sent to a contact through their own pages. I also plan to allow users to save medication instructions and any medical diagnosis via the CRUD methodology.

Finally, yet another aim is to allow the ability to feature a reminder of medication details to a calendar interface that can let users set a title, detail, and time for when the reminder goes. One option would be to use an API for Google Calendar that can support this functionality or use Android Intent to allow users to add the reminder to their Google Calendar.

1.3. Technology

One technology that I've set out to use is Android Studio alongside with Java. After some issues with trying to use Ionic and React for my application, I decided to stick to a technology that I had more familiarity with. Since I've studied with Java the most ever since my PLC course, it made the most sense to go with that instead.

To use Android Studio, I need to create the layout for the app's pages that can provide the structure for where the features will be placed, and the code needed to implement the functionalities outlined above. The same even rings true when developers use Kotlin.

As for the database needed to store the user accounts, contact details and medication information, I plan on using Firebase. Firebase's authentications offer backend services and UI libraries that can better authenticate users when they register for the first time. In their official documentations (Firebase, 2023), it offers multi-factor authentication which brings in extra security. Additionally, its Realtime Database can be stored locally and synchronised for any data changes even when offline. Meanwhile, the Cloud FireStore database offers quicker query speeds and better scaling.

1.4. Structure

If MedsUp could be split into stages, it would be the following:

The first one will be centered around the maintenance and the operations of the app itself, user authentication and creation, adding contact details, saving medication-based text results, and connecting to Firebase databases to store all the aforementioned information.

The app's pages will go like this:

1) Home page
2) Notification creator page
3) Speech to text Recognition page
4) Contact Page
5) Medication page
6) User Profile

The second module is responsible for the text recognition functionality via speech to text transcription technology. Since, this will function much like the microphone button in Google Translate to better implement the feature so that it can allow the user to scan the required instructions with the click of a button.

The third potential module will handle how to use the Google Calendar API. One option can be to add events to a user's Google Calendar by typing the title and description for that reminder and setting the reminder time and duration accordingly. Another option would be to create a CRUD feature to add events to a date chosen by a user.

2.0 System

2.1. Requirements

Security and privacy can be considered the higher priorities since users want the reassurance that their account details and personal information will be protected and store safely. The same rings true for practically every web-based application whether its mobile or desktop-based.

2.1.1. Functional Requirements

The home page will offer a brief description of what the app is about and the functionalities it provides will be displayed as buttons. Each page will also have a button that can bring them back to the home page for easier navigation.

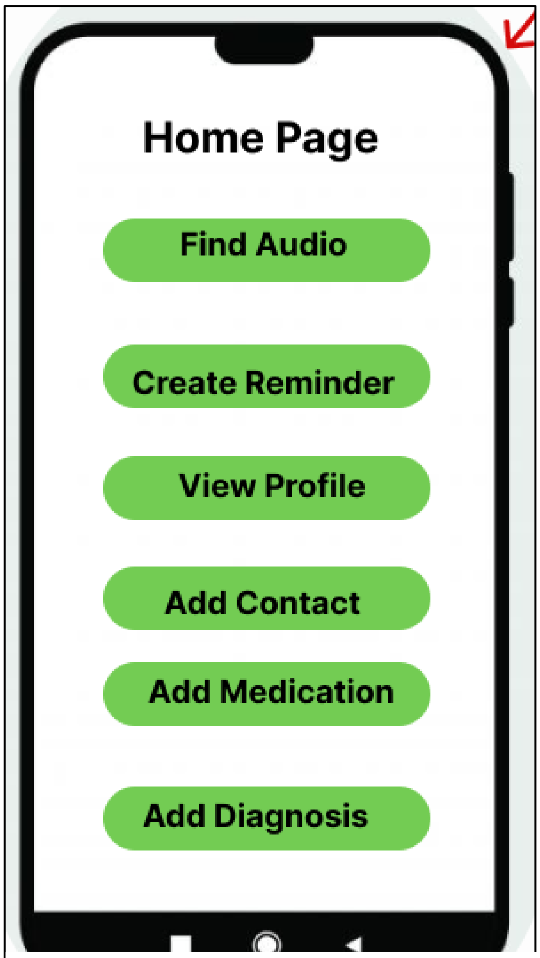


Figure 1: How the home page will look to the user

Upon starting up the app, you'll first see the splash screen that displays the app's official logo which loads into the account registration page.

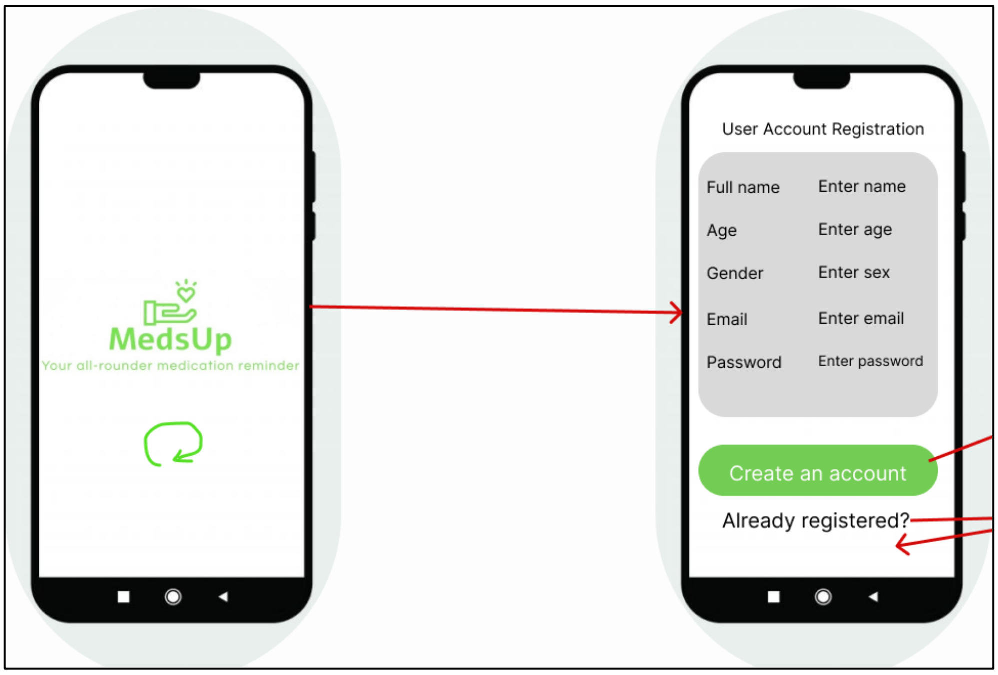


Figure 2: Transition from the splash-screen to the register screen

The registration page (Transitioned from Figures 1 to 2) will allow users to create an account and add their details (age, gender). It also contains a button for moving to the login page if the user has made an account beforehand. The login page does have similarities to the registration expect it lets registered users re-enter their account once they made an account and return to the homepage. The additional register button takes you back to the registration page if you haven't made an account beforehand (Navigation would be Figure 2 to Figure 3 and vice versa).

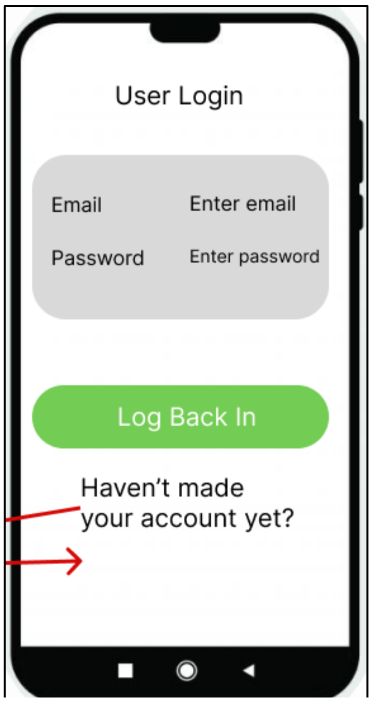


Figure 3: The overall look of the login screen and the required credentials

Clicking on the first button titled Speak Audio (Figure 4) will bring you to the Speech to Text page (Figure 5). Here, users can click on the microphone button to activate the speech recogniser technology to record and grab text from any audio it finds.

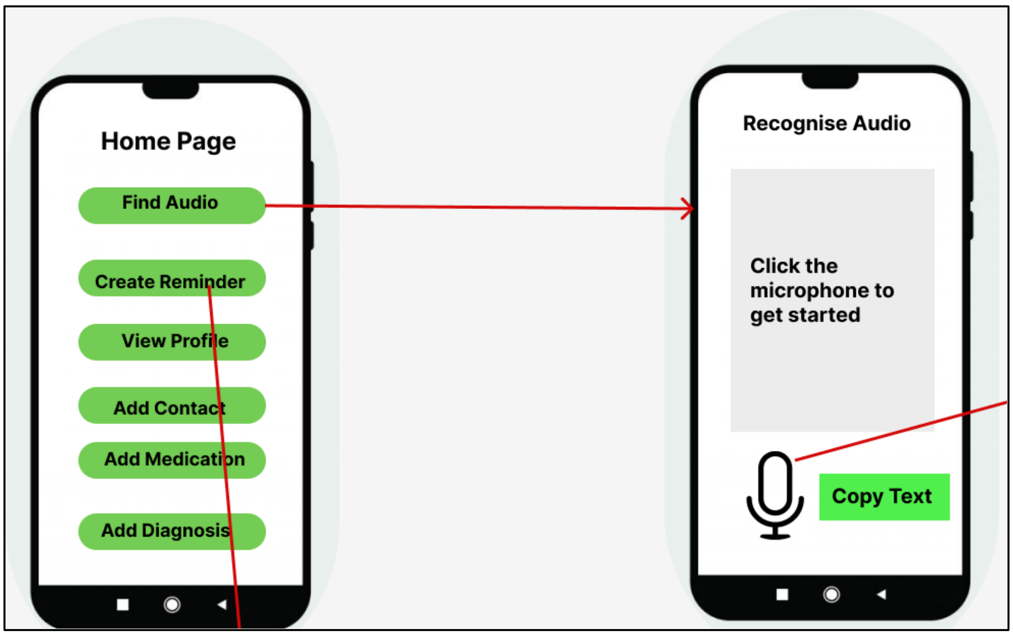


Figure 4: Going to the Recognise Audio page from the home page.

When the microphone button is clicked, you'll see a screen prompting to say out loud your medication instructions. Once you do that, whatever text was found from the audio will be shown on the box. Next to the microphone is the button that can allow you to copy the text onto your phone's clipboard.

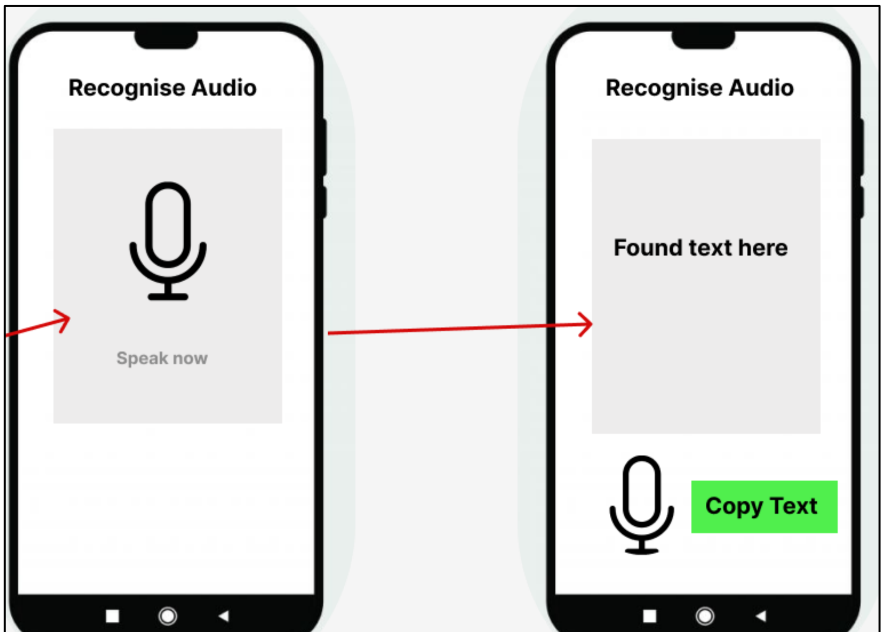
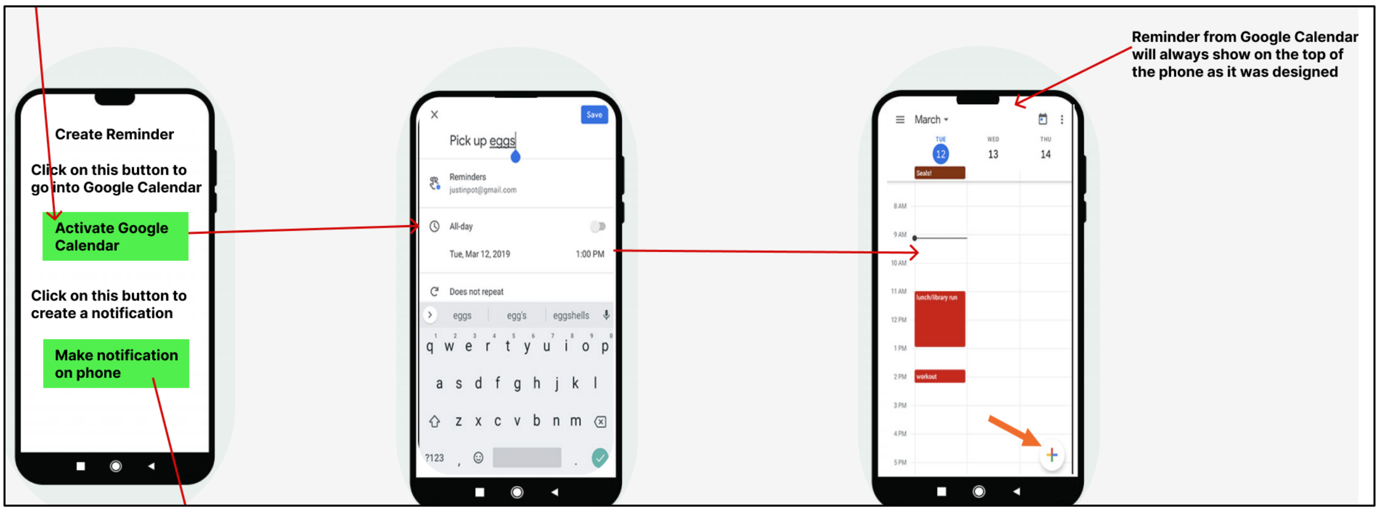


Figure 5: Clicking on the microphone will lead to this prompt where users can say their medication instructions out loud. The recogniser technology will then produce the found text into the grey text box.

The camera page lets users use the camera feature to take a picture of a medication box or writing and click another button to use an OCR to scan for any text (From Figure 4 to 5).

If the user clicks the "Create a Reminder" button, they'll be taken to the following screen. Choosing the first button will take you to your Google Calendar where you can paste the medication instrcutions you got from the Speech to Text page to the


Create Reminder
Click on this button to go into Google Calendar
Activate Google Calendar
Click on this button to create a notification
Make notification on phone
Save
Pick up eggs
Reminders
justinpot@gmail.com
All-day
Tue, Mar 12, 2019
1:00 PM
Does not repeat
Reminder from Google Calendar will always show on the top of the phone as it was designed
March

calendar reminder field and can adjust it to however long you want it to be. One thing to note is that when you first start it out, it will ask you to sign in to your Gmail account but luckily you'll only have to do it once. When you save it, the reminder should function as usual.

Clicking on the Notification Maker button will take you to a screen (Figure 6 from image 1) where you can make a reminder with the Date Time Picker schematics similar to the Real Time Bus App. Here, you can adjust the time for when to take your medication by altering the time to what you want. Saving it will mean that the alarm and notification will trigger at the designated time.

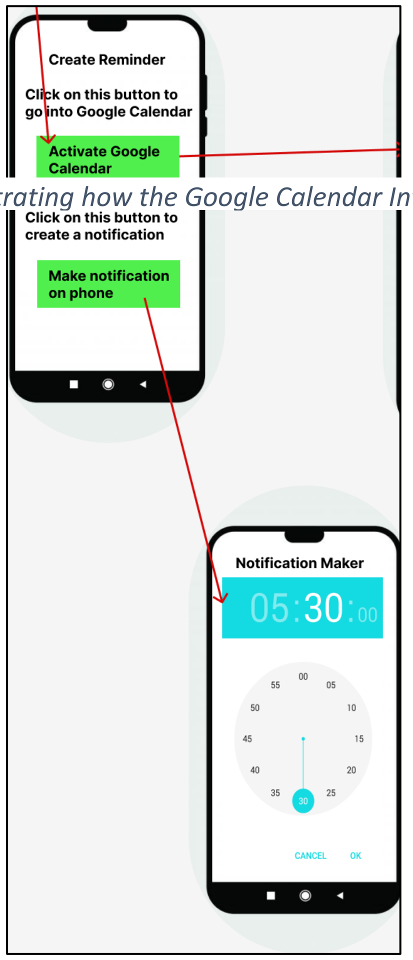

Create Reminder
Click on this button to go into Google Calendar
Activate Google Calendar
Click on this button to create a notification
Make notification on phone
Notification Maker
05:30:00
CANCEL
OK

Figure 6: Demonstrating how the Google Calendar Intent object will function.

Figure 7: How a user can create a notification reminder to trigger at a certain time.

The contact page grants the ability to save contacts to a Firebase Realtime Database and buttons that can let them send emails/text messages for when a user goes through a medication. This will go from Figure 4 to Figure 7.

When the user enters the Contact List page, they should see a scrollable list of saved contacts (if they saved one prior) and a series of buttons down below.

Clicking on the Add Contact button will bring you to a small pop-up window that can allow you to enter a name, email address and telephone number for a contact. Saving it will add the entry into the list view which also contains two buttons: Edit which lets users alter any of the three fields which are then displayed when saved or Delete which will remove the contact entry if you no longer need it.

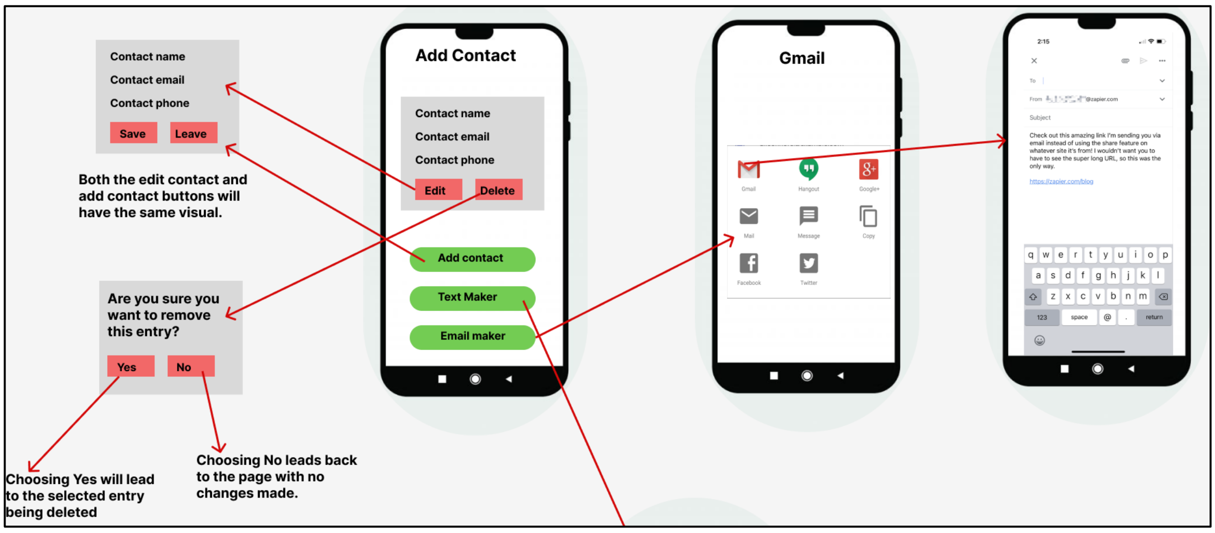

Contact name
Contact email
Contact phone
Save
Leave
Both the edit contact and add contact buttons will have the same visual.
Are you sure you want to remove this entry?
Yes
No
Choosing Yes will lead to the selected entry being deleted
Choosing No leads back to the page with no changes made.
Add Contact
Contact name
Contact email
Contact phone
Edit
Delete
Add contact
Text Maker
Email maker
Gmail
Gmail
Hangout
Google+
Mail
Message
Copy
Facebook
Twitter

Figure 8: Demonstrating how a contact entry can be added to the database and how the email maker will function in conjunction with Gmail.

Clicking on the Email Maker will take you to a page that can let you craft an email to a contact to say that you've taken your medication. Once the sender, subject and the email body is all filled in, you can click a button that can take all of these and put onto a draft in Gmail. After sending you'll need to exit from Gmail and go back to the MedsUpApp using the back button. (Figure 8 from Figure 4)

If you want to send a text message instead, clicking on the button titled "Text Maker" will bring to this page. Here, to send a text message, you'll need to enter a phone number and type in a message in the field below.

Then, click the Send button to send the text message which should display a notification saying that it went through.

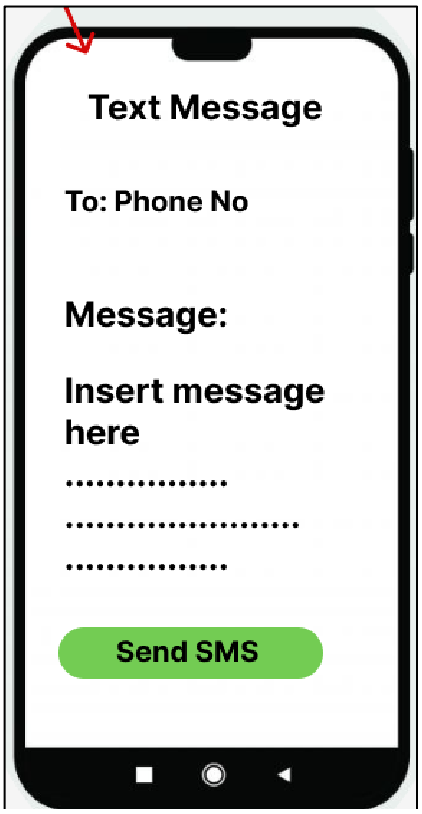


Figure 9: Demonstrating the page where users can send text messages to users.

The profile page is where the user's account details are stored and displayed to the user. It will take the desired account information from the users table in the database and place them in the appropriate fields. Important note is that the profile functions on the Realtime Database while the registration and login pages work under the Firebase Authentication schematic. Like all the other pages, there's a home page button for smoother navigation between pages (In Figure 10, indicated by the red arrows).

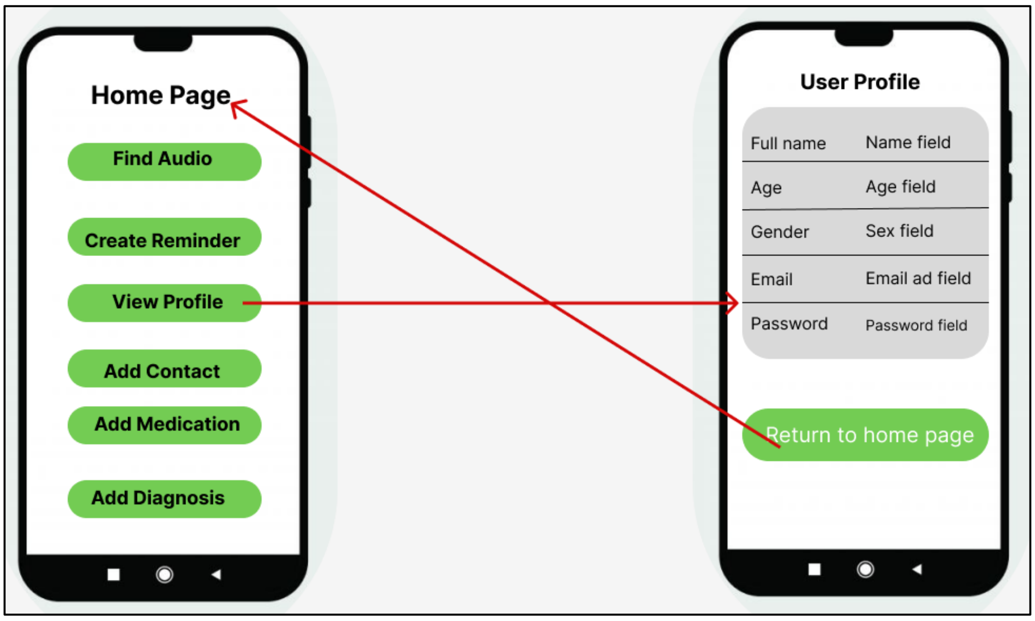


Figure 10: Transition from the home page to the User Profile. Also showcases what should happen when the profile is viewed by the user.

Selecting the "Add Medication" button will bring you to the designated page featuring a scrollable list of saved medication items. Clicking the add medication button will bring you to a pop-up where you can add a name of the medication and the instructions for them. Saving them will add the entry box onto the list and feature two extra buttons inside. "Edit" involves editing the fields of an entry and saving them accordingly and "Delete" will remove the entry from the list.

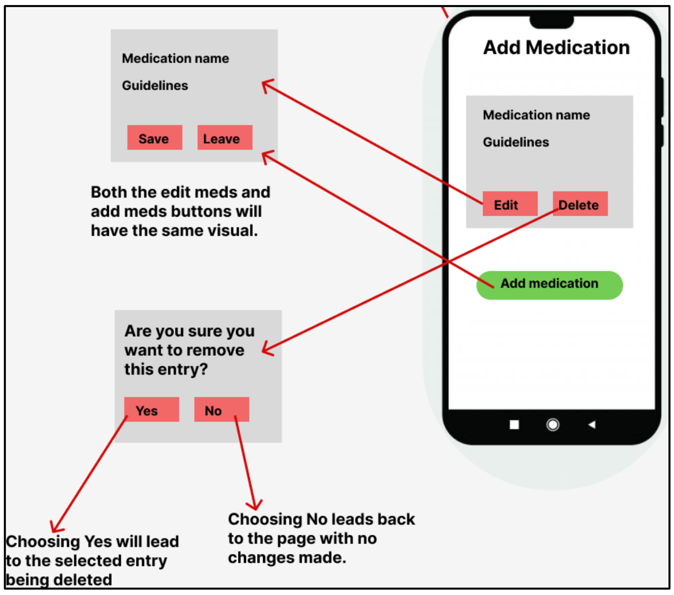

Medication name
Guidelines
Save
Leave
Both the edit meds and add meds buttons will have the same visual.
Are you sure you want to remove this entry?
Yes
No
Choosing Yes will lead to the selected entry being deleted
Choosing No leads back to the page with no changes made.
Add Medication
Medication name
Guidelines
Edit
Delete
Add medication

Figure 11: Demonstrating the CRUD functionality in the Add Medication Page

Selecting the "Add Diagnosis" button will bring you to the designated page featuring a scrollable list of saved medication items. Clicking the add medication button will bring you to a pop-up where you can add any known medical diagnosis that a user may have. Saving them will add the entry box onto the list and feature two extra buttons inside. Edit involves editing the fields of an entry and saving them accordingly and Delete will remove the entry from the list (A three way system in both figures 11 and 12).

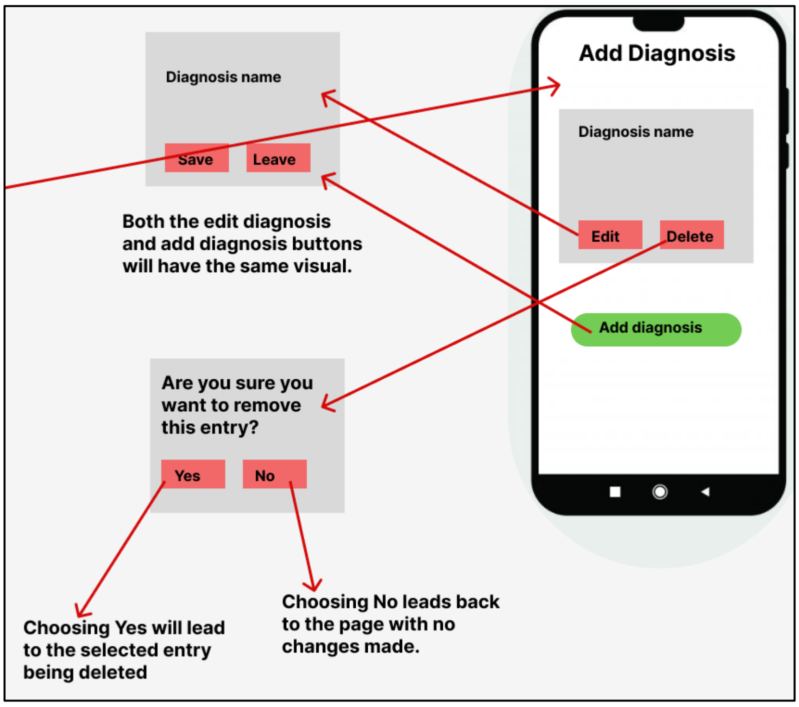

Diagnosis name
Save
Leave
Both the edit diagnosis and add diagnosis buttons will have the same visual.
Add Diagnosis
Diagnosis name
Edit
Delete
Add diagnosis
Are you sure you want to remove this entry?
Yes
No
Choosing Yes will lead to the selected entry being deleted
Choosing No leads back to the page with no changes made.

Figure 12: Showcasing the CRUD functionality for any known medical diagnosis.

2.1.1.1. Use Case Diagram

2.1.1.2. Requirement 1: User Registration

2.1.1.3. Description & Priority

The use case allows users to create a new account to be given access to the app's functionalities and features. It's essential since it's needed to save user details to the Firebase database.

2.1.1.4. Use Case S1

Scope

The scope of this use case is to offer users the ability to register a new account.

Description

This use case describes the new user registration methodology.

Use Case Diagram

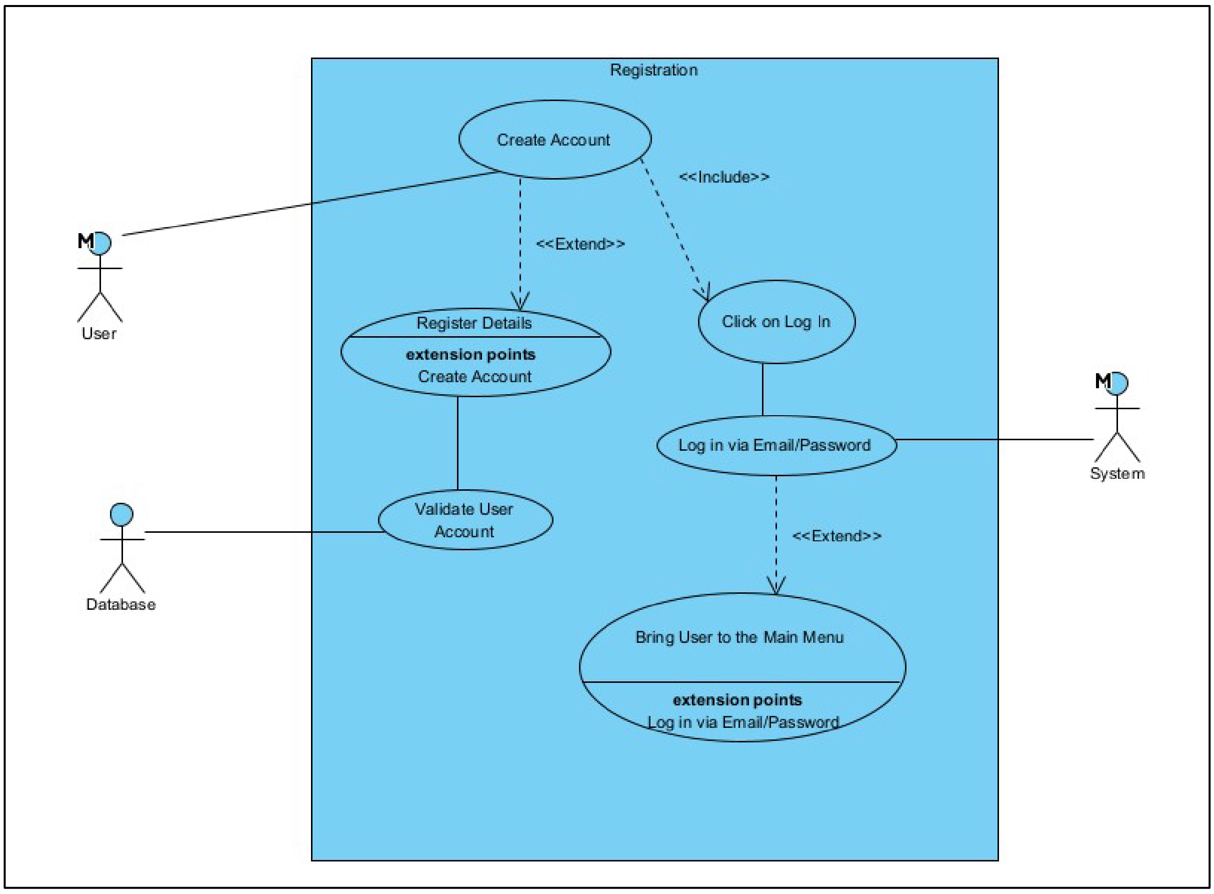


Figure 13: Use Case Diagram for the User Registration and Login

Flow Description

Precondition

The system opens at the registration page

Activation

This use case starts when a user opens the app and wishes to make a new account

Main flow

1. The user types in all the fields needed for a new account to be added to the database
2. The system takes all of the details and adds them into a user table on Realtime (See A1)
3. The system then moves to the Log in(See E1)
4. The User logins in with their email and password

Alternate flow

A1 : Go to the Login Page

1. The system will have the user details saved on the database

2. The User will click on the button that will take them to the login page.
3. The use case continues at position 3 of the main flow

Exceptional flow

E1 : Missing Details

4. The system will recognise the missing fields and inform the user of it.
5. The User will need to fill the fields.
6. The use case continues at position 4 of the main flow

Termination

The system presents the next stage which is the home page

Post condition

The system goes into a wait state

2.1.1.2 Requirement 2: Text Recognition

2.1.1.2.1 Description & Priority

The use case allows users to create a new account to be given access to the app's functionalities and features. It's essential since it's needed to grant authorisation from Firebase.

2.1.1.3.2 Use Case S2

Scope

The scope of this use case is to use the camera interface to take a picture and recognise the text to produce the text results

Description

This use case describes the camera and text recognition features.

Use Case Diagram

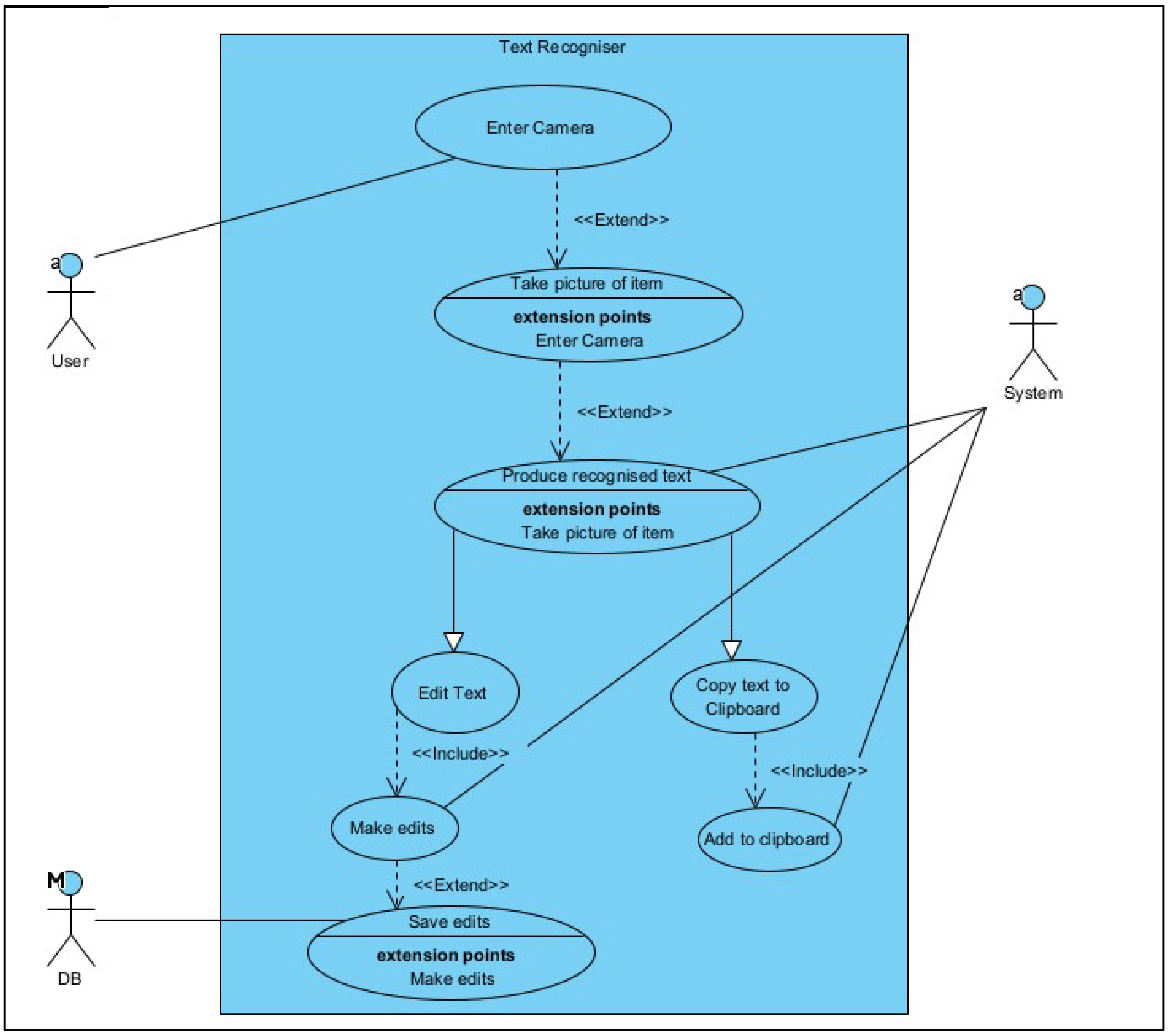


Figure 14: Showcasing the text recogniser functionality.

Flow Description

Precondition

The system enters the camera page when the button is clicked.

Activation

This use case starts when a user activates the camera

Main flow

1. The system will ask for permission when the camera is used.
2. The User allows the permission grant and takes a photo of some text.
3. The system will activate the OCR to find the required text
4. The User can then either copy text to a clipboard or make edits to the text result.

Alternate flow

A1 : Edit Text Results

1. The system takes the user to a page with the found text to make any edits.
2. If step 3 of the main flow succeeds, then the user will make edits for any perceived orders from a GP.
3. The use case continues at position 4 of the main flow

Exceptional flow

E1 : No Text Found

4. The system will run the text recognition technology but find no text in the photo
5. The User will need to then choose/take a picture of text to remedy it
6. The use case continues at position 4 of the main flow

Termination

The system will save the edits to a clipboard and returns the user to the home page.

Post condition

The system goes into a wait state

2.1.1.3 Requirement 3: Calendar

2.1.1.3.1 Description & Priority

The use case lets users interact with a calendar layout to paste in the medical information in a chosen date and set the duration and alarm reminders which is passed as a notification.

2.1.1.3.2 Use Case S3

Scope

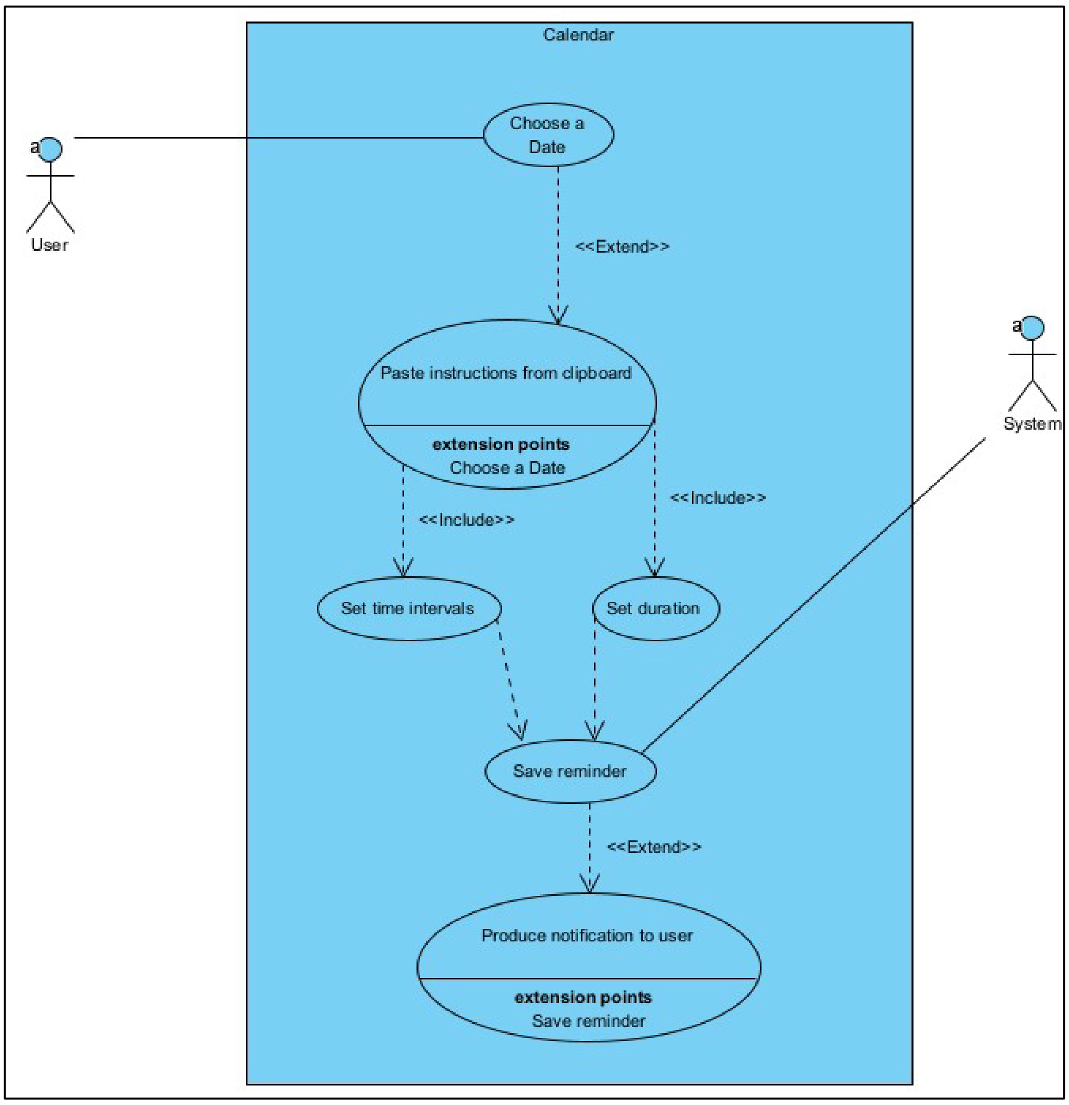
The scope of this use case is to allow a user the ability to use a calendar and enable a reminder which can display the medical information via notifications

Description

This use case describes the calendar interface.

Use Case Diagram

Figure 15: Use case diagram for how users can add reminders to the Google Calendar



Flow Description

Precondition

The system is in the calendar page for the current month.

Activation

This use case starts when the user will click on the Calendar button.

Main flow

1. The system identifies the current month and date
2. The User will choose a date and paste the text into the fields while setting the times for it.
3. The system will save the reminder and the calendar will display it accordingly.
4. The User can click on the date to see the newly created reminder

Alternate flow

A1 : No reminders added

5. The system will display the current date without reminders
6. The User can see the default calendar view with no events added.
7. The use case continues at position 2 of the main flow

Exceptional flow

E1 : No entries filled

8. The system will remind user if they want to continue.
9. The user will choose Yes to exit.
10. The use case continues at position 4 of the main flow

Termination

The system presents the calendar view to the user

Post condition

The system goes into a wait state

2.1.1.4 Requirement 4: Adding contacts

2.1.1.4.1 Description & Priority

The use case lets users enter and save contact information onto the database once certain specified information requirements are met. They can even edit and remove saved contacts if they wish to do so.

2.1.1.3.2 Use Case S4

Scope

The scope of this use case is to allow a user the ability to enter details of a potential emergency contact via the CRUD methodology (Create, Read, Update, Delete).

Description

This use case describes the adding contacts interface.

Use Case Diagram

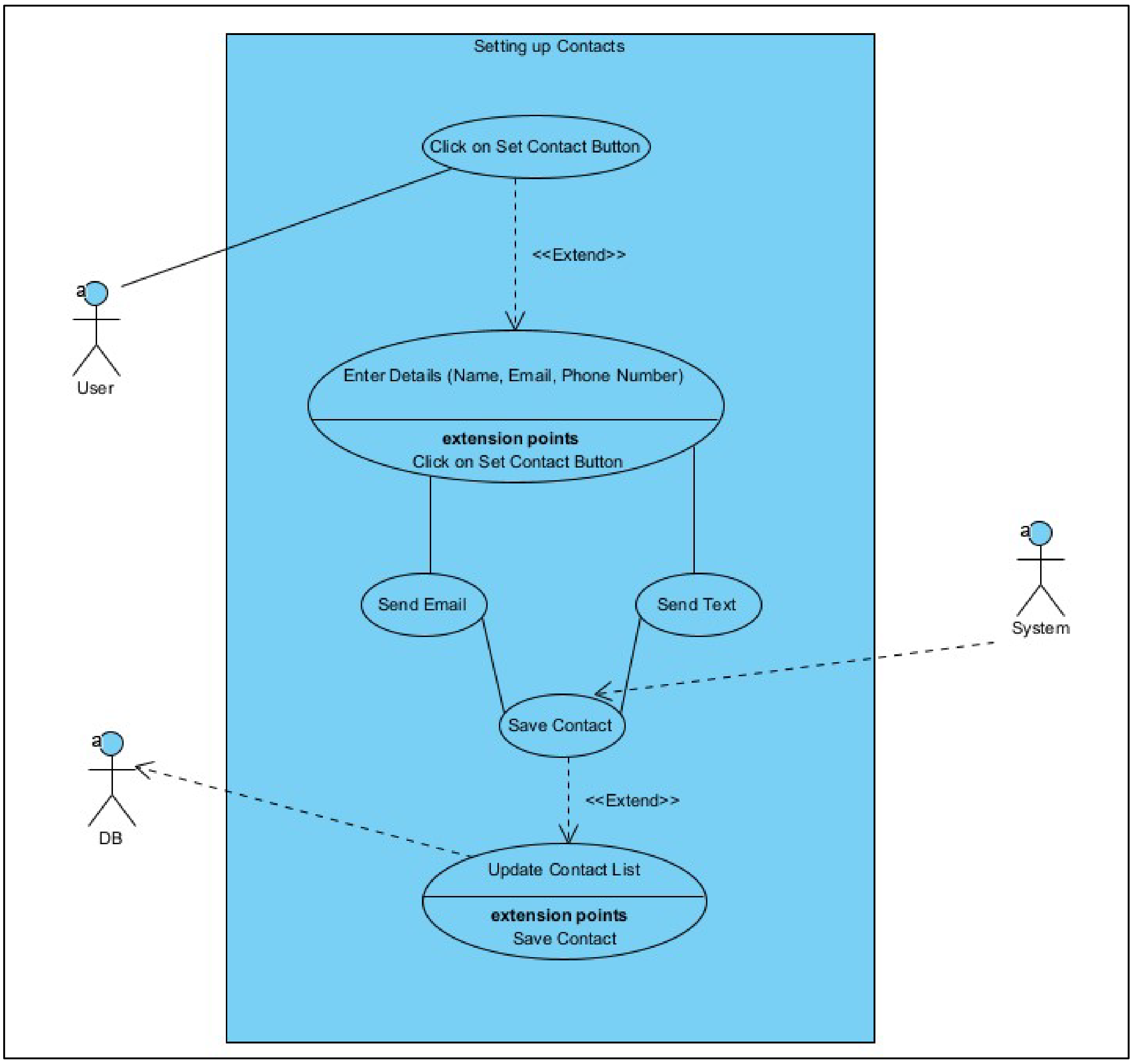


Figure 16: Use Case Diagram for adding contacts and sending messages and emails to them.

Flow Description

Precondition

The system is in the Contact list page

Activation

This use case starts when a user activates the Add Contact Button

Main flow

11. The system identifies any contacts that were saved (if applicable)
12. The User will add all of the details
13. The system will save the information to the database and in the list
14. The User will be able to see newly create contacts.

Alternate flow

A1 : Send text/email

15. The system will let the user confirm the email to send to the contact
16. The User will allow the confirmation needed for the email.
17. The use case continues at position 3 of the main flow

Exceptional flow

E1 : No fields filled

18. The system will confirm that no edits are made to the contact
19. The user will add the details if they decide to do so
20. The use case continues at position 13 of the main flow

Termination

The system presents the updated contact list.

Post condition

The system goes into a wait state

2.1.1.5 Requirement 5: Update User Profile

2.1.1.5.1 Description & Priority

The use case is where the user profile will be created when the user creates a new account and allow for the addition of new user details.

2.1.1.3.2 Use Case S5

Scope

The scope of this use case is to allow a user the ability to view and edit their user profile to add any more information they want to confirm.

Description

This use case describes the user profile.

Use Case Diagram

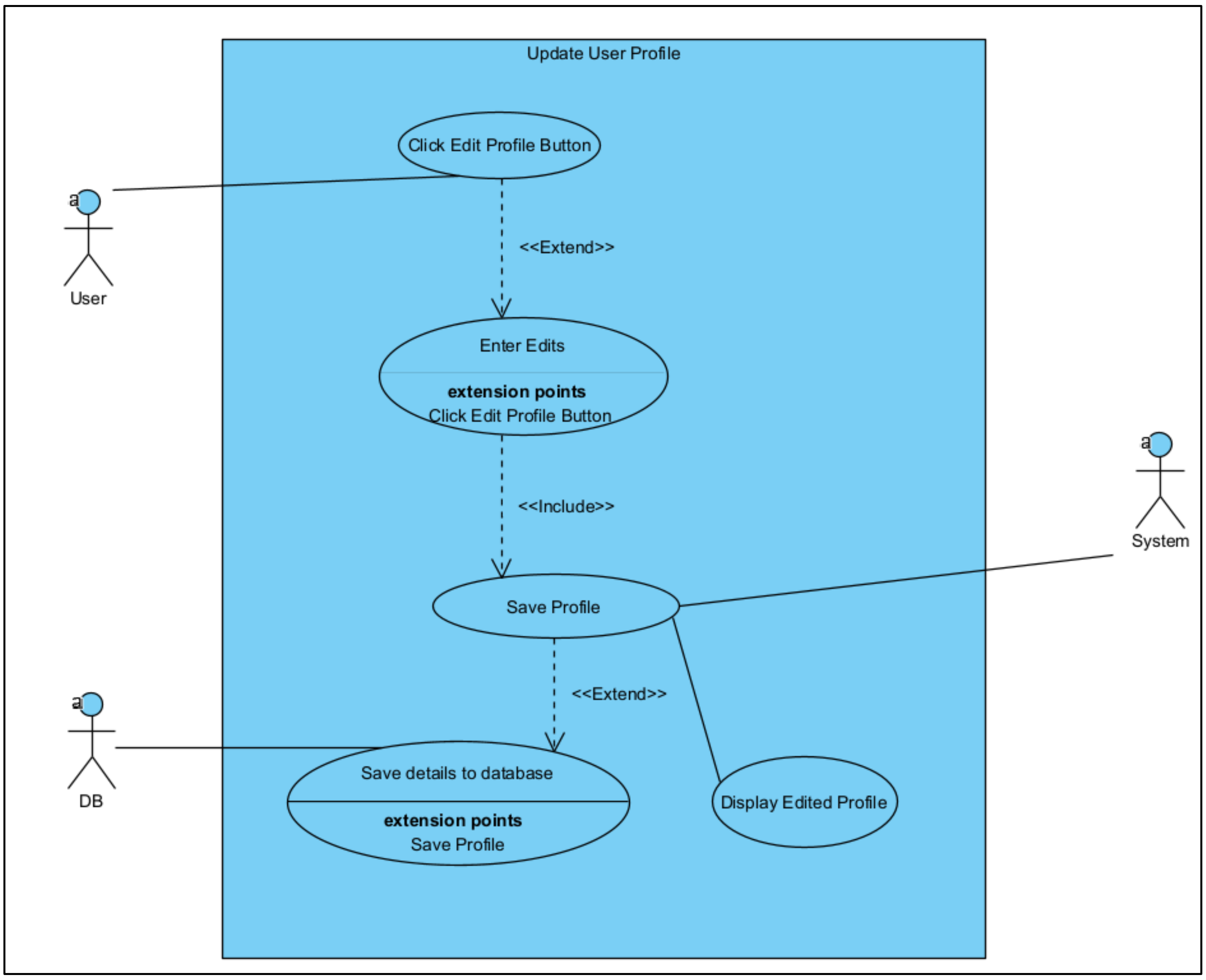


Figure 17: Use case diagram for the User Profile

Flow Description

Precondition

The system will be in the direction of the user profile

Activation

This use case starts when a user clicks on the Edit Profile Button.

Main flow

21. The system identifies the button click and brings the user to Edit page
22. The User will be able to make any edits to the profile variables
23. The system will check for any confirmed edits done and if so, will save it to the database.
24. The User will be able to see their confirmed profile edits.

Alternate flow

A1 : No edits made

25. The system will display a message saying no edits were made.
26. The User will be directed back to the home page
27. The use case continues at position 3 of the main flow

Exceptional flow

E1 : Fields not filled in

28. The system will say that the fields must be filled in
29. The user will add in the details needed to confirm the edits
30. The use case continues at position 4 of the main flow

Termination

The system presents the newly edited profile to the user.

Post condition

The system goes into a wait state

List further functional requirements here, using the same structure as for Requirement1.

2.1.1 Data Requirements

To save user accounts, contact details and medical tables, users will need to enter data for certain fields so that the Realtime Database can be able to save the data for the aforementioned tables. This is also essential since there will be instances where stored data will be retrieved for any updates to the user profile and/or the contact information.

On top of that, for the login schematics, the email and password for each user is stored in order to allow post-registered users to enter back into the application's home page. The two fields are also enhanced with a Firebase Authentication schematic to help ensure secure information storage.

2.1.2 User Requirements

When a person uses the application on their phone, they will need a page that can let them register their account by entering their email and password and personal details. Incidentally, users will also require a home page to clearly specify what the app is all about and what functions it will provide. Each service page will have simple instructions on how users can navigate through each functionality.

2.1.3 Environmental Requirements

2.1.4 Usability Requirements

2.2 Design & Architecture

The Architecture Diagram centers around 3 tiers.

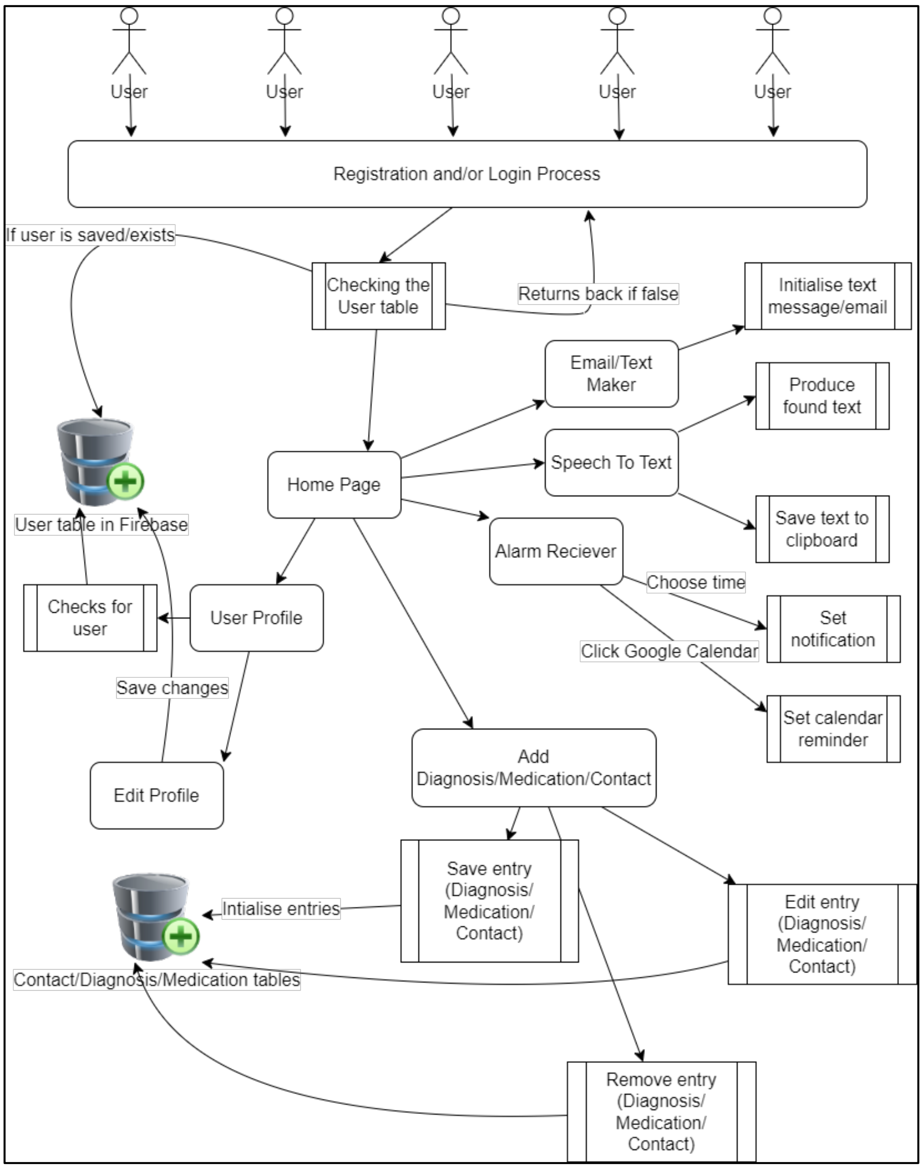


Figure 18: Architecture diagram demonstration processes and actions within each page of the application

When the user registers a new account in the system, the database will take the user account details and save them to a table within the Realtime database. A Firebase authentication object is used to help add a layer of security to the user object. Then, when a user wishes to view their profile, the Realtime database will find the right user entry using a designated ID to ensure the user is found easily. Then the account details will be stored in the correct parameters.

Here, the CRUD functionalities are identical for adding contact, medication information and diagnoses. When a user fills in the required information parameters then the database will add them to their respective tables. If an entry has been updated, then the database will be able to adjust to the changes using an id to ensure that the chosen data entry will match up with what's inside the table. The same logic will also apply for when a data entry is removed as the database will use an id to find the entry that will be removed from the table.

For the speech to text recogniser, the user will click on the microphone causing the system to display the prompt which allows for audio to be recorded. Once the text has been found, the system should in theory, display the acquired text to the user allowing them to edit it to how they see fit.

For the email and calendar schematics, the user will click on the respective buttons which will trigger the system to bring to Gmail/Google Calendar where the user can fill in the required fields (email address, subject and content for an email and frequency and dates for a calendar event).

As for the notification maker, the user will save a title and a designated time for an alarm to trigger. The system should save the notification and activate it when the assigned time is reached (e.g. the alarm will go off at 4:15pm today).

For the class diagram, there's around 7 tables in total. Initially, I planned to have a calendar table but after some reflection, I realised that it seemed a little arbitrary to have. This is also because it's possible to make a recreation of Google Calendar and its reminder systems or even using an API for the former to work.

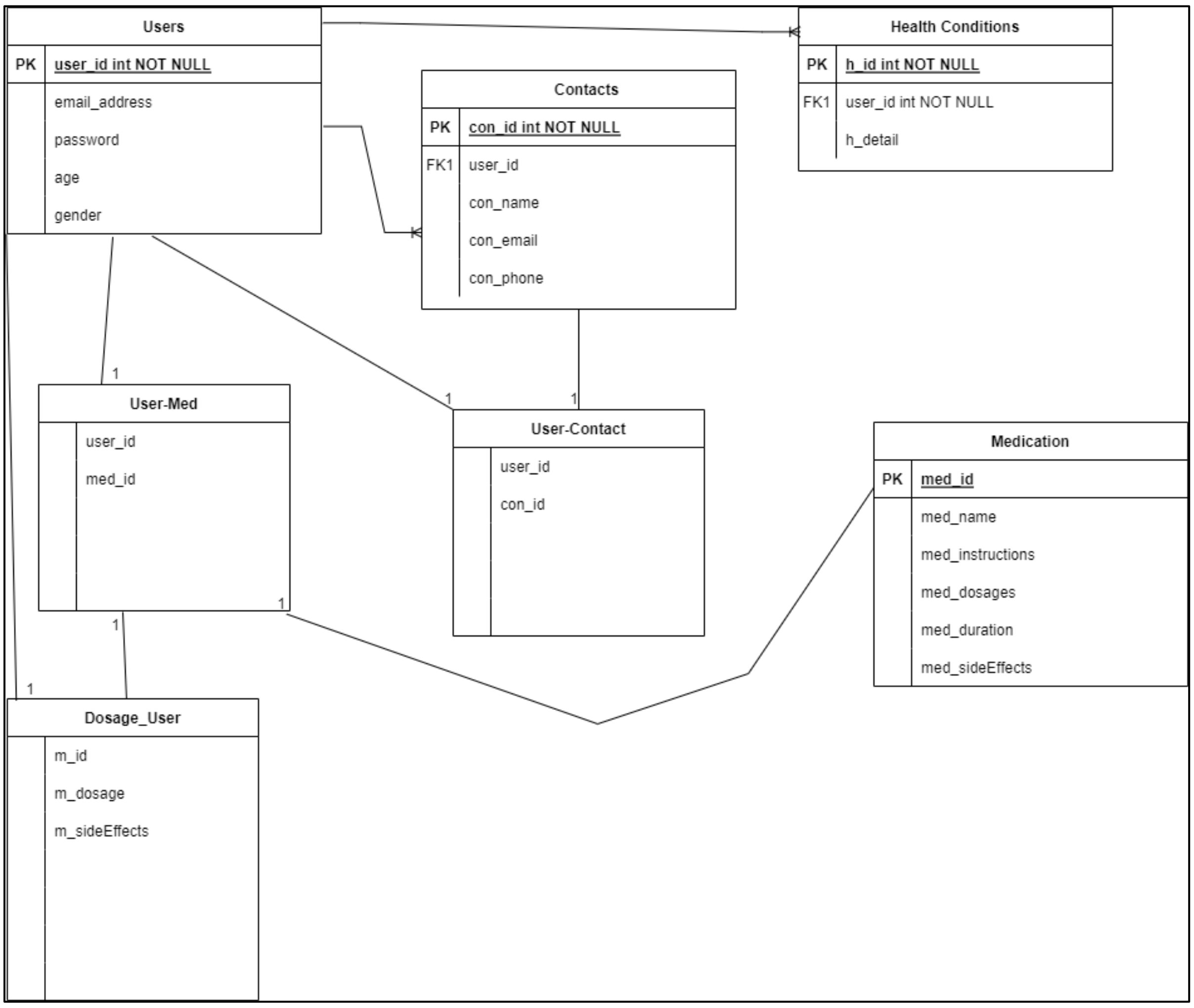


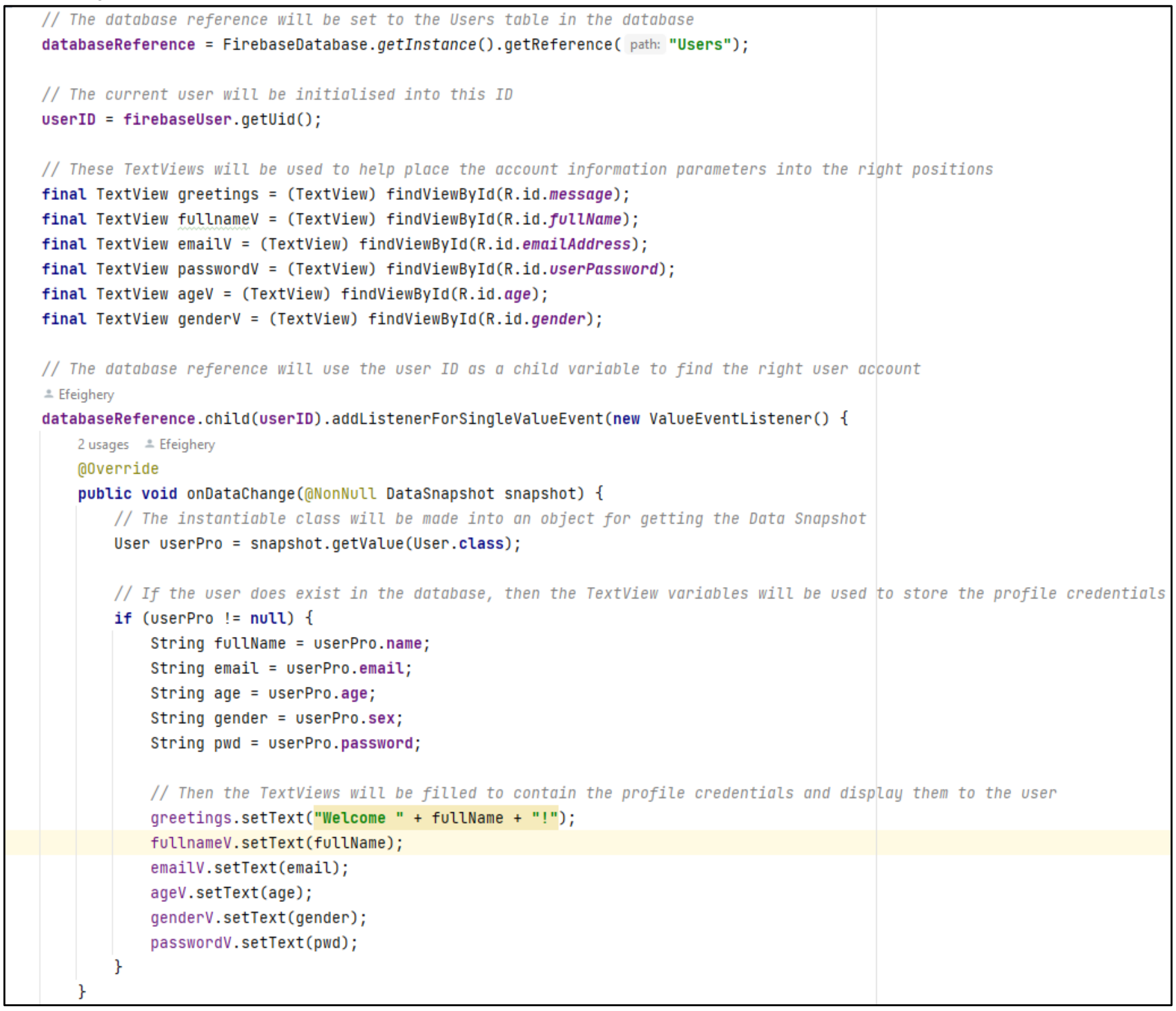
Figure 19: Class diagram for MedsUpApp

2.3 Implementation

As mentioned previously, my initial plan was to use Ionic to develop my application. Unfortunately, during my research and practice, I had some struggles with trying to make the database storage and the text recognition work for me. After a few discussions with my supervisor, I decided to stick to Android Studio and Java as the latter is the one programming language that I have studied the most. In a somewhat similar vein in terms of databases, I initially decided using Realtime Firebase for saving CRUD data but after some difficulty with preventing saved data entries from bleeding in user sessions, I went with FireStore since there was more tutorials that covered saving CRUD data via user IDs.

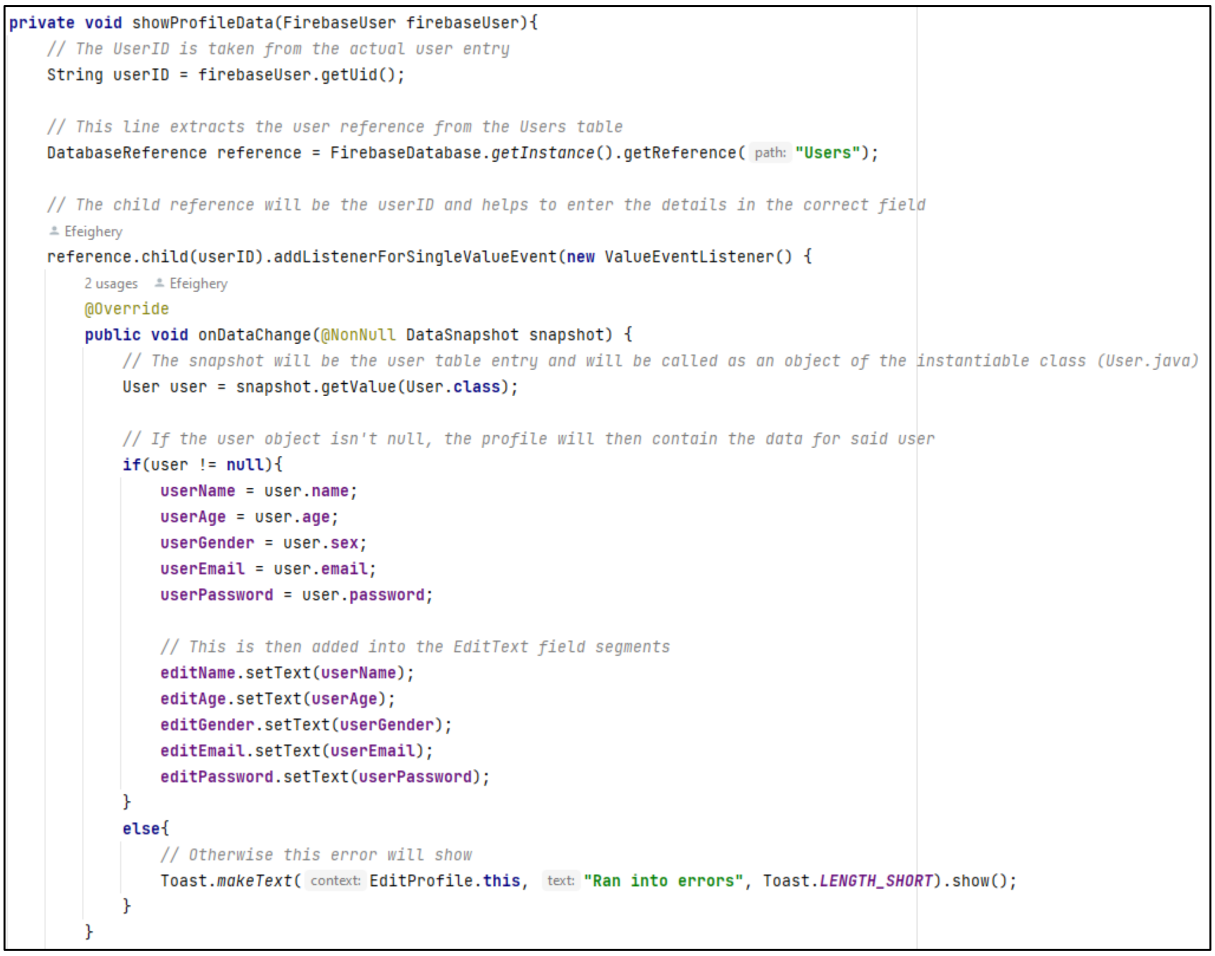
Below are several code snippets with comments that specify what goes on in each code line:

Profile.java:



```
// The database reference will be set to the Users table in the database
databaseReference = FirebaseDatabase.getInstance().getReference( path: "Users");

// The current user will be initialised into this ID
userID = firebaseUser.getUid();

// These TextViews will be used to help place the account information parameters into the right positions
final TextView greetings = (TextView) findViewById(R.id.message);
final TextView fullnameV = (TextView) findViewById(R.id.fullName);
final TextView emailV = (TextView) findViewById(R.id.emailAddress);
final TextView passwordV = (TextView) findViewById(R.id.userPassword);
final TextView ageV = (TextView) findViewById(R.id.age);
final TextView genderV = (TextView) findViewById(R.id.gender);

// The database reference will use the user ID as a child variable to find the right user account
Efeighery
databaseReference.child(userID).addListenerForSingleValueEvent(new ValueEventListener() {
    2 usages  Efeighery
    @Override
    public void onDataChange(@NonNull DataSnapshot snapshot) {
        // The instantiable class will be made into an object for getting the Data Snapshot
        User userPro = snapshot.getValue(User.class);

        // If the user does exist in the database, then the TextView variables will be used to store the profile credentials
        if (userPro != null) {
            String fullName = userPro.name;
            String email = userPro.email;
            String age = userPro.age;
            String gender = userPro.sex;
            String pwd = userPro.password;

            // Then the TextViews will be filled to contain the profile credentials and display them to the user
            greetings.setText("Welcome " + fullName + "!");
            fullnameV.setText(fullName);
            emailV.setText(email);
            ageV.setText(age);
            genderV.setText(gender);
            passwordV.setText(pwd);
        }
    }
```

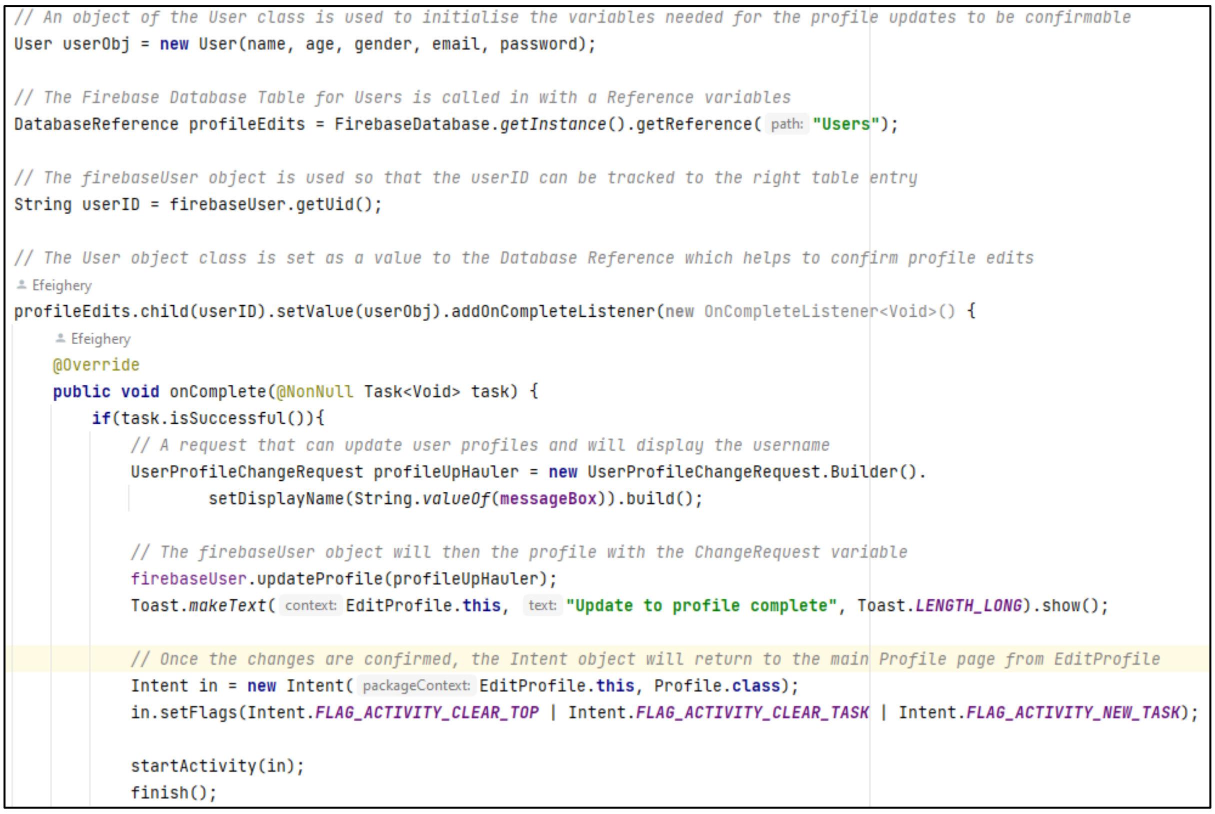
Figure 20: Demonstrating how a profile is created for a user.



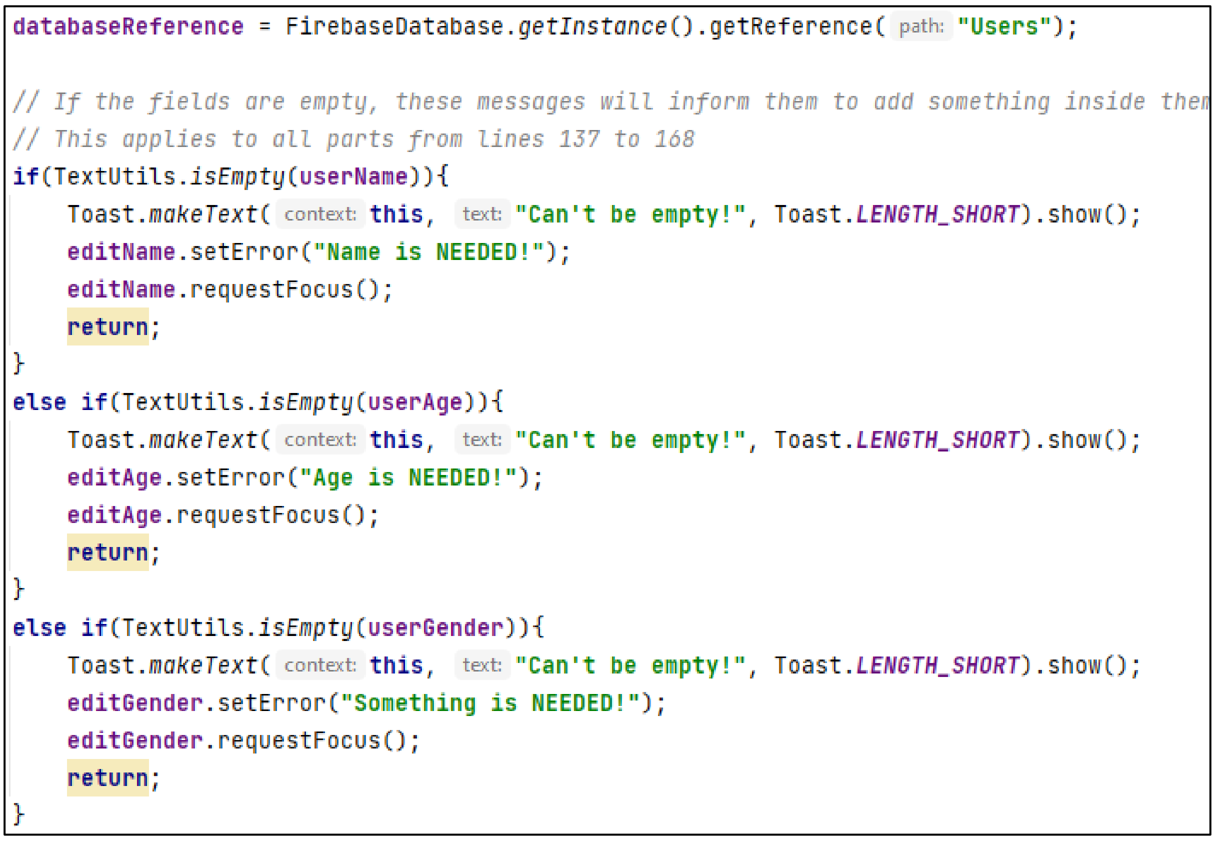
```java
private void showProfileData(FirebaseUser firebaseUser){
    // The UserID is taken from the actual user entry
    String userID = firebaseUser.getUid();

    // This line extracts the user reference from the Users table
    DatabaseReference reference = FirebaseDatabase.getInstance().getReference( path: "Users");

    // The child reference will be the userID and helps to enter the details in the correct field
    Efeighery
    reference.child(userID).addListenerForSingleValueEvent(new ValueEventListener() {
        2 usages   Efeighery
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            // The snapshot will be the user table entry and will be called as an object of the instantiable class (User.java)
            User user = snapshot.getValue(User.class);

            // If the user object isn't null, the profile will then contain the data for said user
            if(user != null){
                userName = user.name;
                userAge = user.age;
                userGender = user.sex;
                userEmail = user.email;
                userPassword = user.password;

                // This is then added into the EditText field segments
                editName.setText(userName);
                editAge.setText(userAge);
                editGender.setText(userGender);
                editEmail.setText(userEmail);
                editPassword.setText(userPassword);
            }
            else{
                // Otherwise this error will show
                Toast.makeText( context: EditProfile.this,  text: "Ran into errors", Toast.LENGTH_SHORT).show();
            }
        }
```

Figure 21: Demonstrating how a profile can be shown to a user.

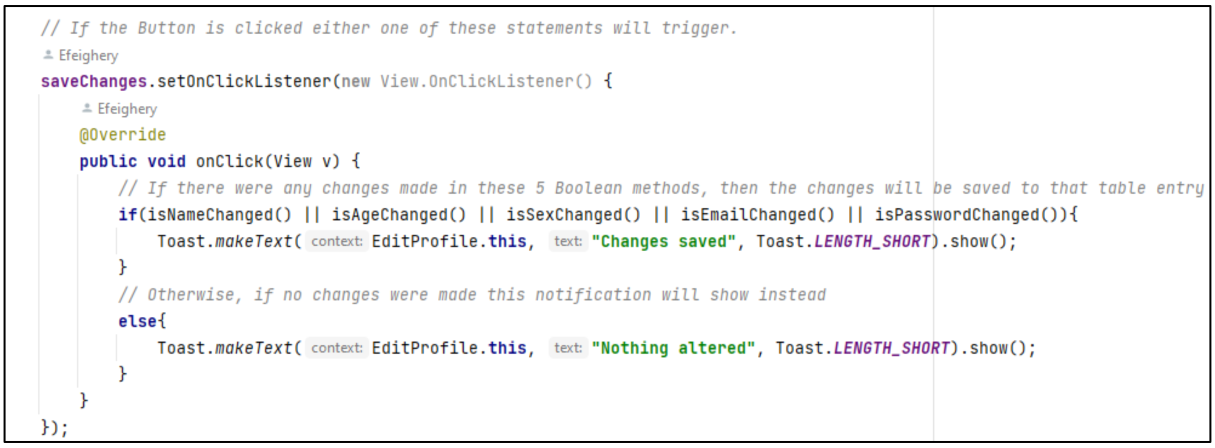
EditProfile.java:



```java
// An object of the User class is used to initialise the variables needed for the profile updates to be confirmable
User userObj = new User(name, age, gender, email, password);

// The Firebase Database Table for Users is called in with a Reference variables
DatabaseReference profileEdits = FirebaseDatabase.getInstance().getReference( path: "Users");

// The firebaseUser object is used so that the userID can be tracked to the right table entry
String userID = firebaseUser.getUid();

// The User object class is set as a value to the Database Reference which helps to confirm profile edits
Efeighery
profileEdits.child(userID).setValue(userObj).addOnCompleteListener(new OnCompleteListener<Void>() {
    Efeighery
    @Override
    public void onComplete(@NonNull Task<Void> task) {
        if(task.isSuccessful()){
            // A request that can update user profiles and will display the username
            UserProfileChangeRequest profileUpHauler = new UserProfileChangeRequest.Builder().
                    setDisplayName(String.valueOf(messageBox)).build();

            // The firebaseUser object will then the profile with the ChangeRequest variable
            firebaseUser.updateProfile(profileUpHauler);
            Toast.makeText( context: EditProfile.this,  text: "Update to profile complete", Toast.LENGTH_LONG).show();

            // Once the changes are confirmed, the Intent object will return to the main Profile page from EditProfile
            Intent in = new Intent( packageContext: EditProfile.this, Profile.class);
            in.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP | Intent.FLAG_ACTIVITY_CLEAR_TASK | Intent.FLAG_ACTIVITY_NEW_TASK);

            startActivity(in);
            finish();
```

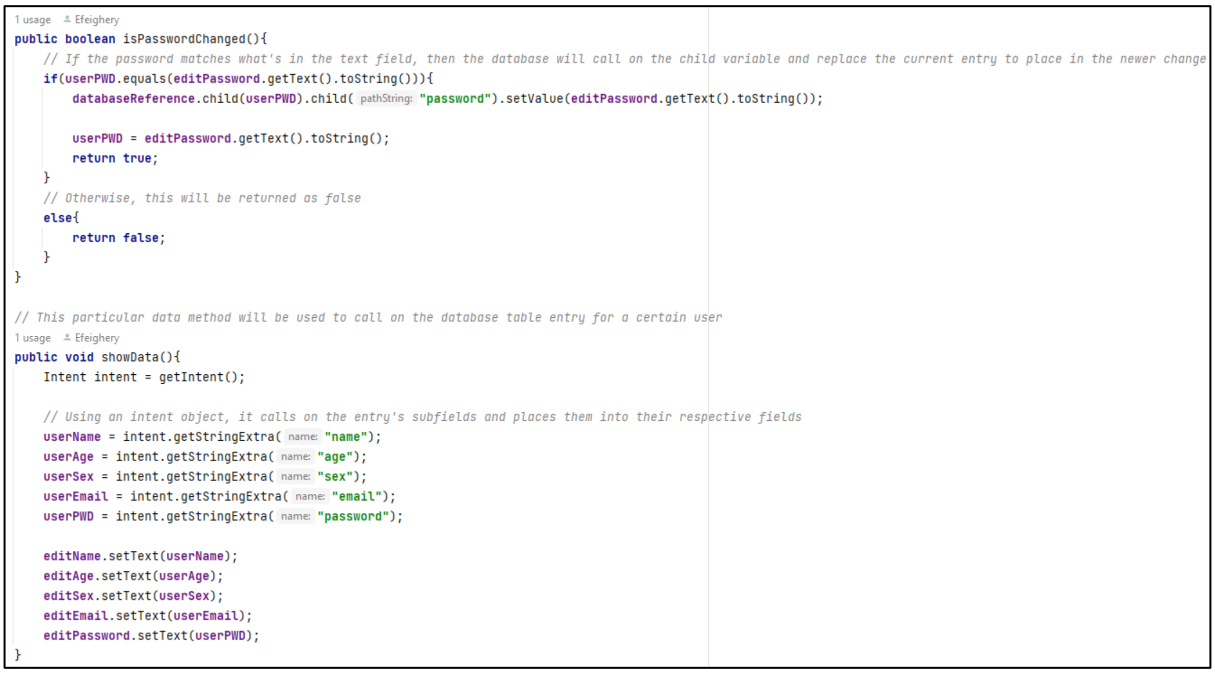


```java
databaseReference = FirebaseDatabase.getInstance().getReference( path: "Users");

// If the fields are empty, these messages will inform them to add something inside the
// This applies to all parts from lines 137 to 168
if(TextUtils.isEmpty(userName)){
    Toast.makeText( context: this,  text: "Can't be empty!", Toast.LENGTH_SHORT).show();
    editName.setError("Name is NEEDED!");
    editName.requestFocus();
    return;
}
else if(TextUtils.isEmpty(userAge)){
    Toast.makeText( context: this,  text: "Can't be empty!", Toast.LENGTH_SHORT).show();
    editAge.setError("Age is NEEDED!");
    editAge.requestFocus();
    return;
}
else if(TextUtils.isEmpty(userGender)){
    Toast.makeText( context: this,  text: "Can't be empty!", Toast.LENGTH_SHORT).show();
    editGender.setError("Something is NEEDED!");
    editGender.requestFocus();
    return;
}
```



```java
// If the Button is clicked either one of these statements will trigger.
 Efeighery
saveChanges.setOnClickListener(new View.OnClickListener() {
     Efeighery
    @Override
    public void onClick(View v) {
        // If there were any changes made in these 5 Boolean methods, then the changes will be saved to that table entry
        if(isNameChanged() || isAgeChanged() || isSexChanged() || isEmailChanged() || isPasswordChanged()){
            Toast.makeText( context: EditProfile.this,  text: "Changes saved", Toast.LENGTH_SHORT).show();
        }
        // Otherwise, if no changes were made this notification will show instead
        else{
            Toast.makeText( context: EditProfile.this,  text: "Nothing altered", Toast.LENGTH_SHORT).show();
        }
    }
});
```

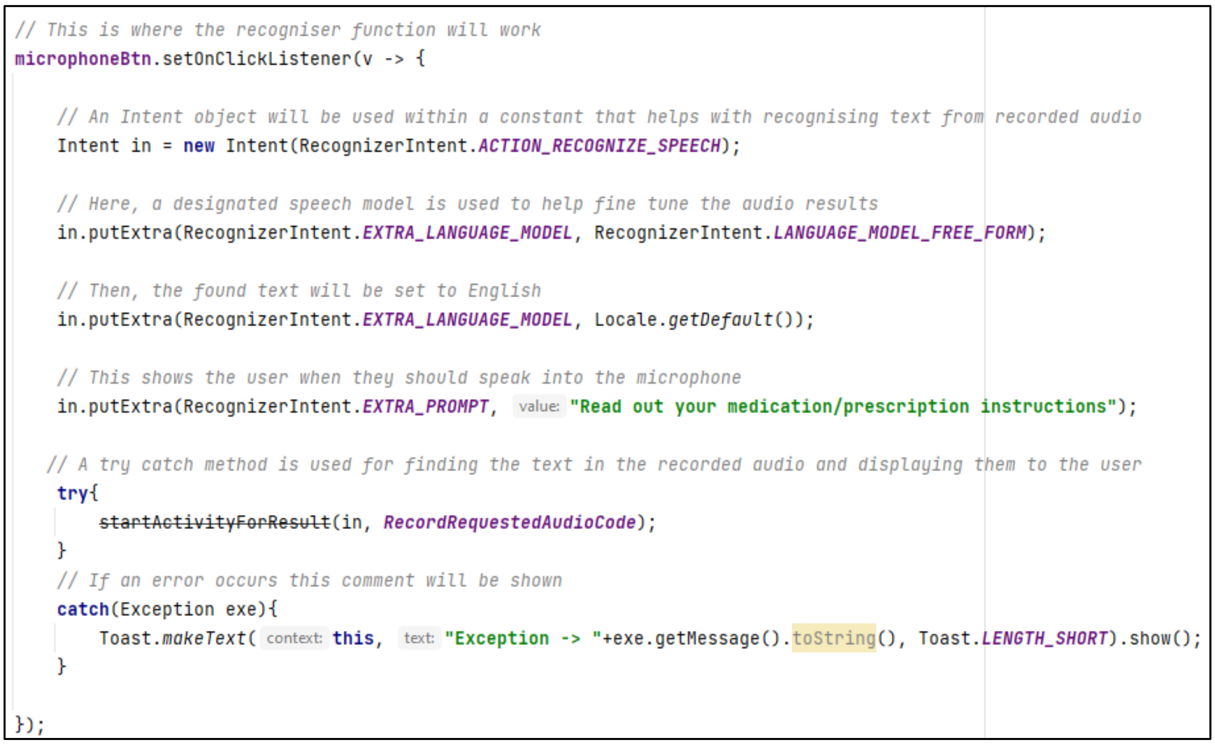
Figure 22: How the Profile changes are confirmed to the user and how they're added to the profile. See screenshots above.



```java
1 usage   Efeighery
public boolean isPasswordChanged(){
    // If the password matches what's in the text field, then the database will call on the child variable and replace the current entry to place in the newer change
    if(userPWD.equals(editPassword.getText().toString())){
        databaseReference.child(userPWD).child( pathString: "password").setValue(editPassword.getText().toString());

        userPWD = editPassword.getText().toString();
        return true;
    }
    // Otherwise, this will be returned as false
    else{
        return false;
    }
}

// This particular data method will be used to call on the database table entry for a certain user
1 usage   Efeighery
public void showData(){
    Intent intent = getIntent();

    // Using an intent object, it calls on the entry's subfields and places them into their respective fields
    userName = intent.getStringExtra( name: "name");
    userAge = intent.getStringExtra( name: "age");
    userSex = intent.getStringExtra( name: "sex");
    userEmail = intent.getStringExtra( name: "email");
    userPWD = intent.getStringExtra( name: "password");

    editName.setText(userName);
    editAge.setText(userAge);
    editSex.setText(userSex);
    editEmail.setText(userEmail);
    editPassword.setText(userPWD);
}
```

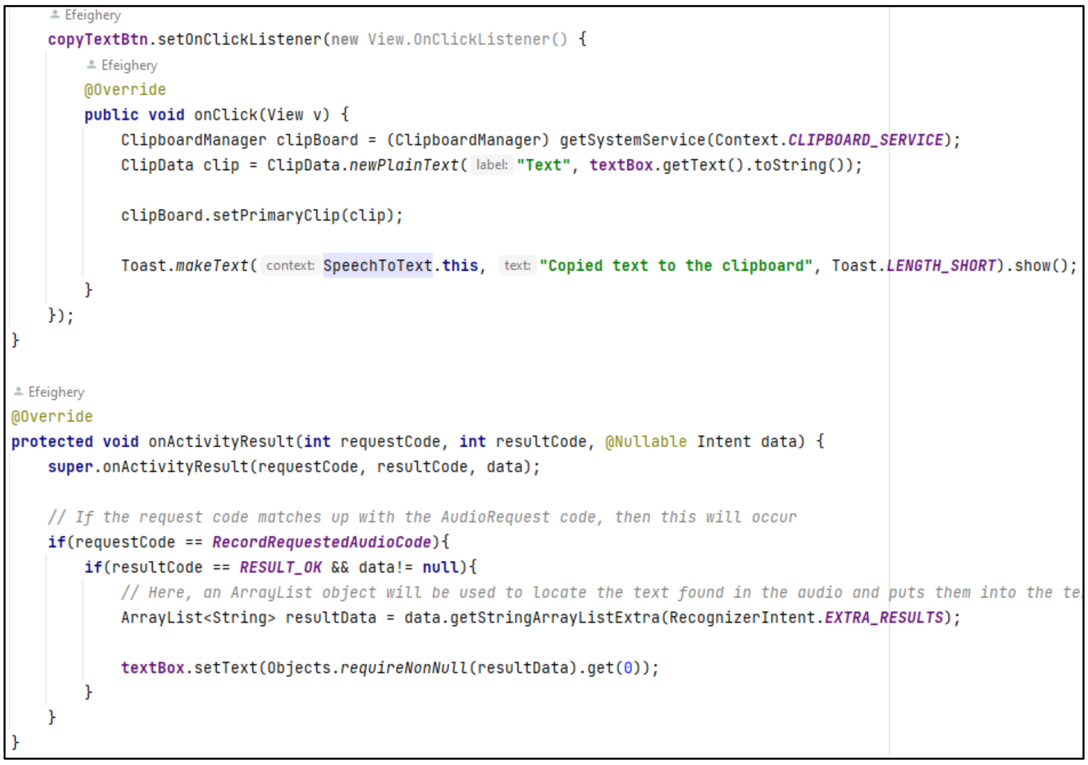
Figure 23: Acquiring data from the user table and how a profile field can be checked for any changes.

SpeechToText.java:



```
// This is where the recogniser function will work
microphoneBtn.setOnClickListener(v -> {

    // An Intent object will be used within a constant that helps with recognising text from recorded audio
    Intent in = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);

    // Here, a designated speech model is used to help fine tune the audio results
    in.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL, RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);

    // Then, the found text will be set to English
    in.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL, Locale.getDefault());

    // This shows the user when they should speak into the microphone
    in.putExtra(RecognizerIntent.EXTRA_PROMPT,  value: "Read out your medication/prescription instructions");

   // A try catch method is used for finding the text in the recorded audio and displaying them to the user
    try{
        ~~startActivityForResult~~(in, RecordRequestedAudioCode);
    }
    // If an error occurs this comment will be shown
    catch(Exception exe){
        Toast.makeText( context: this,  text: "Exception -> "+exe.getMessage().toString(), Toast.LENGTH_SHORT).show();
    }

});
```

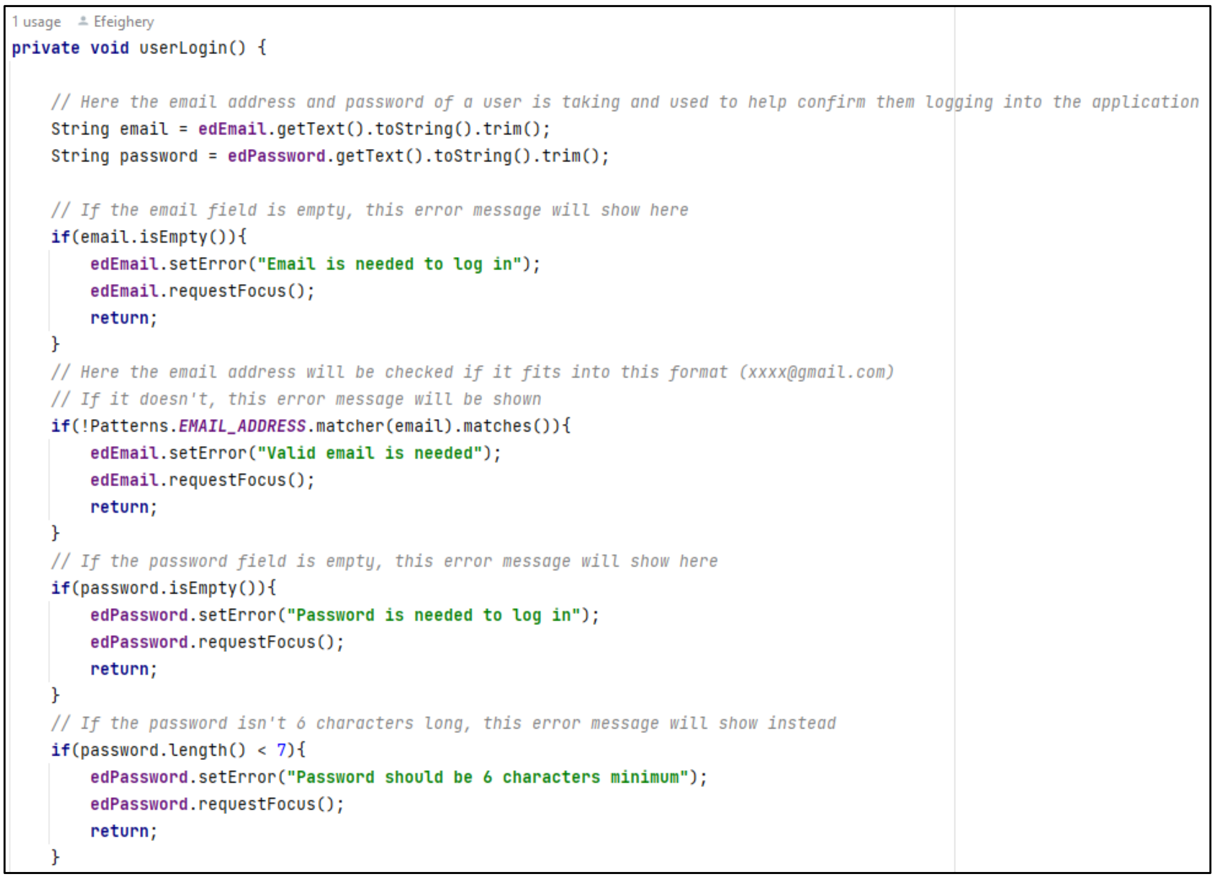
Figure 24: Showing how the microphone functions.



```
    Efeighery
    copyTextBtn.setOnClickListener(new View.OnClickListener() {
        Efeighery
        @Override
        public void onClick(View v) {
            ClipboardManager clipBoard = (ClipboardManager) getSystemService(Context.CLIPBOARD_SERVICE);
            ClipData clip = ClipData.newPlainText( label: "Text", textBox.getText().toString());

            clipBoard.setPrimaryClip(clip);

            Toast.makeText( context: SpeechToText.this,  text: "Copied text to the clipboard", Toast.LENGTH_SHORT).show();
        }
    });
}

Efeighery
@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    // If the request code matches up with the AudioRequest code, then this will occur
    if(requestCode == RecordRequestedAudioCode){
        if(resultCode == RESULT_OK && data!= null){
            // Here, an ArrayList object will be used to locate the text found in the audio and puts them into the te
            ArrayList<String> resultData = data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);

            textBox.setText(Objects.requireNonNull(resultData).get(0));
        }
    }
}
```

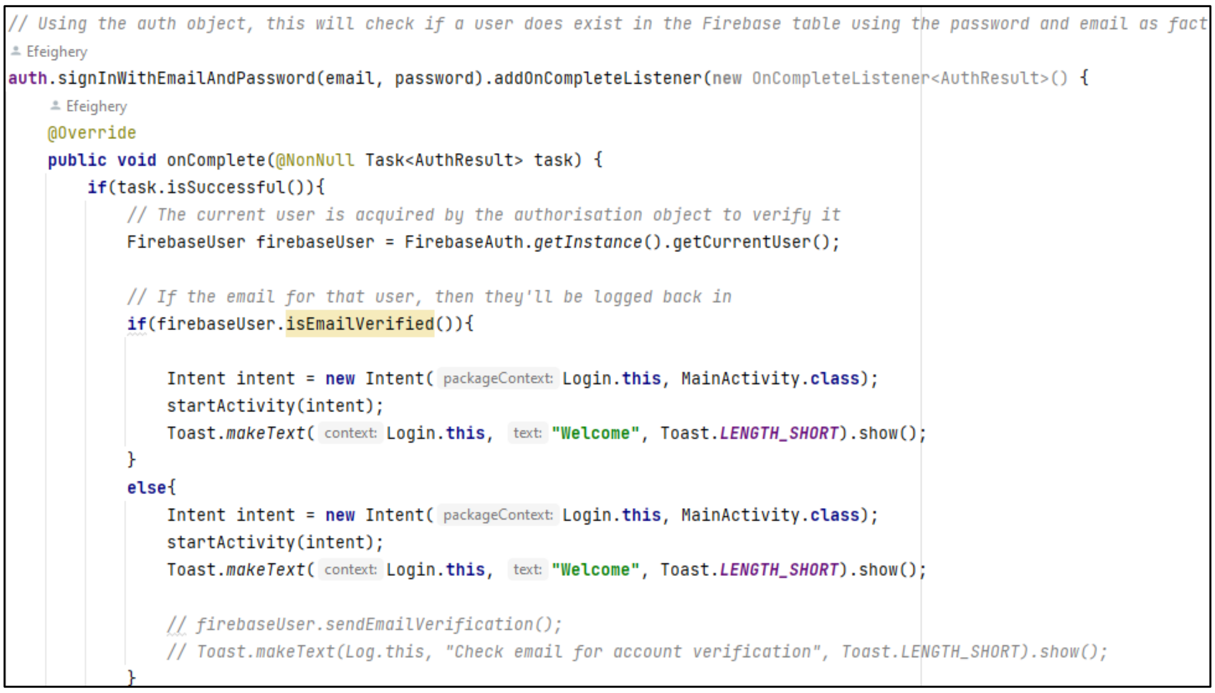
Figure 25: How the found text is produced and the schematics for the copy button

Login.java:



```
1 usage  Efeighery
private void userLogin() {

    // Here the email address and password of a user is taking and used to help confirm them logging into the application
    String email = edEmail.getText().toString().trim();
    String password = edPassword.getText().toString().trim();

    // If the email field is empty, this error message will show here
    if(email.isEmpty()){
        edEmail.setError("Email is needed to log in");
        edEmail.requestFocus();
        return;
    }
    // Here the email address will be checked if it fits into this format (xxxx@gmail.com)
    // If it doesn't, this error message will be shown
    if(!Patterns.EMAIL_ADDRESS.matcher(email).matches()){
        edEmail.setError("Valid email is needed");
        edEmail.requestFocus();
        return;
    }
    // If the password field is empty, this error message will show here
    if(password.isEmpty()){
        edPassword.setError("Password is needed to log in");
        edPassword.requestFocus();
        return;
    }
    // If the password isn't 6 characters long, this error message will show instead
    if(password.length() < 7){
        edPassword.setError("Password should be 6 characters minimum");
        edPassword.requestFocus();
        return;
    }
```

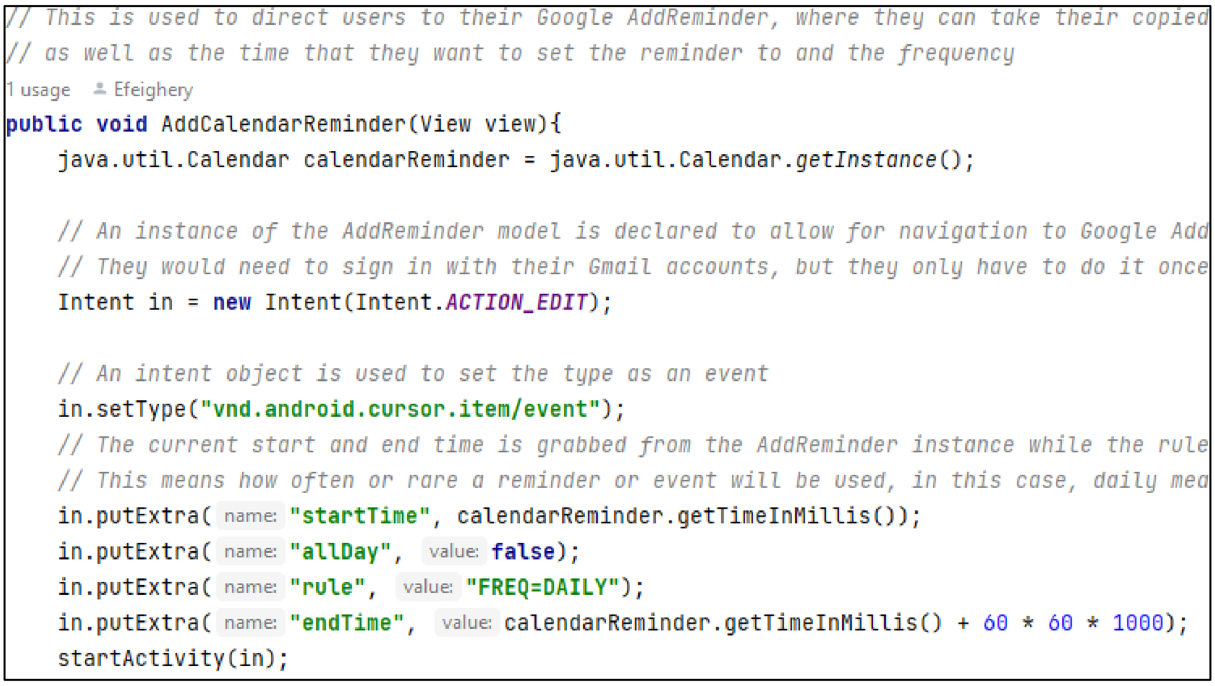
Figure 26: The fact checkers for user login schematics.



```
// Using the auth object, this will check if a user does exist in the Firebase table using the password and email as fact
 Efeighery
auth.signInWithEmailAndPassword(email, password).addOnCompleteListener(new OnCompleteListener<AuthResult>() {
     Efeighery
    @Override
    public void onComplete(@NonNull Task<AuthResult> task) {
        if(task.isSuccessful()){
            // The current user is acquired by the authorisation object to verify it
            FirebaseUser firebaseUser = FirebaseAuth.getInstance().getCurrentUser();

            // If the email for that user, then they'll be logged back in
            if(firebaseUser.isEmailVerified()){

                Intent intent = new Intent( packageContext: Login.this, MainActivity.class);
                startActivity(intent);
                Toast.makeText( context: Login.this,  text: "Welcome", Toast.LENGTH_SHORT).show();
            }
            else{
                Intent intent = new Intent( packageContext: Login.this, MainActivity.class);
                startActivity(intent);
                Toast.makeText( context: Login.this,  text: "Welcome", Toast.LENGTH_SHORT).show();

                // firebaseUser.sendEmailVerification();
                // Toast.makeText(Log.this, "Check email for account verification", Toast.LENGTH_SHORT).show();
            }
```

Figure 27: Adding authorisation for a user logging in.

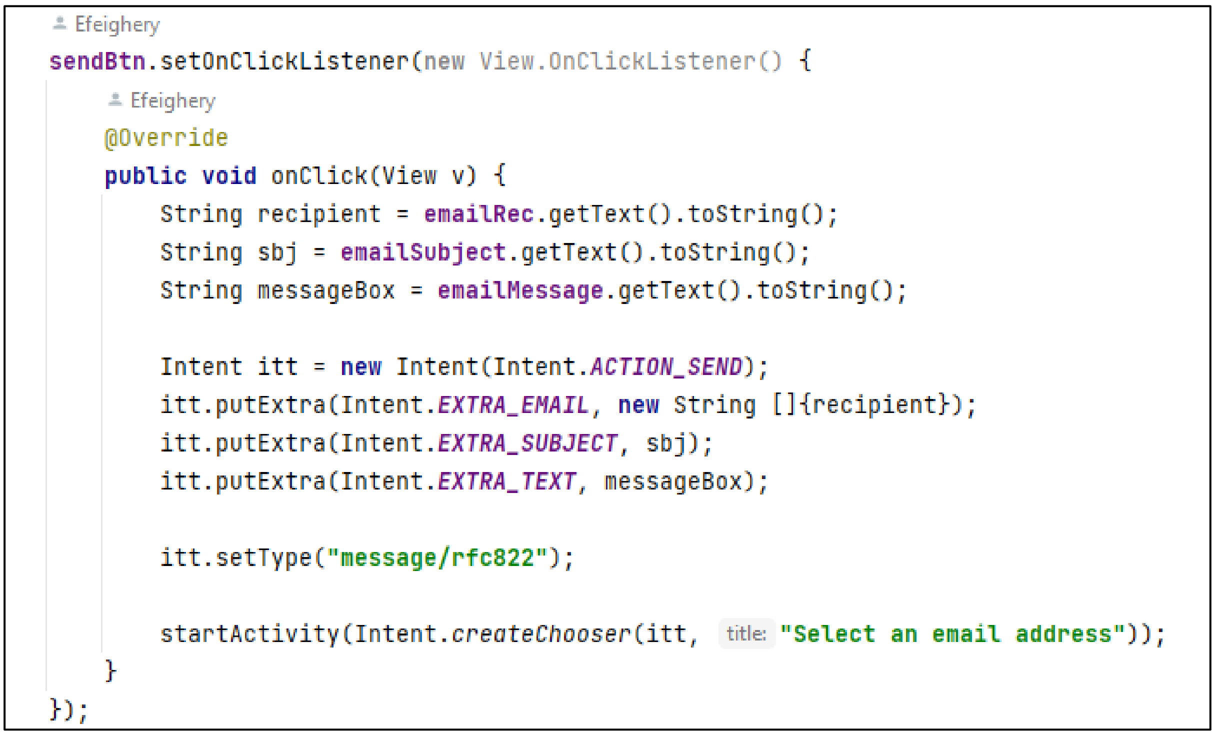
AddReminder.java:



```
// This is used to direct users to their Google AddReminder, where they can take their copied
// as well as the time that they want to set the reminder to and the frequency
1 usage   Efeighery
public void AddCalendarReminder(View view){
    java.util.Calendar calendarReminder = java.util.Calendar.getInstance();

    // An instance of the AddReminder model is declared to allow for navigation to Google Add
    // They would need to sign in with their Gmail accounts, but they only have to do it once
    Intent in = new Intent(Intent.ACTION_EDIT);

    // An intent object is used to set the type as an event
    in.setType("vnd.android.cursor.item/event");
    // The current start and end time is grabbed from the AddReminder instance while the rule
    // This means how often or rare a reminder or event will be used, in this case, daily mea
    in.putExtra( name: "startTime", calendarReminder.getTimeInMillis());
    in.putExtra( name: "allDay",  value: false);
    in.putExtra( name: "rule",  value: "FREQ=DAILY");
    in.putExtra( name: "endTime",  value: calendarReminder.getTimeInMillis() + 60 * 60 * 1000);
    startActivity(in);
```

Figure 28: Parameters for a Google Calendar event notification

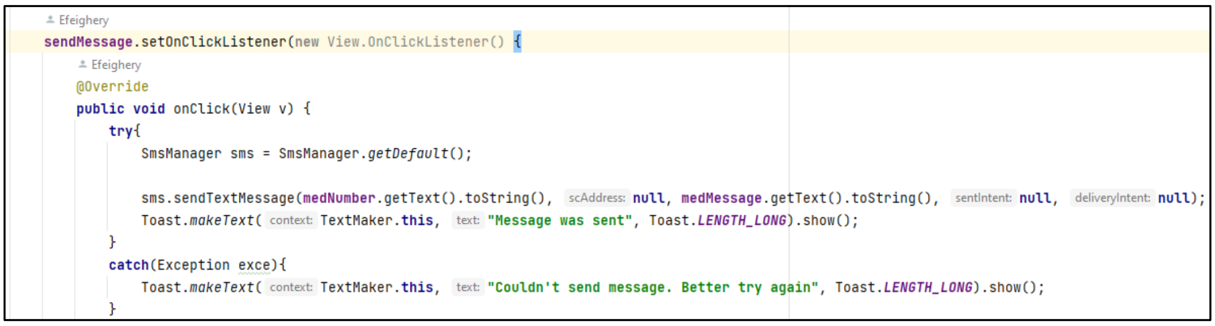
EmailMaker.java:



```
   Efeighery
   sendBtn.setOnClickListener(new View.OnClickListener() {
       Efeighery
       @Override
       public void onClick(View v) {
           String recipient = emailRec.getText().toString();
           String sbj = emailSubject.getText().toString();
           String messageBox = emailMessage.getText().toString();

           Intent itt = new Intent(Intent.ACTION_SEND);
           itt.putExtra(Intent.EXTRA_EMAIL, new String []{recipient});
           itt.putExtra(Intent.EXTRA_SUBJECT, sbj);
           itt.putExtra(Intent.EXTRA_TEXT, messageBox);

           itt.setType("message/rfc822");

           startActivity(Intent.createChooser(itt,  title: "Select an email address"));
       }
   });
```

Figure 29: Showcasing how a formatted email is set up in Gmail.

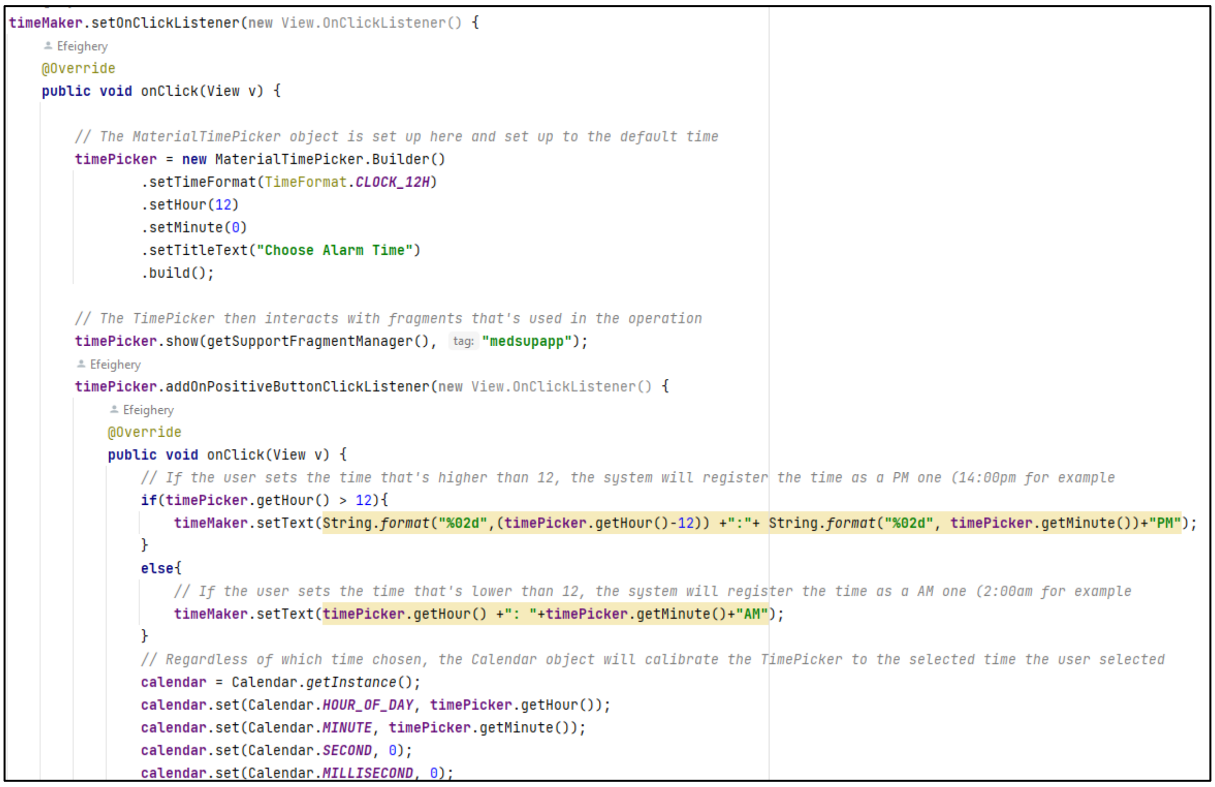
TextMaker.java:



```
    Efeighery
    sendMessage.setOnClickListener(new View.OnClickListener() {
        Efeighery
        @Override
        public void onClick(View v) {
            try{
                SmsManager sms = SmsManager.getDefault();

                sms.sendTextMessage(medNumber.getText().toString(),  scAddress: null, medMessage.getText().toString(),  sentIntent: null,  deliveryIntent: null);
                Toast.makeText( context: TextMaker.this,  text: "Message was sent", Toast.LENGTH_LONG).show();
            }
            catch(Exception exce){
                Toast.makeText( context: TextMaker.this,  text: "Couldn't send message. Better try again", Toast.LENGTH_LONG).show();
            }
```

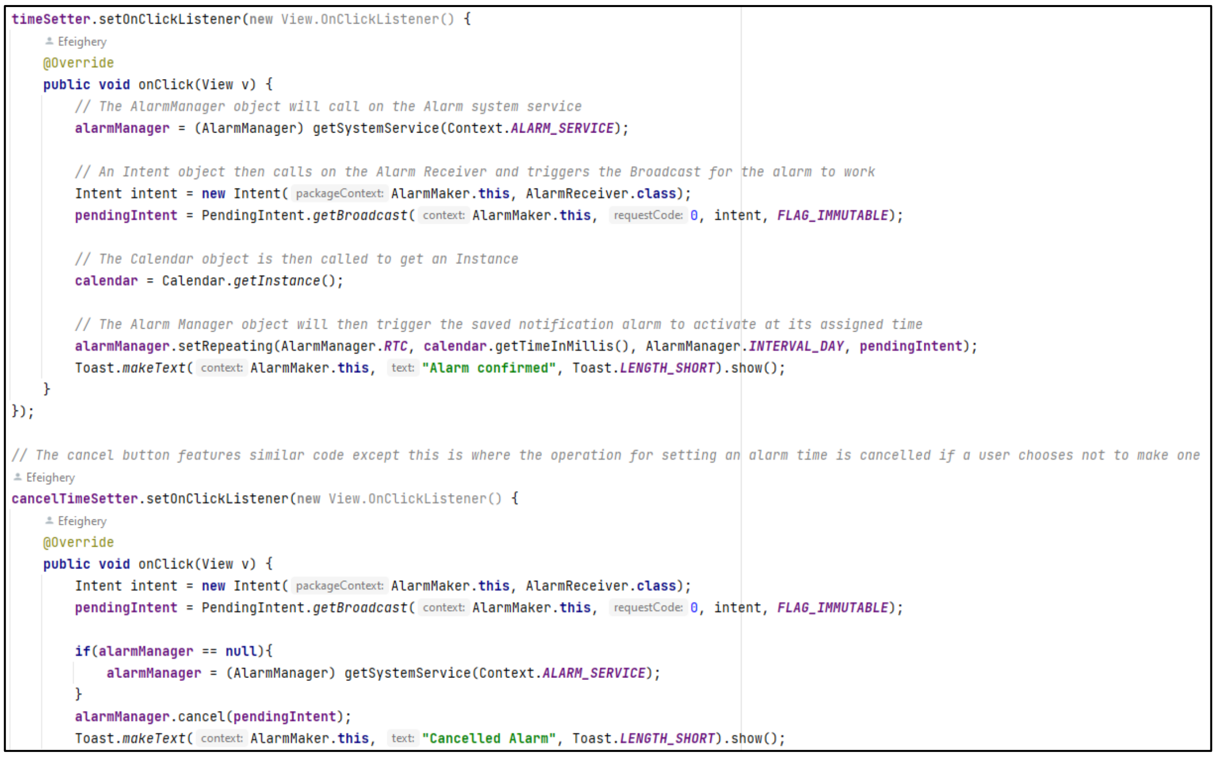
Figure 30: The parameters for sending a text message to a recipient.

NotificationCreator.java:



```
timeMaker.setOnClickListener(new View.OnClickListener() {
    Efeighery
    @Override
    public void onClick(View v) {

        // The MaterialTimePicker object is set up here and set up to the default time
        timePicker = new MaterialTimePicker.Builder()
                .setTimeFormat(TimeFormat.CLOCK_12H)
                .setHour(12)
                .setMinute(0)
                .setTitleText("Choose Alarm Time")
                .build();

        // The TimePicker then interacts with fragments that's used in the operation
        timePicker.show(getSupportFragmentManager(),  tag: "medsupapp");
        Efeighery
        timePicker.addOnPositiveButtonClickListener(new View.OnClickListener() {
            Efeighery
            @Override
            public void onClick(View v) {
                // If the user sets the time that's higher than 12, the system will register the time as a PM one (14:00pm for example
                if(timePicker.getHour() > 12){
                    timeMaker.setText(String.format("%02d",(timePicker.getHour()-12)) +":"+ String.format("%02d", timePicker.getMinute())+"PM");
                }
                else{
                    // If the user sets the time that's lower than 12, the system will register the time as a AM one (2:00am for example
                    timeMaker.setText(timePicker.getHour() +": "+timePicker.getMinute()+"AM");
                }
                // Regardless of which time chosen, the Calendar object will calibrate the TimePicker to the selected time the user selected
                calendar = Calendar.getInstance();
                calendar.set(Calendar.HOUR_OF_DAY, timePicker.getHour());
                calendar.set(Calendar.MINUTE, timePicker.getMinute());
                calendar.set(Calendar.SECOND, 0);
                calendar.set(Calendar.MILLISECOND, 0);
```

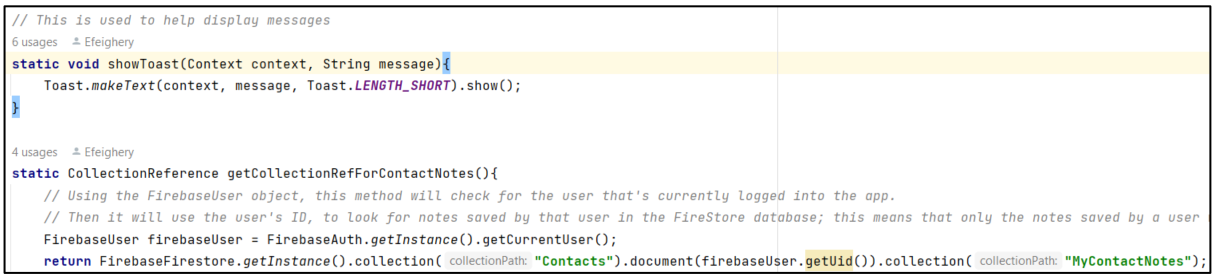
Figure 31: The methods for setting a reminder time, via the Calendar instance.



```
timeSetter.setOnClickListener(new View.OnClickListener() {
    Efeighery
    @Override
    public void onClick(View v) {
        // The AlarmManager object will call on the Alarm system service
        alarmManager = (AlarmManager) getSystemService(Context.ALARM_SERVICE);

        // An Intent object then calls on the Alarm Receiver and triggers the Broadcast for the alarm to work
        Intent intent = new Intent( packageContext: AlarmMaker.this, AlarmReceiver.class);
        pendingIntent = PendingIntent.getBroadcast( context: AlarmMaker.this,  requestCode: 0, intent, FLAG_IMMUTABLE);

        // The Calendar object is then called to get an Instance
        calendar = Calendar.getInstance();

        // The Alarm Manager object will then trigger the saved notification alarm to activate at its assigned time
        alarmManager.setRepeating(AlarmManager.RTC, calendar.getTimeInMillis(), AlarmManager.INTERVAL_DAY, pendingIntent);
        Toast.makeText( context: AlarmMaker.this,  text: "Alarm confirmed", Toast.LENGTH_SHORT).show();
    }
});

// The cancel button features similar code except this is where the operation for setting an alarm time is cancelled if a user chooses not to make one
Efeighery
cancelTimeSetter.setOnClickListener(new View.OnClickListener() {
    Efeighery
    @Override
    public void onClick(View v) {
        Intent intent = new Intent( packageContext: AlarmMaker.this, AlarmReceiver.class);
        pendingIntent = PendingIntent.getBroadcast( context: AlarmMaker.this,  requestCode: 0, intent, FLAG_IMMUTABLE);

        if(alarmManager == null){
            alarmManager = (AlarmManager) getSystemService(Context.ALARM_SERVICE);
        }
        alarmManager.cancel(pendingIntent);
        Toast.makeText( context: AlarmMaker.this,  text: "Cancelled Alarm", Toast.LENGTH_SHORT).show();
```

Figure 32: Methods for confirming and cancelling a newly created alarm respectively.

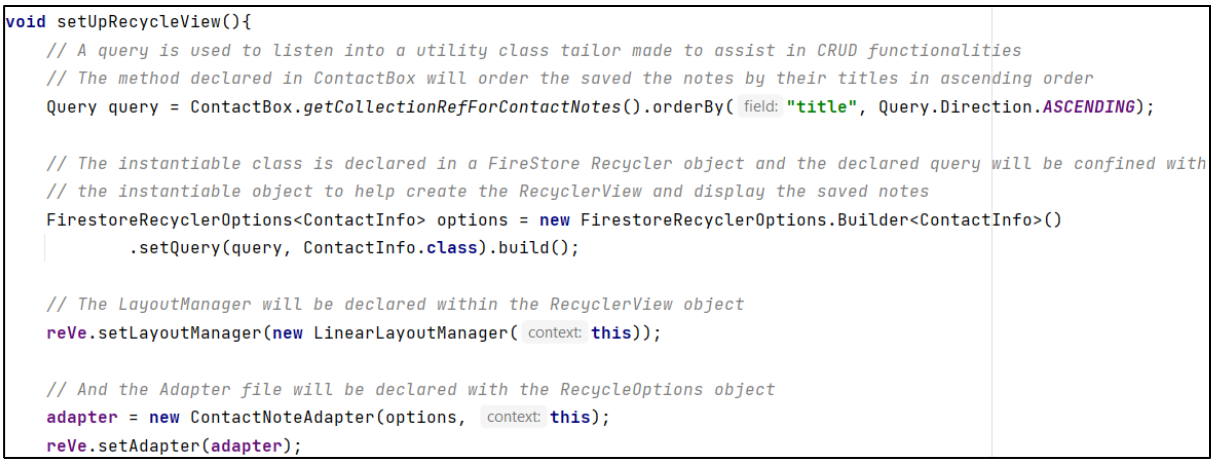
ContactBox.java:



```
// This is used to help display messages
6 usages  Efeighery
static void showToast(Context context, String message){
    Toast.makeText(context, message, Toast.LENGTH_SHORT).show();
}

4 usages  Efeighery
static CollectionReference getCollectionRefForContactNotes(){
    // Using the FirebaseUser object, this method will check for the user that's currently logged into the app.
    // Then it will use the user's ID, to look for notes saved by that user in the FireStore database; this means that only the notes saved by a user
    FirebaseUser firebaseUser = FirebaseAuth.getInstance().getCurrentUser();
    return FirebaseFirestore.getInstance().collection( collectionPath: "Contacts").document(firebaseUser.getUid()).collection( collectionPath: "MyContactNotes");
```

Figure 33: The instantiable class that allows the application to collect all of the saved contact notes in their designated database table.

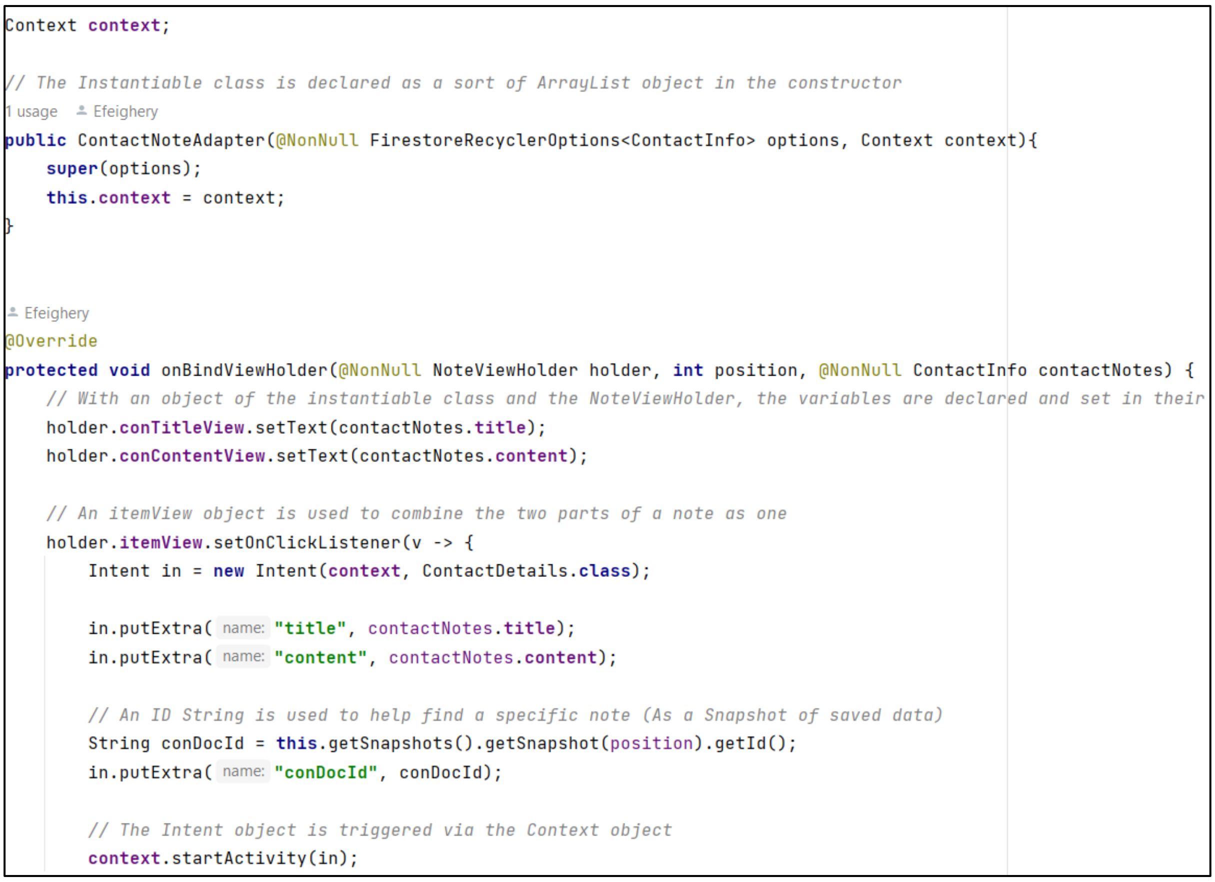
ContactNotes.java:



```
void setUpRecycleView(){
    // A query is used to listen into a utility class tailor made to assist in CRUD functionalities
    // The method declared in ContactBox will order the saved the notes by their titles in ascending order
    Query query = ContactBox.getCollectionRefForContactNotes().orderBy( field: "title", Query.Direction.ASCENDING);

    // The instantiable class is declared in a FireStore Recycler object and the declared query will be confined with
    // the instantiable object to help create the RecyclerView and display the saved notes
    FirestoreRecyclerOptions<ContactInfo> options = new FirestoreRecyclerOptions.Builder<ContactInfo>()
            .setQuery(query, ContactInfo.class).build();

    // The LayoutManager will be declared within the RecyclerView object
    reVe.setLayoutManager(new LinearLayoutManager( context: this));

    // And the Adapter file will be declared with the RecycleOptions object
    adapter = new ContactNoteAdapter(options,  context: this);
    reVe.setAdapter(adapter);
```

Figure 34: Gathering the saved contact notes from FireStore and setting up them in a list view.

ContactNoteAdapter.java:



```
Context context;

// The Instantiable class is declared as a sort of ArrayList object in the constructor
1 usage  Efeighery
public ContactNoteAdapter(@NonNull FirestoreRecyclerOptions<ContactInfo> options, Context context){
    super(options);
    this.context = context;
}


 Efeighery
@Override
protected void onBindViewHolder(@NonNull NoteViewHolder holder, int position, @NonNull ContactInfo contactNotes) {
    // With an object of the instantiable class and the NoteViewHolder, the variables are declared and set in their
    holder.conTitleView.setText(contactNotes.title);
    holder.conContentView.setText(contactNotes.content);

    // An itemView object is used to combine the two parts of a note as one
    holder.itemView.setOnClickListener(v -> {
        Intent in = new Intent(context, ContactDetails.class);

        in.putExtra( name: "title", contactNotes.title);
        in.putExtra( name: "content", contactNotes.content);

        // An ID String is used to help find a specific note (As a Snapshot of saved data)
        String conDocId = this.getSnapshots().getSnapshot(position).getId();
        in.putExtra( name: "conDocId", conDocId);

        // The Intent object is triggered via the Context object
        context.startActivity(in);
```

Figure 35: Setting the ViewHolder for the Adapter object for Contact Notes with an object of the instantiable class.

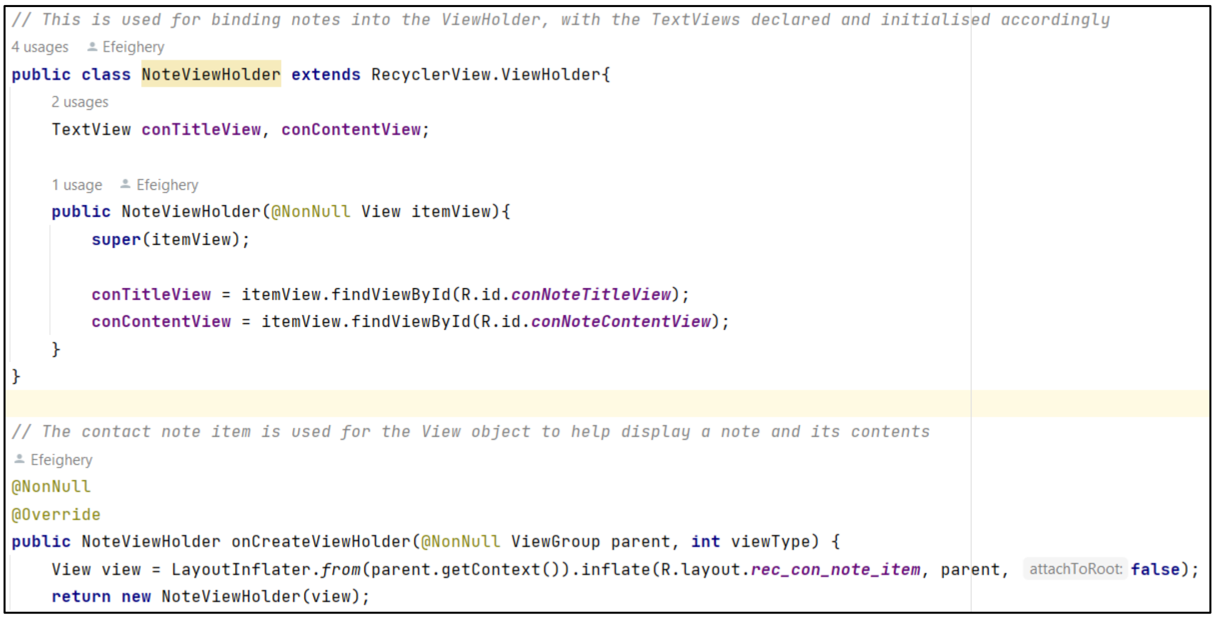

```
// This is used for binding notes into the ViewHolder, with the TextViews declared and initialised accordingly
4 usages  Efeighery
public class NoteViewHolder extends RecyclerView.ViewHolder{
    2 usages
    TextView conTitleView, conContentView;

    1 usage  Efeighery
    public NoteViewHolder(@NonNull View itemView){
        super(itemView);

        conTitleView = itemView.findViewById(R.id.conNoteTitleView);
        conContentView = itemView.findViewById(R.id.conNoteContentView);
    }
}

// The contact note item is used for the View object to help display a note and its contents
Efeighery
@NonNull
@Override
public NoteViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.rec_con_note_item, parent,  attachToRoot: false);
    return new NoteViewHolder(view);
```

Figure 36: The ViewHolder binder and initialising the note item for Contact notes.

ContactDetails.java:

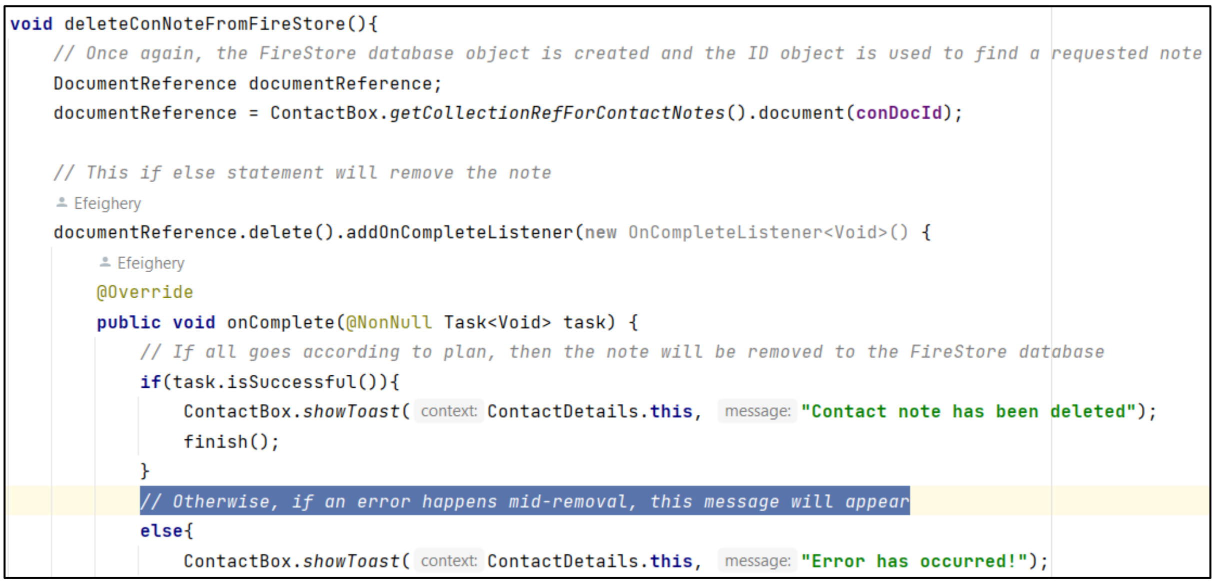
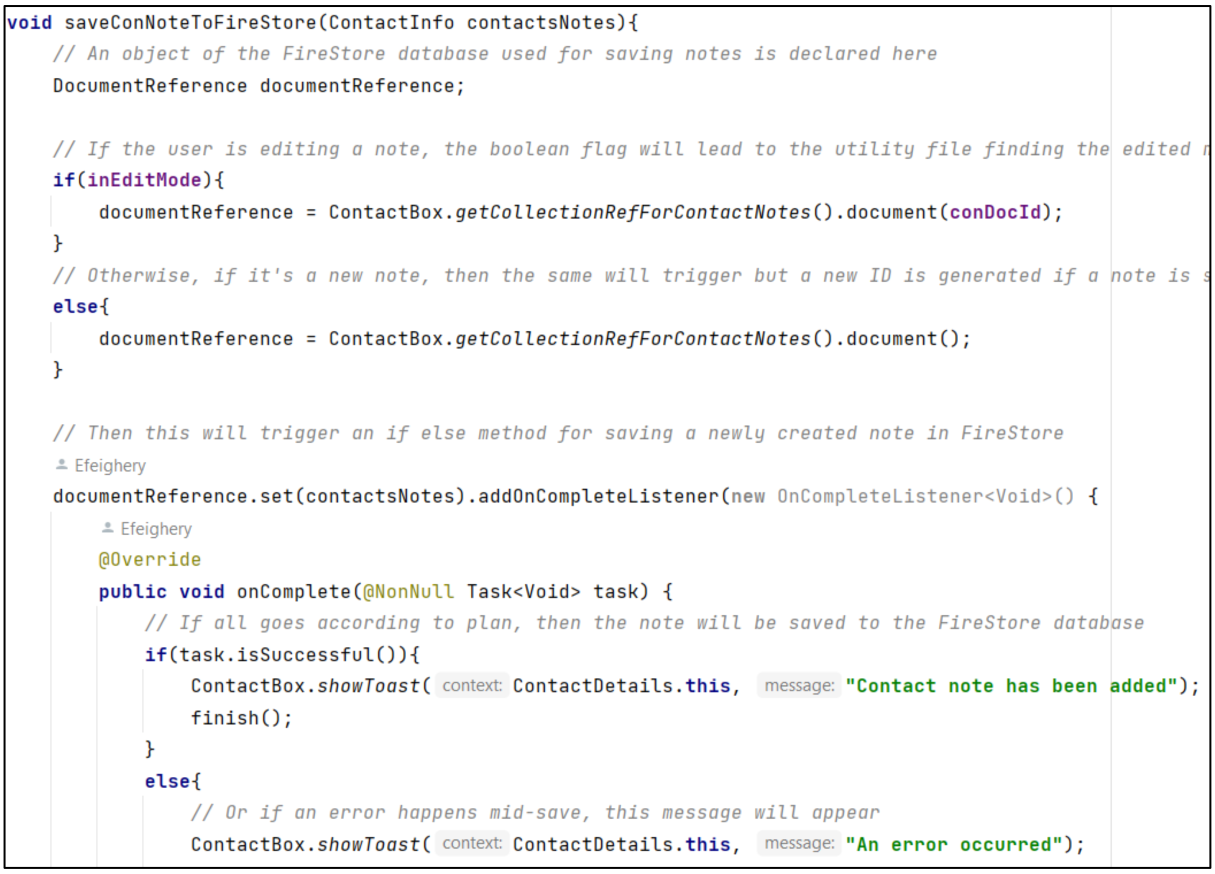
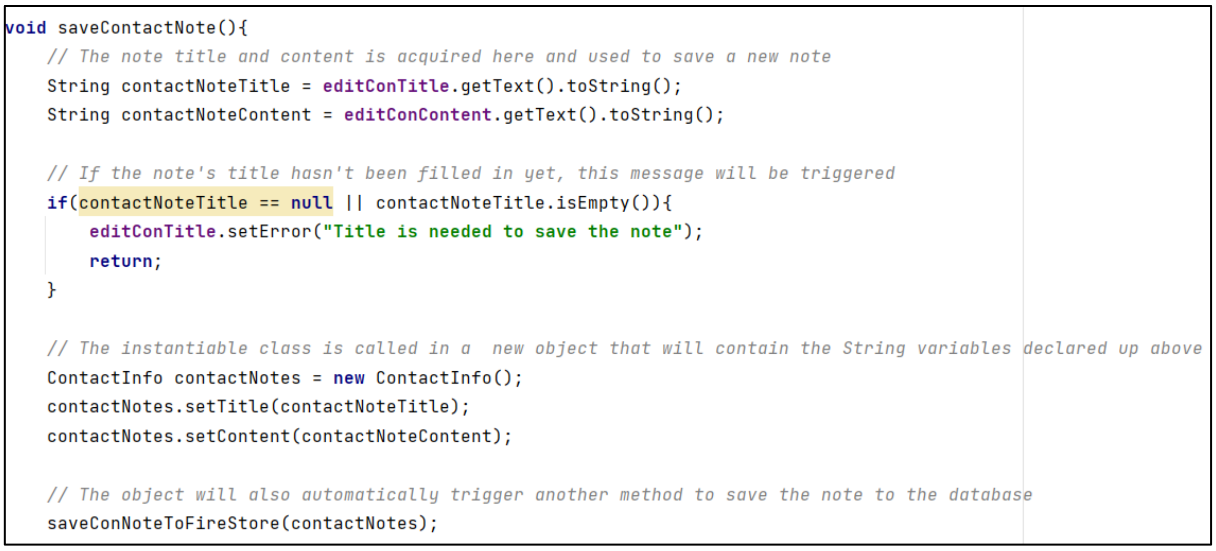

```
void deleteConNoteFromFireStore(){
    // Once again, the FireStore database object is created and the ID object is used to find a requested note
    DocumentReference documentReference;
    documentReference = ContactBox.getCollectionRefForContactNotes().document(conDocId);

    // This if else statement will remove the note
    Efeighery
    documentReference.delete().addOnCompleteListener(new OnCompleteListener<Void>() {
        Efeighery
        @Override
        public void onComplete(@NonNull Task<Void> task) {
            // If all goes according to plan, then the note will be removed to the FireStore database
            if(task.isSuccessful()){
                ContactBox.showToast( context: ContactDetails.this,  message: "Contact note has been deleted");
                finish();
            }
            // Otherwise, if an error happens mid-removal, this message will appear
            else{
                ContactBox.showToast( context: ContactDetails.this,  message: "Error has occurred!");
```

Figure 37: The method where a user deletes a Contact note from its FireStore Database table.



```java
void saveConNoteToFireStore(ContactInfo contactsNotes){
    // An object of the FireStore database used for saving notes is declared here
    DocumentReference documentReference;

    // If the user is editing a note, the boolean flag will lead to the utility file finding the edited r
    if(inEditMode){
        documentReference = ContactBox.getCollectionRefForContactNotes().document(conDocId);
    }
    // Otherwise, if it's a new note, then the same will trigger but a new ID is generated if a note is s
    else{
        documentReference = ContactBox.getCollectionRefForContactNotes().document();
    }

    // Then this will trigger an if else method for saving a newly created note in FireStore
    Efeighery
    documentReference.set(contactsNotes).addOnCompleteListener(new OnCompleteListener<Void>() {
        Efeighery
        @Override
        public void onComplete(@NonNull Task<Void> task) {
            // If all goes according to plan, then the note will be saved to the FireStore database
            if(task.isSuccessful()){
                ContactBox.showToast( context: ContactDetails.this,  message: "Contact note has been added");
                finish();
            }
            else{
                // Or if an error happens mid-save, this message will appear
                ContactBox.showToast( context: ContactDetails.this,  message: "An error occurred");
```

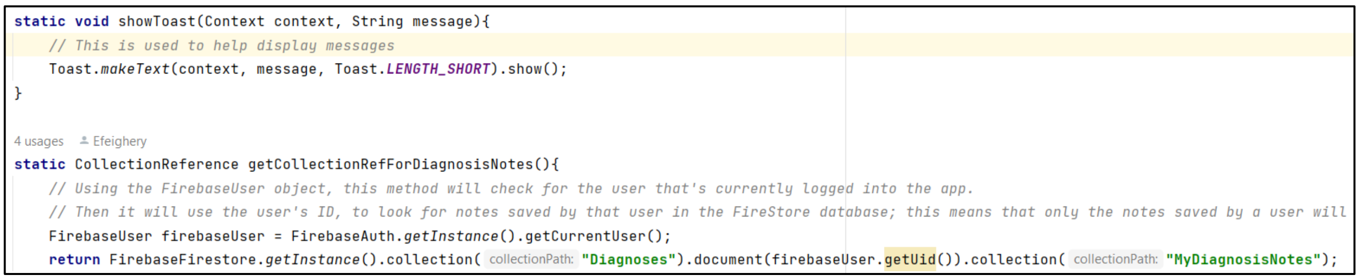
Figure 38: The method where a user saves a Contact note to the FireStore Database table using a designated ID.



```java
void saveContactNote(){
    // The note title and content is acquired here and used to save a new note
    String contactNoteTitle = editConTitle.getText().toString();
    String contactNoteContent = editConContent.getText().toString();

    // If the note's title hasn't been filled in yet, this message will be triggered
    if(contactNoteTitle == null || contactNoteTitle.isEmpty()){
        editConTitle.setError("Title is needed to save the note");
        return;
    }

    // The instantiable class is called in a  new object that will contain the String variables declared up above
    ContactInfo contactNotes = new ContactInfo();
    contactNotes.setTitle(contactNoteTitle);
    contactNotes.setContent(contactNoteContent);

    // The object will also automatically trigger another method to save the note to the database
    saveConNoteToFireStore(contactNotes);
```

Figure 39: The first save note method which uses an object of the instantiable class.

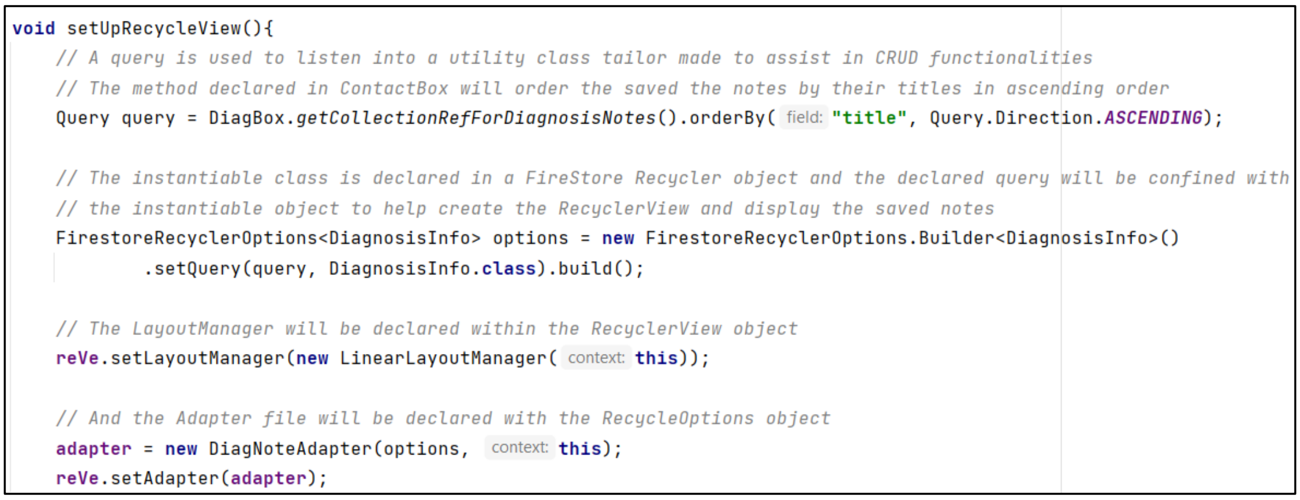
DiagBox.java:



```
static void showToast(Context context, String message){
    // This is used to help display messages
    Toast.makeText(context, message, Toast.LENGTH_SHORT).show();
}

4 usages  Efeighery
static CollectionReference getCollectionRefForDiagnosisNotes(){
    // Using the FirebaseUser object, this method will check for the user that's currently logged into the app.
    // Then it will use the user's ID, to look for notes saved by that user in the FireStore database; this means that only the notes saved by a user will
    FirebaseUser firebaseUser = FirebaseAuth.getInstance().getCurrentUser();
    return FirebaseFirestore.getInstance().collection( collectionPath: "Diagnoses").document(firebaseUser.getUid()).collection( collectionPath: "MyDiagnosisNotes");
```

Figure 40: The instantiable class that allows the application to collect all of the saved diagnosis notes in their designated database table.

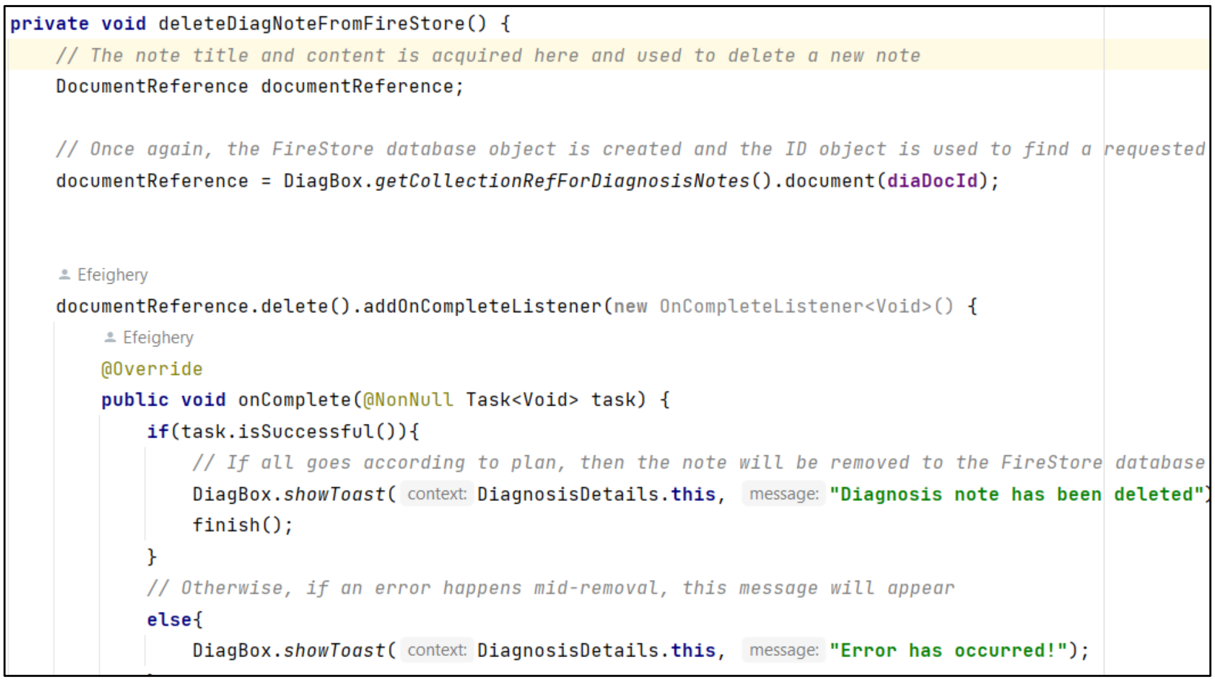
DiagnosesNotes.java:



```
void setUpRecycleView(){
    // A query is used to listen into a utility class tailor made to assist in CRUD functionalities
    // The method declared in ContactBox will order the saved the notes by their titles in ascending order
    Query query = DiagBox.getCollectionRefForDiagnosisNotes().orderBy( field: "title", Query.Direction.ASCENDING);

    // The instantiable class is declared in a FireStore Recycler object and the declared query will be confined with
    // the instantiable object to help create the RecyclerView and display the saved notes
    FirestoreRecyclerOptions<DiagnosisInfo> options = new FirestoreRecyclerOptions.Builder<DiagnosisInfo>()
            .setQuery(query, DiagnosisInfo.class).build();

    // The LayoutManager will be declared within the RecyclerView object
    reVe.setLayoutManager(new LinearLayoutManager( context: this));

    // And the Adapter file will be declared with the RecycleOptions object
    adapter = new DiagNoteAdapter(options,  context: this);
    reVe.setAdapter(adapter);
```

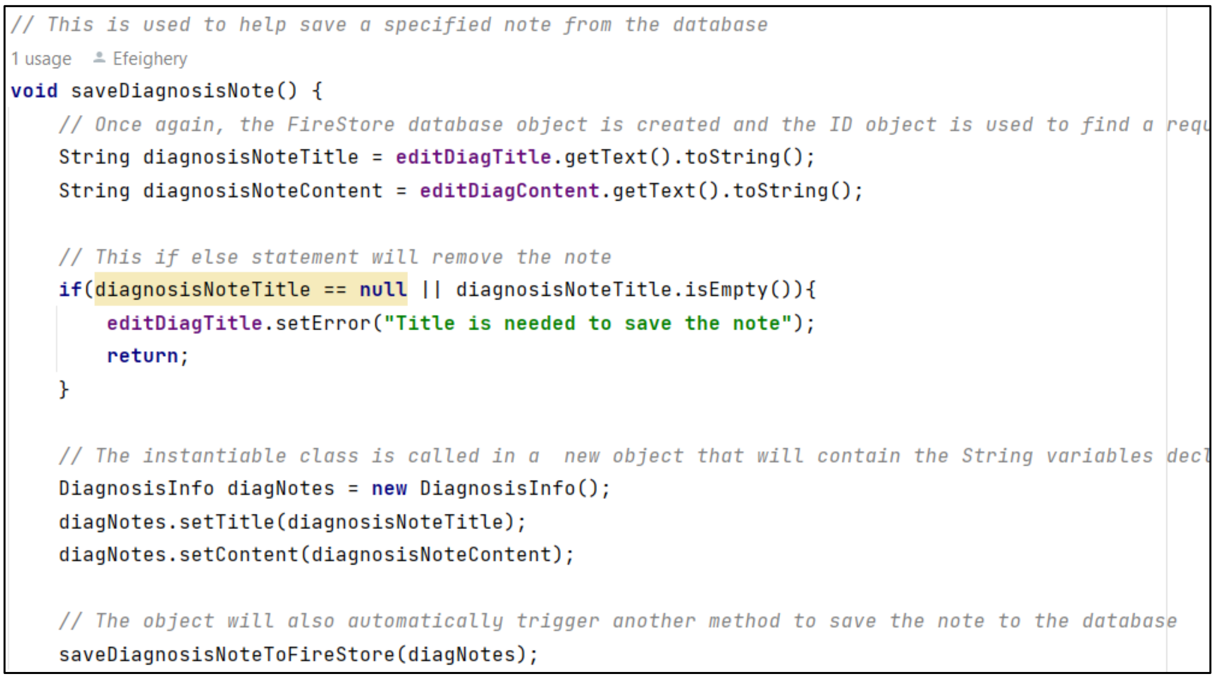
Figure 41: Gathering the saved diagnosis notes from FireStore and setting up them in a list view.

DiagnosisDetails.java:



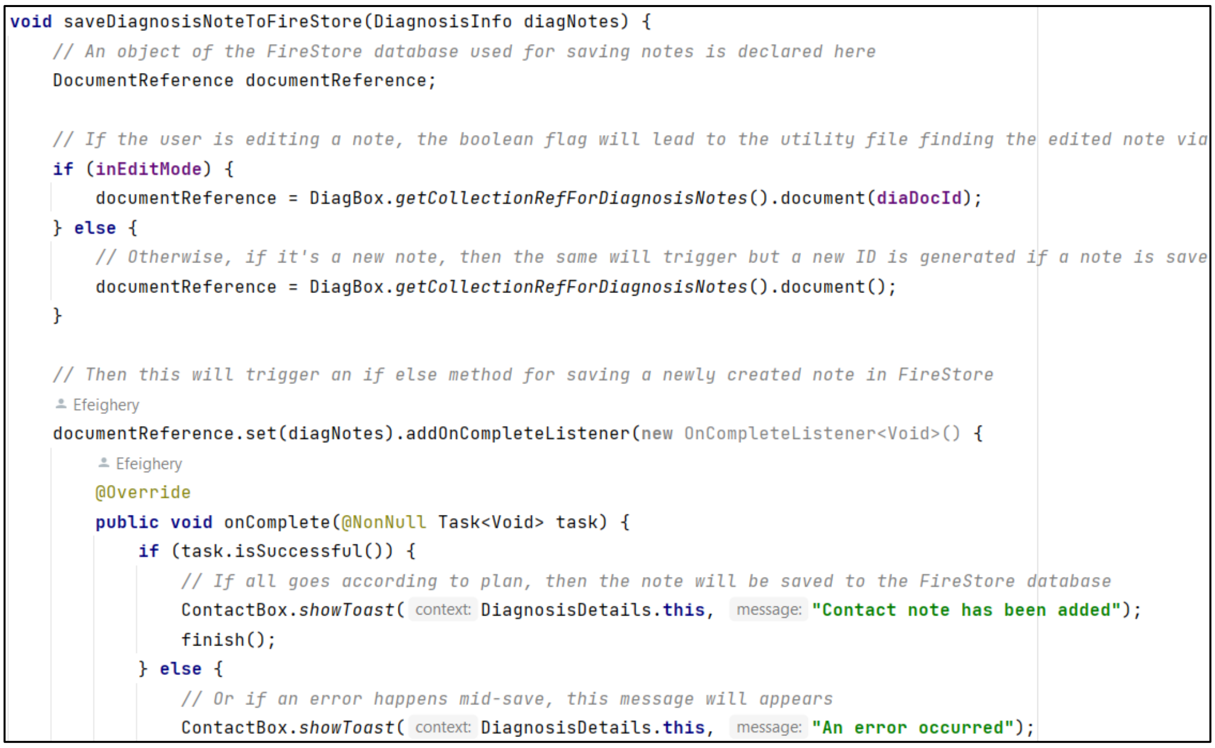
```
private void deleteDiagNoteFromFireStore() {
    // The note title and content is acquired here and used to delete a new note
    DocumentReference documentReference;

    // Once again, the FireStore database object is created and the ID object is used to find a requested
    documentReference = DiagBox.getCollectionRefForDiagnosisNotes().document(diaDocId);


    Efeighery
    documentReference.delete().addOnCompleteListener(new OnCompleteListener<Void>() {
        Efeighery
        @Override
        public void onComplete(@NonNull Task<Void> task) {
            if(task.isSuccessful()){
                // If all goes according to plan, then the note will be removed to the FireStore database
                DiagBox.showToast( context: DiagnosisDetails.this,  message: "Diagnosis note has been deleted")
                finish();
            }
            // Otherwise, if an error happens mid-removal, this message will appear
            else{
                DiagBox.showToast( context: DiagnosisDetails.this,  message: "Error has occurred!");
```

Figure 42: The method where a user deletes a Diagnosis note from its FireStore Database table.



```
// This is used to help save a specified note from the database
1 usage  Efeighery
void saveDiagnosisNote() {
    // Once again, the FireStore database object is created and the ID object is used to find a requ
    String diagnosisNoteTitle = editDiagTitle.getText().toString();
    String diagnosisNoteContent = editDiagContent.getText().toString();

    // This if else statement will remove the note
    if(diagnosisNoteTitle == null || diagnosisNoteTitle.isEmpty()){
        editDiagTitle.setError("Title is needed to save the note");
        return;
    }

    // The instantiable class is called in a  new object that will contain the String variables decl
    DiagnosisInfo diagNotes = new DiagnosisInfo();
    diagNotes.setTitle(diagnosisNoteTitle);
    diagNotes.setContent(diagnosisNoteContent);

    // The object will also automatically trigger another method to save the note to the database
    saveDiagnosisNoteToFireStore(diagNotes);
```

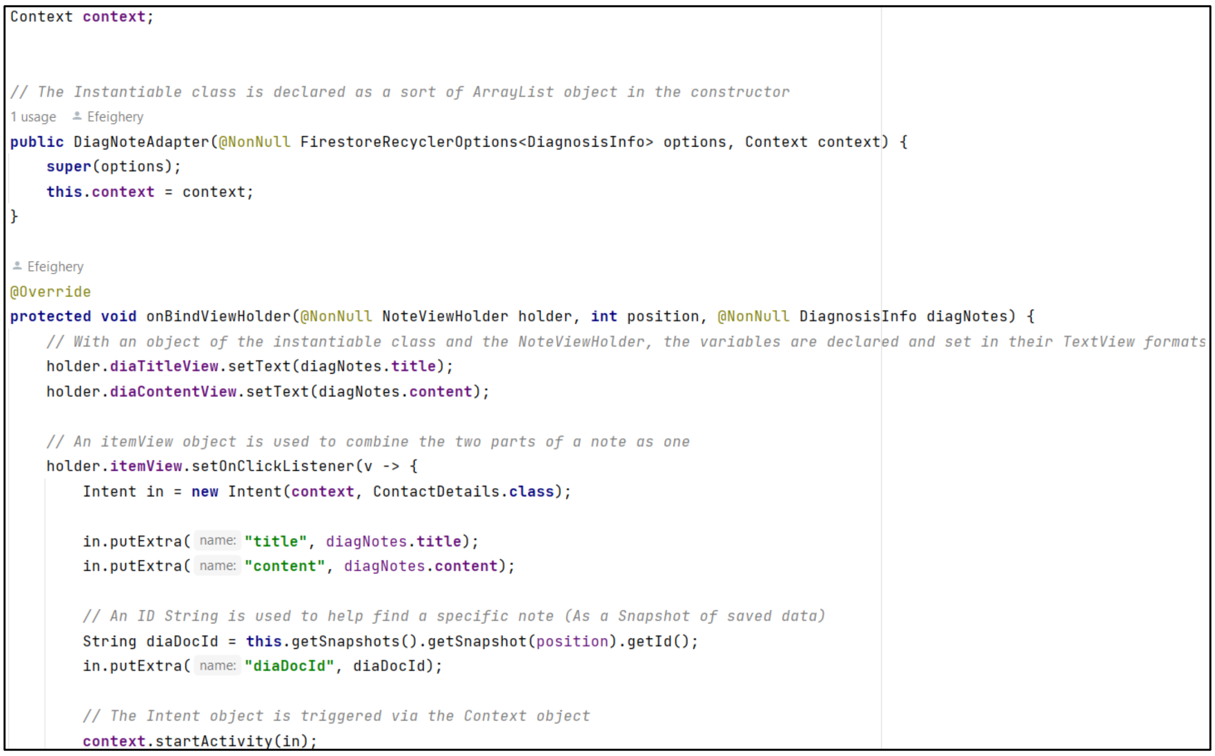
Figure 43: The first save note method which uses an object of the instantiable class.



```
void saveDiagnosisNoteToFireStore(DiagnosisInfo diagNotes) {
    // An object of the FireStore database used for saving notes is declared here
    DocumentReference documentReference;

    // If the user is editing a note, the boolean flag will lead to the utility file finding the edited note via
    if (inEditMode) {
        documentReference = DiagBox.getCollectionRefForDiagnosisNotes().document(diaDocId);
    } else {
        // Otherwise, if it's a new note, then the same will trigger but a new ID is generated if a note is save
        documentReference = DiagBox.getCollectionRefForDiagnosisNotes().document();
    }

    // Then this will trigger an if else method for saving a newly created note in FireStore
    Efeighery
    documentReference.set(diagNotes).addOnCompleteListener(new OnCompleteListener<Void>() {
        Efeighery
        @Override
        public void onComplete(@NonNull Task<Void> task) {
            if (task.isSuccessful()) {
                // If all goes according to plan, then the note will be saved to the FireStore database
                ContactBox.showToast( context: DiagnosisDetails.this,  message: "Contact note has been added");
                finish();
            } else {
                // Or if an error happens mid-save, this message will appears
                ContactBox.showToast( context: DiagnosisDetails.this,  message: "An error occurred");
```

Figure 44: The method where a user saves a Diagnosis note to the FireStore Database table using a designated ID.

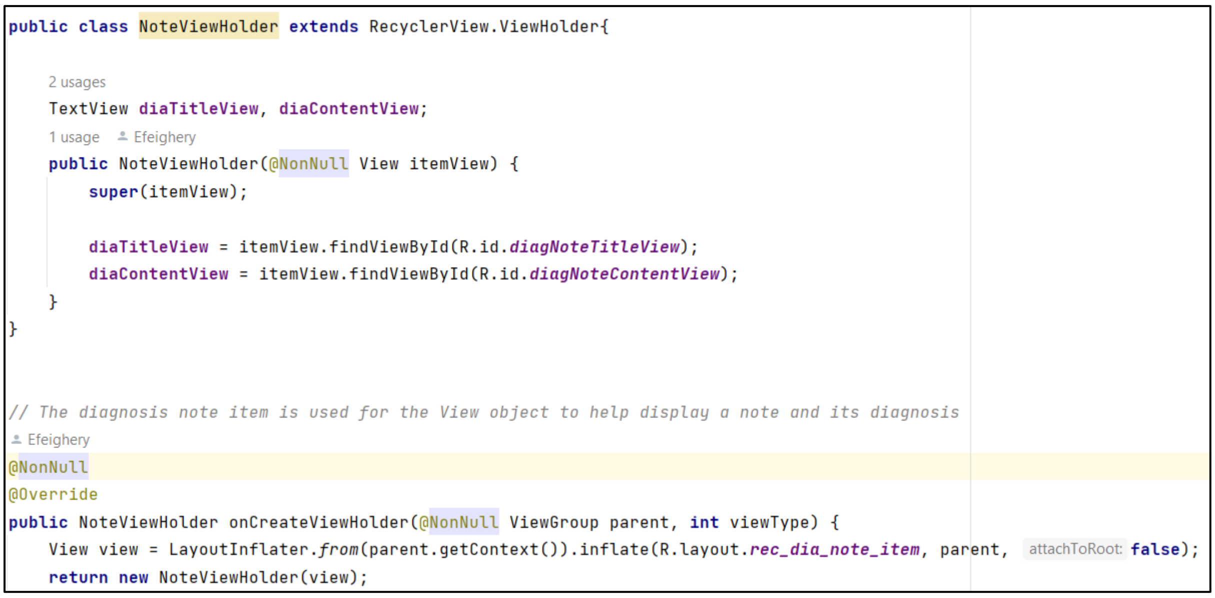
DiagNoteAdapter.java:



```
Context context;


// The Instantiable class is declared as a sort of ArrayList object in the constructor
1 usage   Efeighery
public DiagNoteAdapter(@NonNull FirestoreRecyclerOptions<DiagnosisInfo> options, Context context) {
    super(options);
    this.context = context;
}

Efeighery
@Override
protected void onBindViewHolder(@NonNull NoteViewHolder holder, int position, @NonNull DiagnosisInfo diagNotes) {
    // With an object of the instantiable class and the NoteViewHolder, the variables are declared and set in their TextView formats
    holder.diaTitleView.setText(diagNotes.title);
    holder.diaContentView.setText(diagNotes.content);

    // An itemView object is used to combine the two parts of a note as one
    holder.itemView.setOnClickListener(v -> {
        Intent in = new Intent(context, ContactDetails.class);

        in.putExtra( name: "title", diagNotes.title);
        in.putExtra( name: "content", diagNotes.content);

        // An ID String is used to help find a specific note (As a Snapshot of saved data)
        String diaDocId = this.getSnapshots().getSnapshot(position).getId();
        in.putExtra( name: "diaDocId", diaDocId);

        // The Intent object is triggered via the Context object
        context.startActivity(in);
```

Figure 45: Setting the ViewHolder for the Adapter object for Diagnosis Notes with an object of the instantiable class.

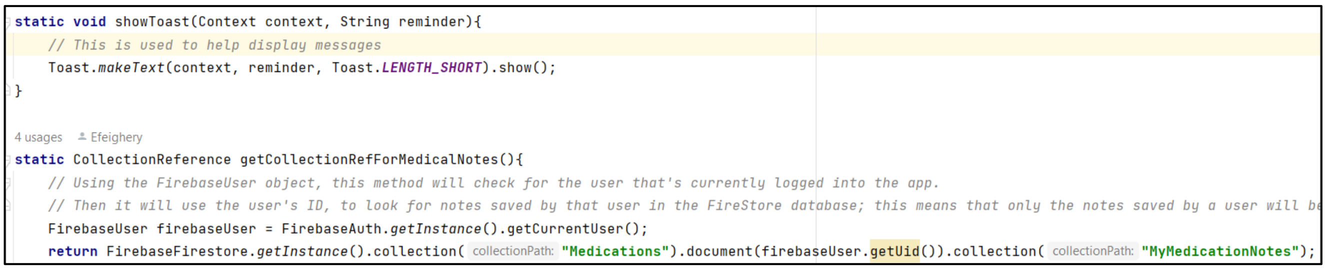
DiagNoteAdapter.java:



```
public class NoteViewHolder extends RecyclerView.ViewHolder{

    2 usages
    TextView diaTitleView, diaContentView;
    1 usage  Efeighery
    public NoteViewHolder(@NonNull View itemView) {
        super(itemView);

        diaTitleView = itemView.findViewById(R.id.diagNoteTitleView);
        diaContentView = itemView.findViewById(R.id.diagNoteContentView);
    }
}


// The diagnosis note item is used for the View object to help display a note and its diagnosis
 Efeighery
@NonNull
@Override
public NoteViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.rec_dia_note_item, parent,  attachToRoot: false);
    return new NoteViewHolder(view);
```

Figure 46: The ViewHolder binder and initialising the note item for Diagnosis notes.

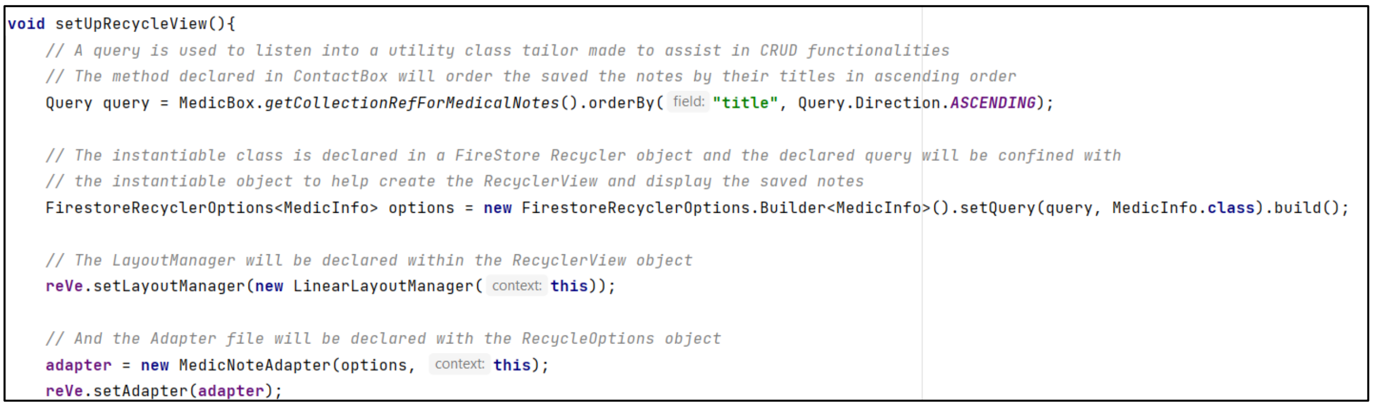
MedicBox.java:



```
static void showToast(Context context, String reminder){
    // This is used to help display messages
    Toast.makeText(context, reminder, Toast.LENGTH_SHORT).show();
}

4 usages  Efeighery
static CollectionReference getCollectionRefForMedicalNotes(){
    // Using the FirebaseUser object, this method will check for the user that's currently logged into the app.
    // Then it will use the user's ID, to look for notes saved by that user in the FireStore database; this means that only the notes saved by a user will be
    FirebaseUser firebaseUser = FirebaseAuth.getInstance().getCurrentUser();
    return FirebaseFirestore.getInstance().collection( collectionPath: "Medications").document(firebaseUser.getUid()).collection( collectionPath: "MyMedicationNotes");
```

Figure 47: The instantiable class that allows the application to collect all of the saved medication notes in their designated database table.

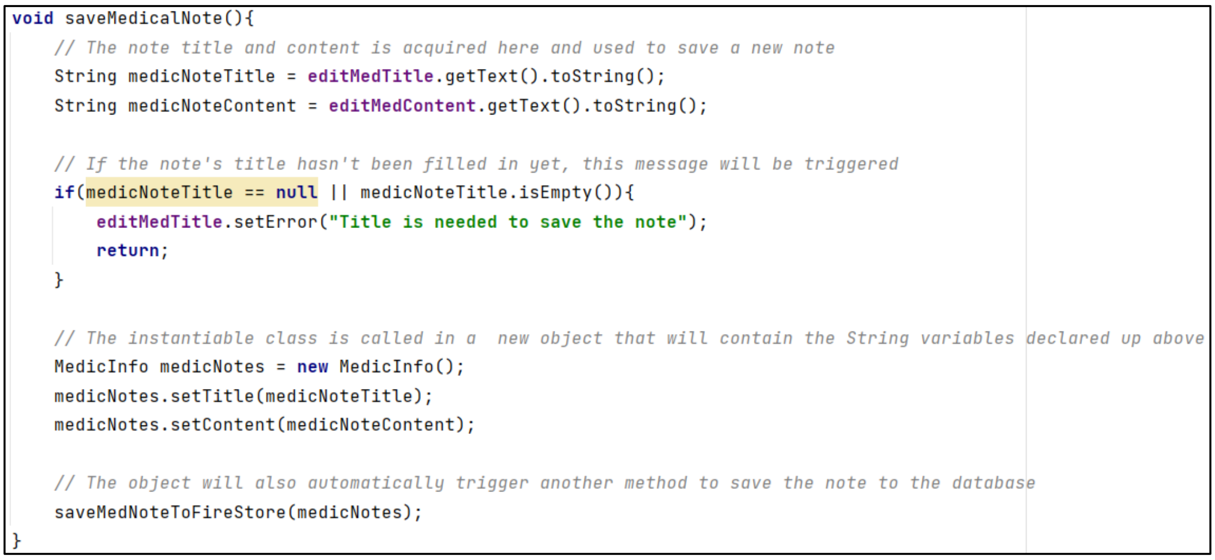
MedicNotes.java:



```
void setUpRecycleView(){
    // A query is used to listen into a utility class tailor made to assist in CRUD functionalities
    // The method declared in ContactBox will order the saved the notes by their titles in ascending order
    Query query = MedicBox.getCollectionRefForMedicalNotes().orderBy( field: "title", Query.Direction.ASCENDING);

    // The instantiable class is declared in a FireStore Recycler object and the declared query will be confined with
    // the instantiable object to help create the RecyclerView and display the saved notes
    FirestoreRecyclerOptions<MedicInfo> options = new FirestoreRecyclerOptions.Builder<MedicInfo>().setQuery(query, MedicInfo.class).build();

    // The LayoutManager will be declared within the RecyclerView object
    reVe.setLayoutManager(new LinearLayoutManager( context: this));

    // And the Adapter file will be declared with the RecycleOptions object
    adapter = new MedicNoteAdapter(options,  context: this);
    reVe.setAdapter(adapter);
```

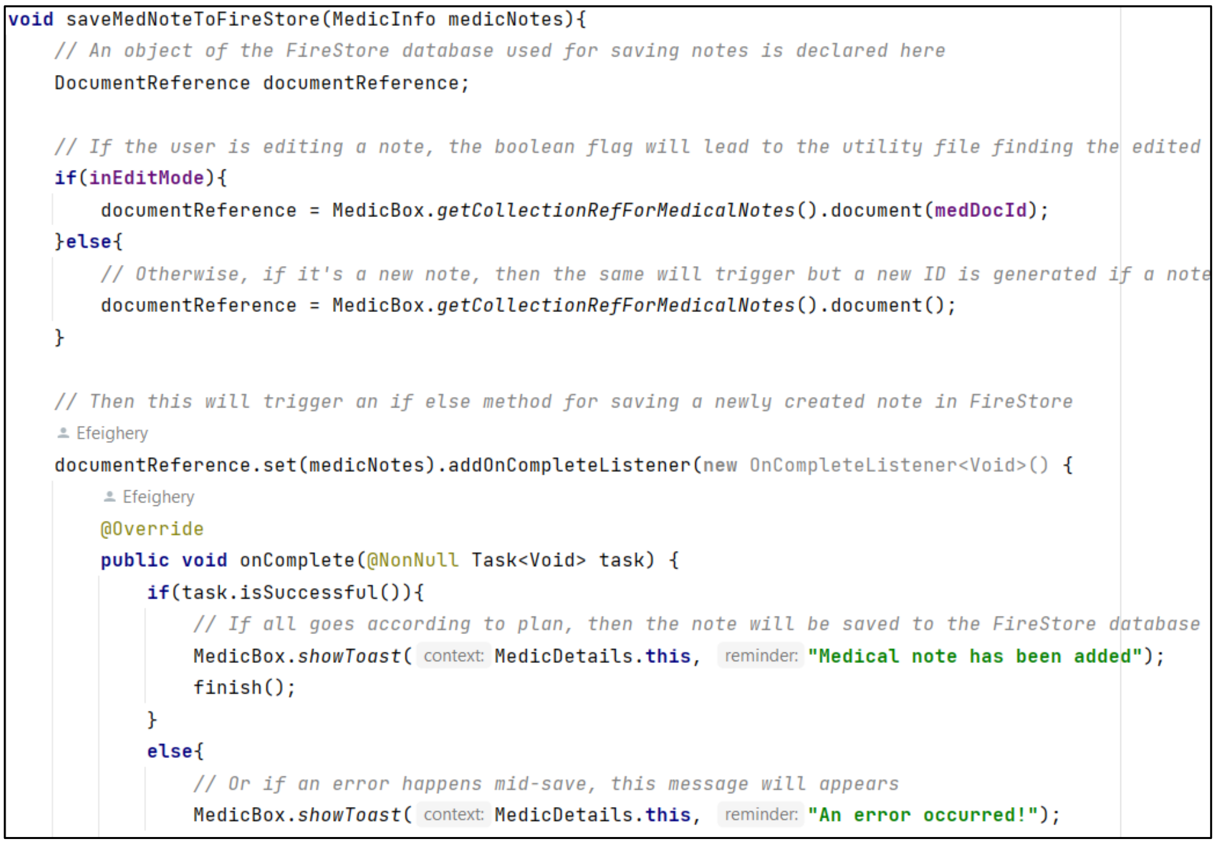
Figure 48: Gathering the saved medication notes from FireStore and setting up them in a list view.

MedicDetails.java:



```java
void saveMedicalNote(){
    // The note title and content is acquired here and used to save a new note
    String medicNoteTitle = editMedTitle.getText().toString();
    String medicNoteContent = editMedContent.getText().toString();

    // If the note's title hasn't been filled in yet, this message will be triggered
    if(medicNoteTitle == null || medicNoteTitle.isEmpty()){
        editMedTitle.setError("Title is needed to save the note");
        return;
    }

    // The instantiable class is called in a  new object that will contain the String variables declared up above
    MedicInfo medicNotes = new MedicInfo();
    medicNotes.setTitle(medicNoteTitle);
    medicNotes.setContent(medicNoteContent);

    // The object will also automatically trigger another method to save the note to the database
    saveMedNoteToFireStore(medicNotes);
}
```

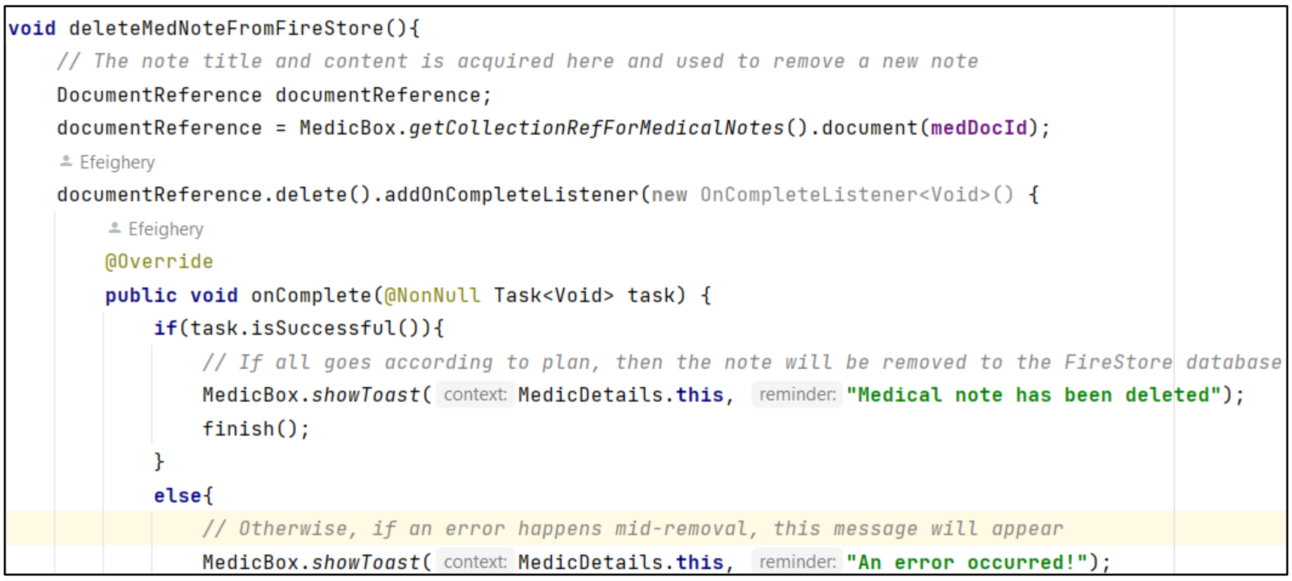
Figure 49: The first save note method which uses an object of the instantiable class.



```java
void saveMedNoteToFireStore(MedicInfo medicNotes){
    // An object of the FireStore database used for saving notes is declared here
    DocumentReference documentReference;

    // If the user is editing a note, the boolean flag will lead to the utility file finding the edited
    if(inEditMode){
        documentReference = MedicBox.getCollectionRefForMedicalNotes().document(medDocId);
    }else{
        // Otherwise, if it's a new note, then the same will trigger but a new ID is generated if a note
        documentReference = MedicBox.getCollectionRefForMedicalNotes().document();
    }

    // Then this will trigger an if else method for saving a newly created note in FireStore
    Efeighery
    documentReference.set(medicNotes).addOnCompleteListener(new OnCompleteListener<Void>() {
        Efeighery
        @Override
        public void onComplete(@NonNull Task<Void> task) {
            if(task.isSuccessful()){
                // If all goes according to plan, then the note will be saved to the FireStore database
                MedicBox.showToast( context: MedicDetails.this,  reminder: "Medical note has been added");
                finish();
            }
            else{
                // Or if an error happens mid-save, this message will appears
                MedicBox.showToast( context: MedicDetails.this,  reminder: "An error occurred!");
```

Figure 50: The method where a user saves a Medication note to the FireStore Database table using a designated ID.

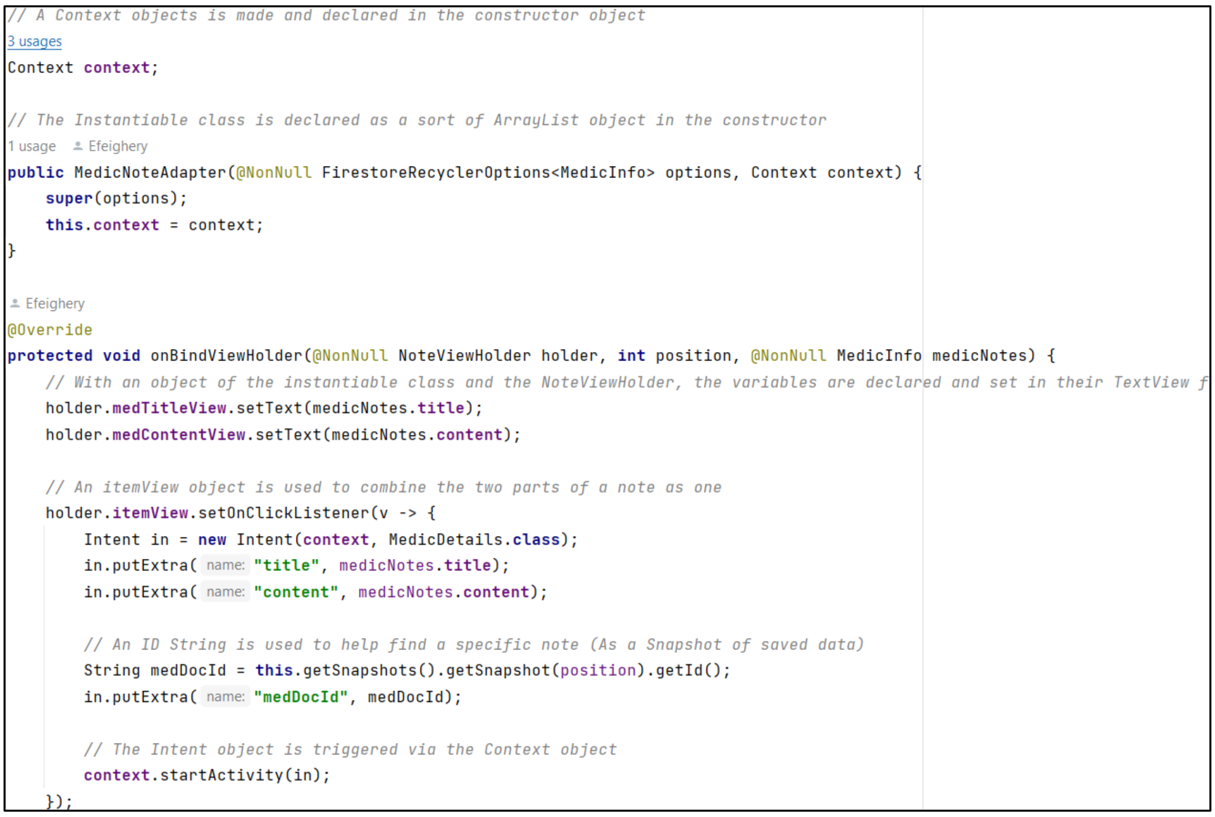
MedicDetails.java:



```
void deleteMedNoteFromFireStore(){
    // The note title and content is acquired here and used to remove a new note
    DocumentReference documentReference;
    documentReference = MedicBox.getCollectionRefForMedicalNotes().document(medDocId);
    Efeighery
    documentReference.delete().addOnCompleteListener(new OnCompleteListener<Void>() {
        Efeighery
        @Override
        public void onComplete(@NonNull Task<Void> task) {
            if(task.isSuccessful()){
                // If all goes according to plan, then the note will be removed to the FireStore database
                MedicBox.showToast( context: MedicDetails.this,  reminder: "Medical note has been deleted");
                finish();
            }
            else{
                // Otherwise, if an error happens mid-removal, this message will appear
                MedicBox.showToast( context: MedicDetails.this,  reminder: "An error occurred!");
```

Figure 51: The method where a user deletes a Medication note from its FireStore Database table.

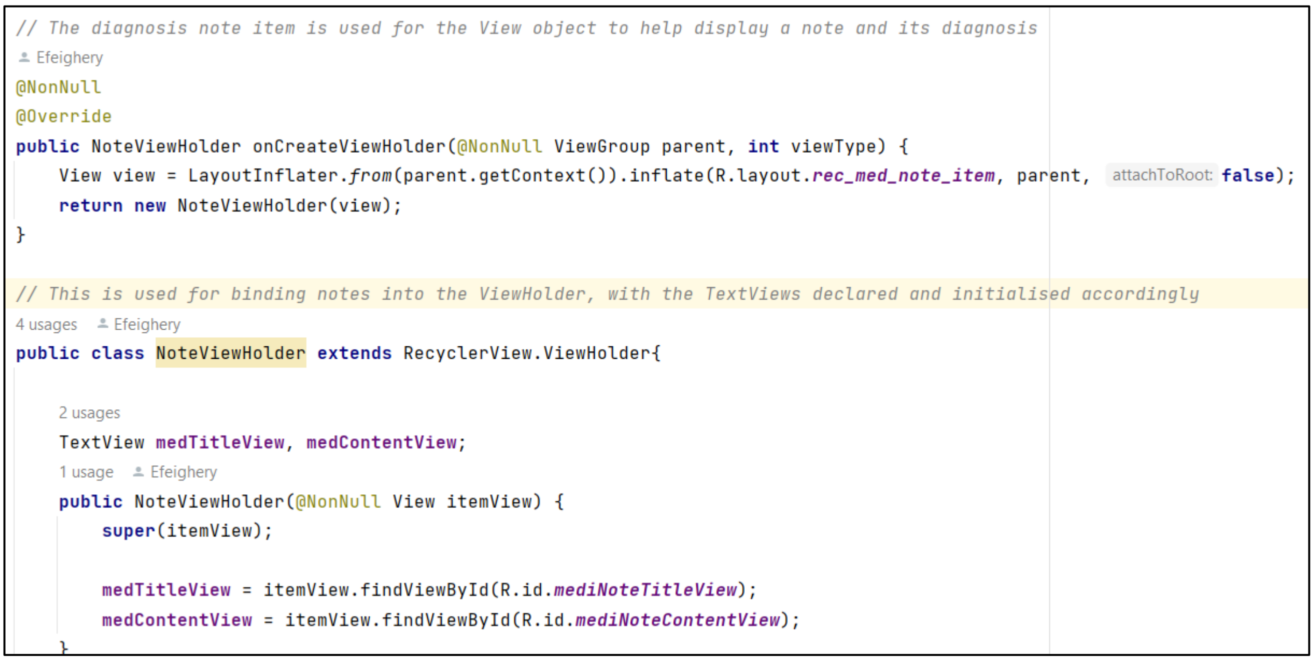
MedicNoteAdapter.java:



```
// A Context objects is made and declared in the constructor object
3 usages
Context context;

// The Instantiable class is declared as a sort of ArrayList object in the constructor
1 usage  Efeighery
public MedicNoteAdapter(@NonNull FirestoreRecyclerOptions<MedicInfo> options, Context context) {
    super(options);
    this.context = context;
}

Efeighery
@Override
protected void onBindViewHolder(@NonNull NoteViewHolder holder, int position, @NonNull MedicInfo medicNotes) {
    // With an object of the instantiable class and the NoteViewHolder, the variables are declared and set in their TextView f
    holder.medTitleView.setText(medicNotes.title);
    holder.medContentView.setText(medicNotes.content);

    // An itemView object is used to combine the two parts of a note as one
    holder.itemView.setOnClickListener(v -> {
        Intent in = new Intent(context, MedicDetails.class);
        in.putExtra( name: "title", medicNotes.title);
        in.putExtra( name: "content", medicNotes.content);

        // An ID String is used to help find a specific note (As a Snapshot of saved data)
        String medDocId = this.getSnapshots().getSnapshot(position).getId();
        in.putExtra( name: "medDocId", medDocId);

        // The Intent object is triggered via the Context object
        context.startActivity(in);
    });
```

Figure 52: Setting the ViewHolder for the Adapter object for Medication Notes with an object of the instantiable class.

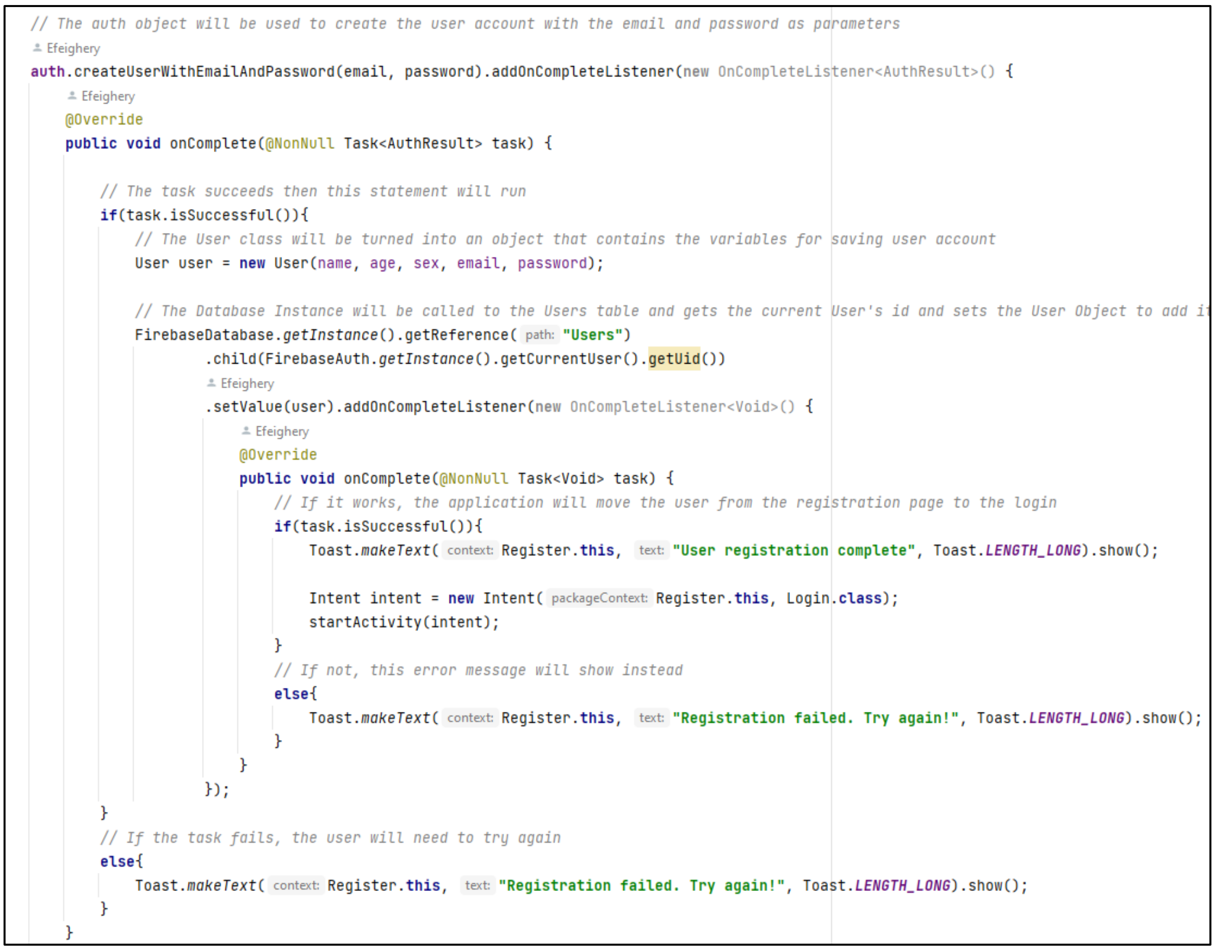
MedicNoteAdapter.java:



```
// The diagnosis note item is used for the View object to help display a note and its diagnosis
Efeighery
@NonNull
@Override
public NoteViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.rec_med_note_item, parent,  attachToRoot: false);
    return new NoteViewHolder(view);
}

// This is used for binding notes into the ViewHolder, with the TextViews declared and initialised accordingly
4 usages  Efeighery
public class NoteViewHolder extends RecyclerView.ViewHolder{

    2 usages
    TextView medTitleView, medContentView;
    1 usage  Efeighery
    public NoteViewHolder(@NonNull View itemView) {
        super(itemView);

        medTitleView = itemView.findViewById(R.id.mediNoteTitleView);
        medContentView = itemView.findViewById(R.id.mediNoteContentView);
    }
```

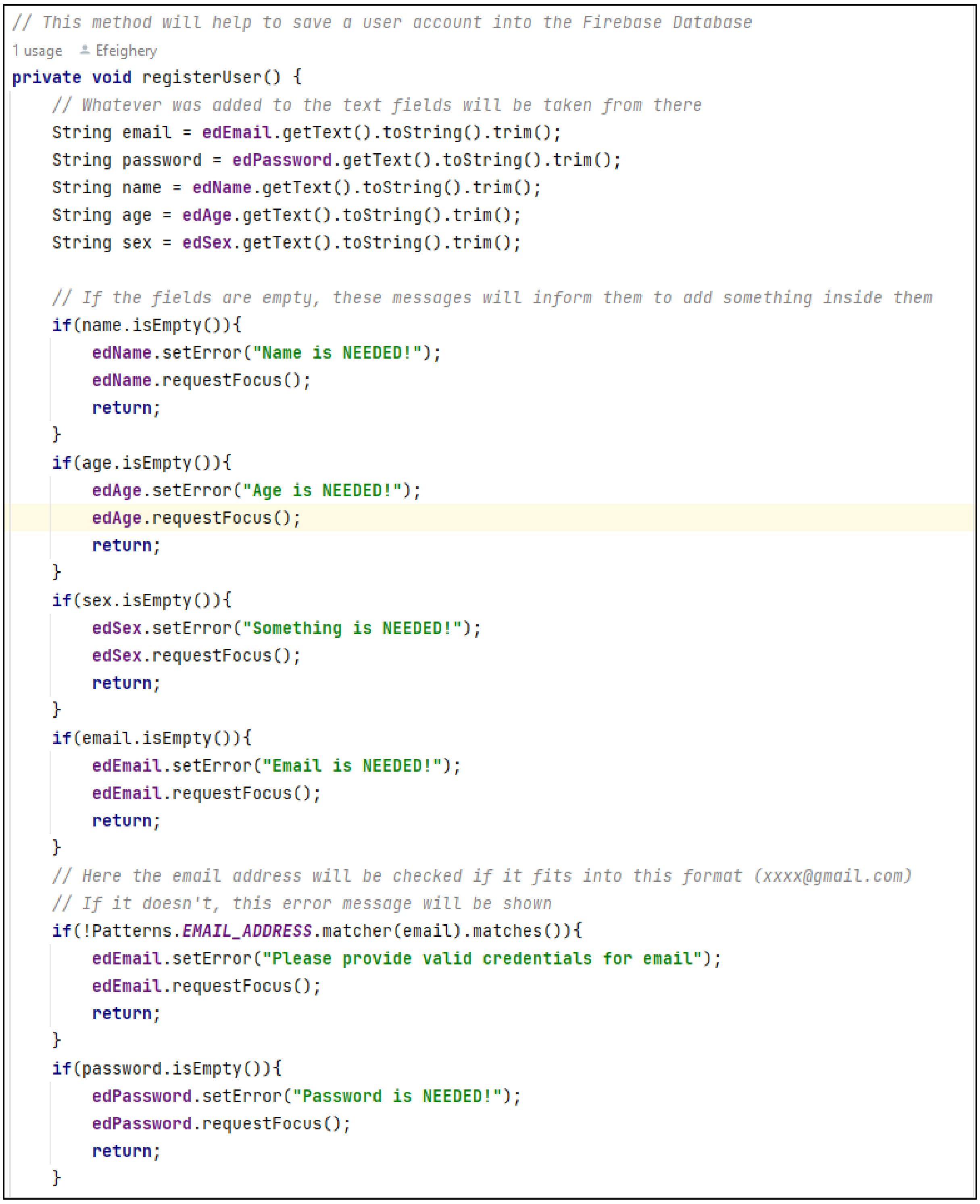
Figure 53: The ViewHolder binder and initialising the note item for Medication notes.

Register.java:



```java
// The auth object will be used to create the user account with the email and password as parameters
Efeighery
auth.createUserWithEmailAndPassword(email, password).addOnCompleteListener(new OnCompleteListener<AuthResult>() {
    Efeighery
    @Override
    public void onComplete(@NonNull Task<AuthResult> task) {

        // The task succeeds then this statement will run
        if(task.isSuccessful()){
            // The User class will be turned into an object that contains the variables for saving user account
            User user = new User(name, age, sex, email, password);

            // The Database Instance will be called to the Users table and gets the current User's id and sets the User Object to add i
            FirebaseDatabase.getInstance().getReference( path: "Users")
                    .child(FirebaseAuth.getInstance().getCurrentUser().getUid())
                    Efeighery
                    .setValue(user).addOnCompleteListener(new OnCompleteListener<Void>() {
                        Efeighery
                        @Override
                        public void onComplete(@NonNull Task<Void> task) {
                            // If it works, the application will move the user from the registration page to the login
                            if(task.isSuccessful()){
                                Toast.makeText( context: Register.this,  text: "User registration complete", Toast.LENGTH_LONG).show();

                                Intent intent = new Intent( packageContext: Register.this, Login.class);
                                startActivity(intent);
                            }
                            // If not, this error message will show instead
                            else{
                                Toast.makeText( context: Register.this,  text: "Registration failed. Try again!", Toast.LENGTH_LONG).show();
                            }
                        }
                    });
        }
        // If the task fails, the user will need to try again
        else{
            Toast.makeText( context: Register.this,  text: "Registration failed. Try again!", Toast.LENGTH_LONG).show();
        }
    }
```

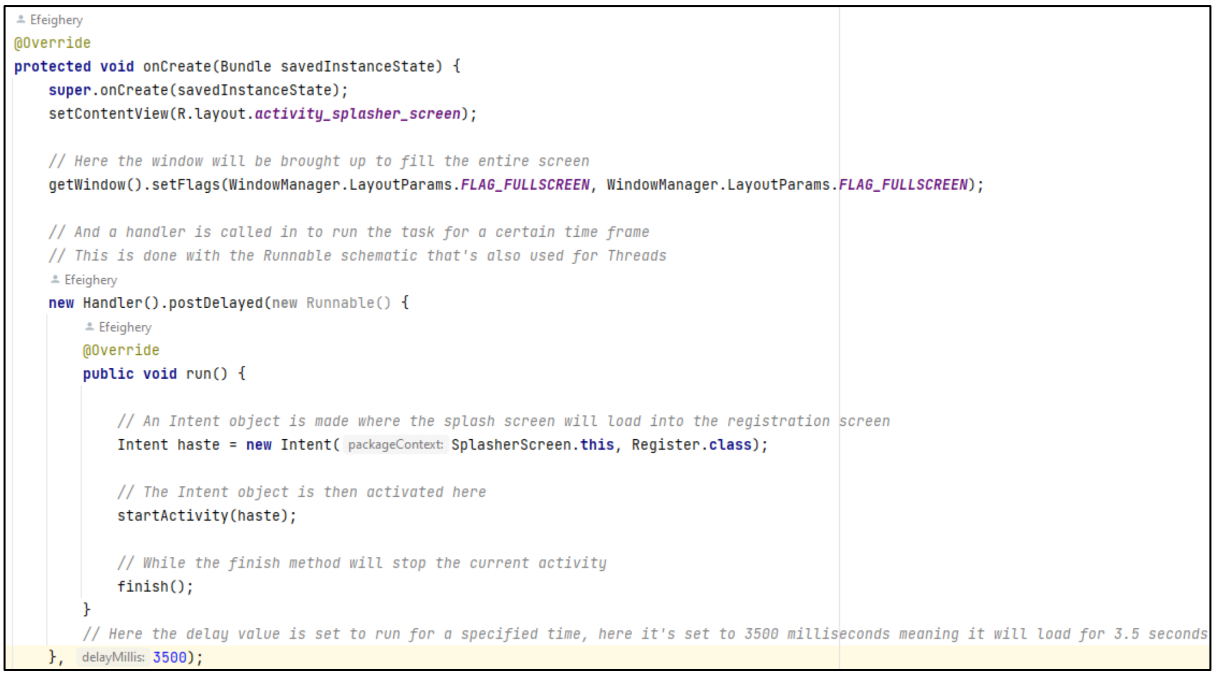
Figure 54: The process of adding authentication to a new user.



```
// This method will help to save a user account into the Firebase Database
1 usage  Efeighery
private void registerUser() {
    // Whatever was added to the text fields will be taken from there
    String email = edEmail.getText().toString().trim();
    String password = edPassword.getText().toString().trim();
    String name = edName.getText().toString().trim();
    String age = edAge.getText().toString().trim();
    String sex = edSex.getText().toString().trim();

    // If the fields are empty, these messages will inform them to add something inside them
    if(name.isEmpty()){
        edName.setError("Name is NEEDED!");
        edName.requestFocus();
        return;
    }
    if(age.isEmpty()){
        edAge.setError("Age is NEEDED!");
        edAge.requestFocus();
        return;
    }
    if(sex.isEmpty()){
        edSex.setError("Something is NEEDED!");
        edSex.requestFocus();
        return;
    }
    if(email.isEmpty()){
        edEmail.setError("Email is NEEDED!");
        edEmail.requestFocus();
        return;
    }
    // Here the email address will be checked if it fits into this format (xxxx@gmail.com)
    // If it doesn't, this error message will be shown
    if(!Patterns.EMAIL_ADDRESS.matcher(email).matches()){
        edEmail.setError("Please provide valid credentials for email");
        edEmail.requestFocus();
        return;
    }
    if(password.isEmpty()){
        edPassword.setError("Password is NEEDED!");
        edPassword.requestFocus();
        return;
    }
```

Figure 55: The method used to register a new User to the Database table.

SplashScreen.java:



```
Efeighery
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_splasher_screen);

    // Here the window will be brought up to fill the entire screen
    getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN, WindowManager.LayoutParams.FLAG_FULLSCREEN);

    // And a handler is called in to run the task for a certain time frame
    // This is done with the Runnable schematic that's also used for Threads
    Efeighery
    new Handler().postDelayed(new Runnable() {
        Efeighery
        @Override
        public void run() {

            // An Intent object is made where the splash screen will load into the registration screen
            Intent haste = new Intent( packageContext: SplasherScreen.this, Register.class);

            // The Intent object is then activated here
            startActivity(haste);

            // While the finish method will stop the current activity
            finish();
        }
        // Here the delay value is set to run for a specified time, here it's set to 3500 milliseconds meaning it will load for 3.5 seconds
    }, delayMillis: 3500);
```

Figure 56: Demonstrating what makes a Splasher screen and how the app's logo is displayed for branding and aesthetics.

2.4 Graphical User Interface (GUI)

Registration Page:

When the user boots up the app, they will be taken to the registration page which allows users to register their information to create an account. If the fields are empty, the system will notify the user to fill them in so that the user account can be confirmed into the Firebase database user table. If they already created an account, they could click the button below the Register Account one, to log in.

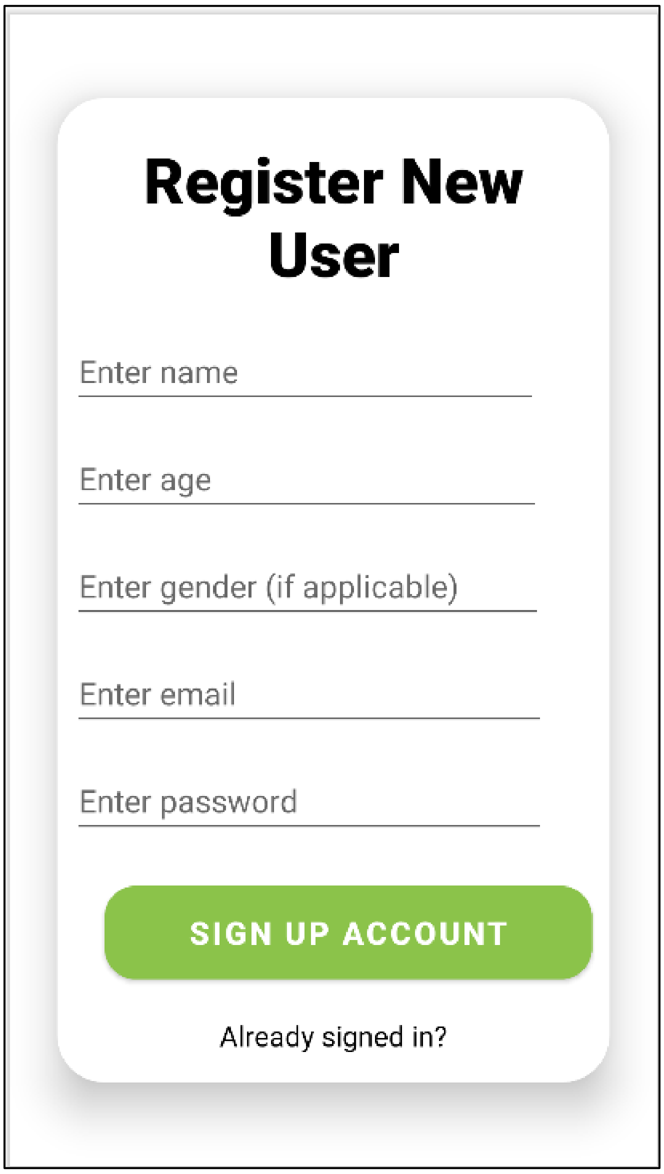


Figure 57: Registration page with an option to log in if a user has logged in previously.

Login Page:

In the login page, users only need to enter their usernames and password to enter the home page if they have already made an account prior. Otherwise, the query button below the Registration page will be used to create an account.

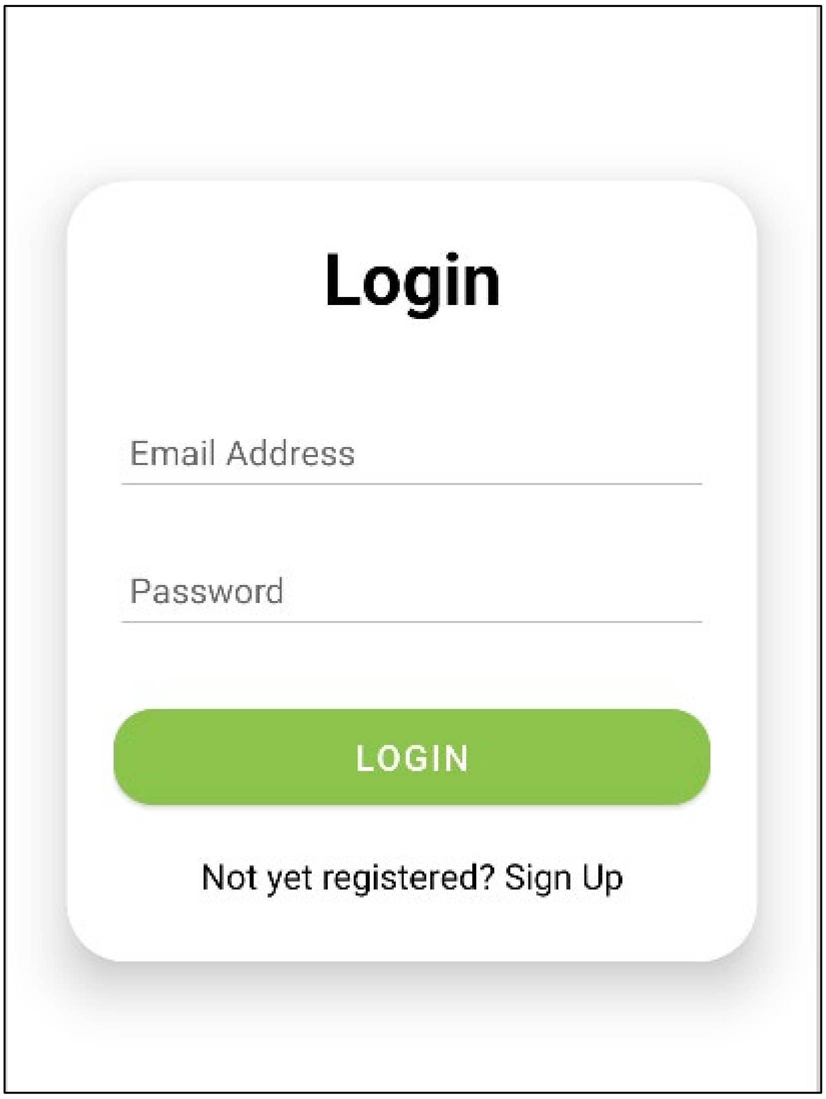
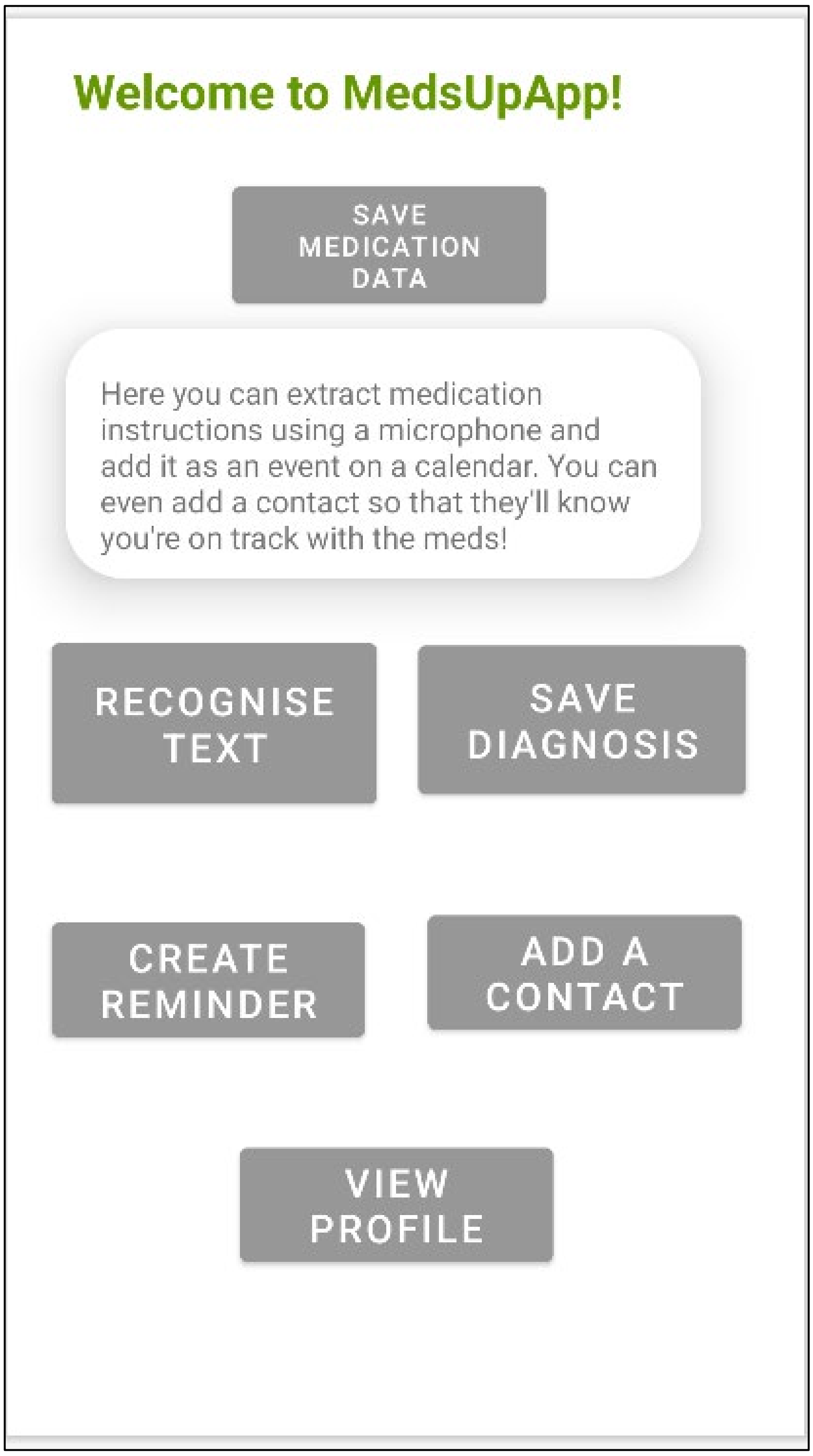


Figure 58: The overall layout of the login page

Home Page:

Once the users make it through either pathway, they'll be taken to the home page which gives a brief description on what each page does and offers buttons to access said pages. There's even a log out button to exit the application.



Welcome to MedsUpApp!

Here you can extract medication instructions using a microphone and add it as an event on a calendar. You can even add a contact so that they'll know you're on track with the meds!

Figure 59: The home page featuring a description of what you can do in the application.

User Profile and Edit Pages:

In the user profile, the user account details will be searched in the database and taken to display them in the profile. The edit profile page will allow users to change any of the fields to whatever they see fit. Once they click the save button, the new user details will be added to the current profile.

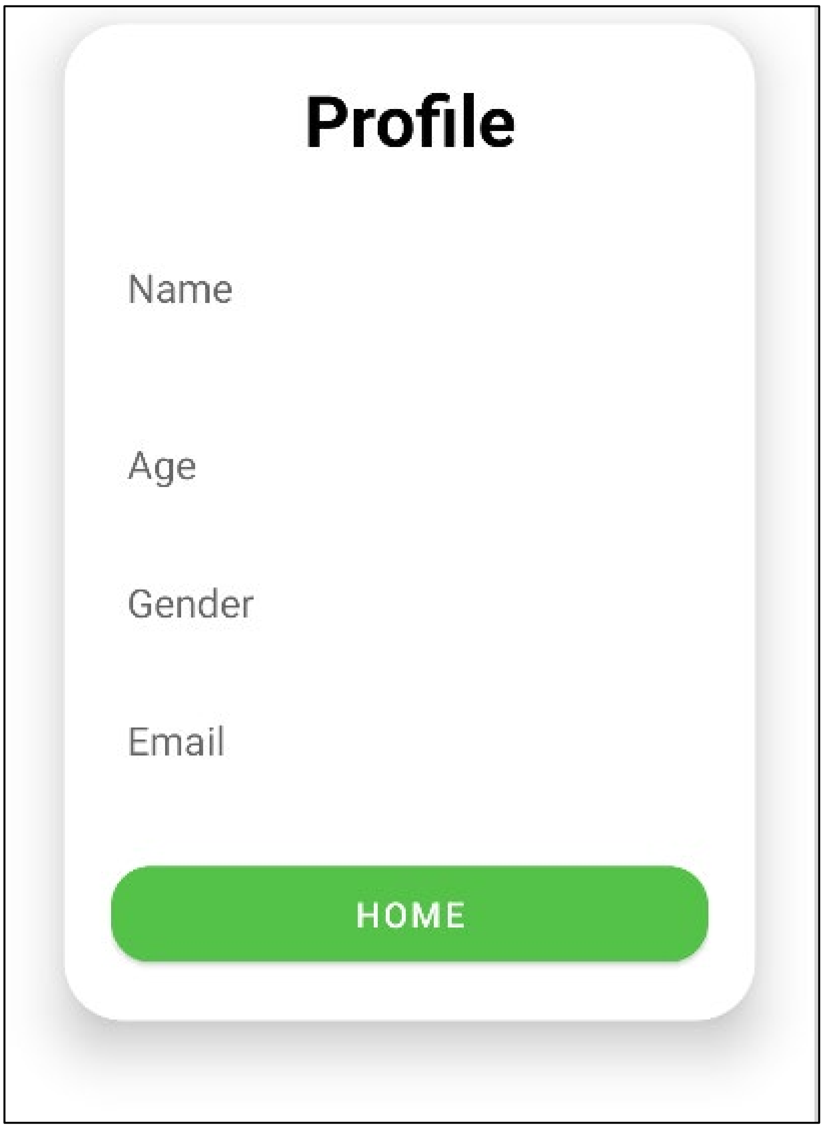
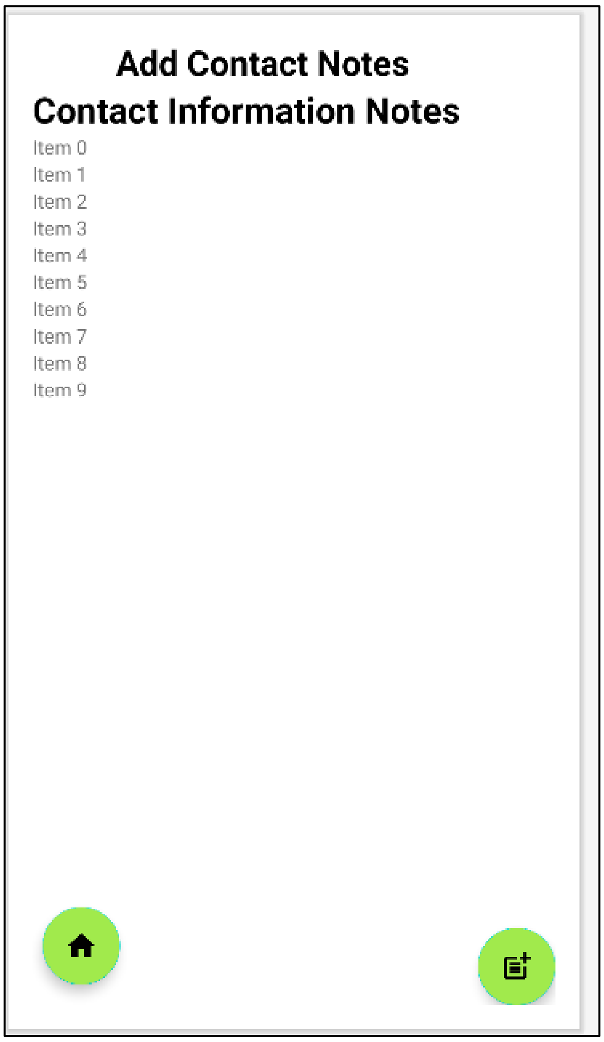


Figure 60: The user profile which acts as a sort of data retrieval function.

Contacts Page:

In the contacts page, a user can add a note containing information about a contact by entering their name, email address and phone number.

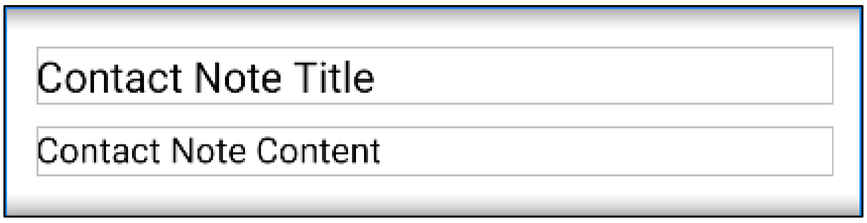

Add Contact Notes

Contact Information Notes

Item 0
Item 1
Item 2
Item 3
Item 4
Item 5
Item 6
Item 7
Item 8
Item 9

Figure 61: The menu that displays saved contact notes.

Once the details are added, the contact can be saved to a database table and will be displayed in the following format as seen in the image below.


Contact Note Title

Contact Note Content

Figure 62: The layout of the note's contents (title and content)

If a contact is chosen to be deleted, the message found at the bottom of the page will be used to confirm if the user wants to go through with this. If they click it, the contact will be permanently deleted, if they choose not to, the contact will remain in the list.

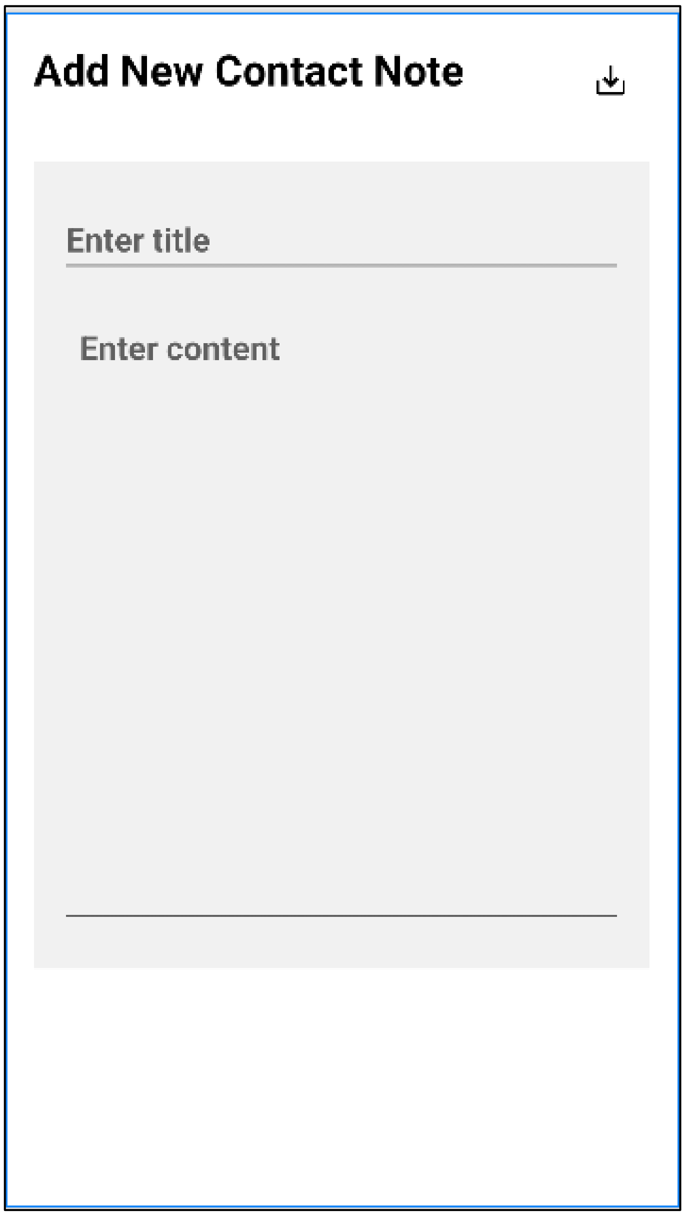


Figure 63: How a contact entry will appear when saved on the respective list.

Email Maker page:

When the user goes into the email page from the add contact one, they'll have to type in an email address, the subject of the email and the message body. In this case, letting a contact know you took your medication for the day. When you click on the format email button, it can take you straight to your Gmail. As with the calendar maker button, you'll be asked to login to your Gmail but only once.

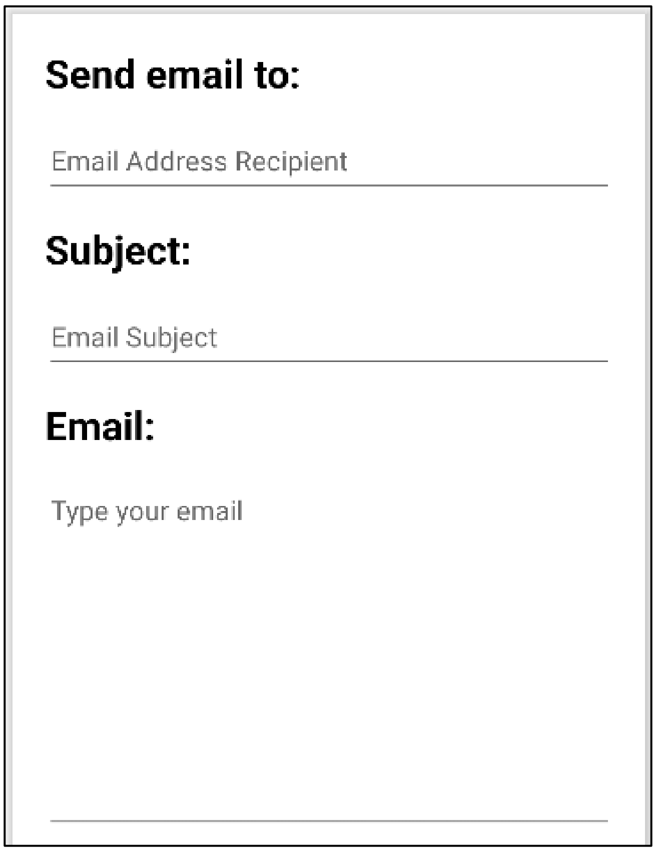


Figure 64: The email maker page layout

Speech to Text Page:

In the speech to text page, you should see a microphone button and an empty text box. The user needs to click on the microphone which will showcase a prompt for the user to say their medication instructions out loud. Once the recogniser technology finds all of the words in the captured audio, the text box will display these words and the user can click on the “Copy Text” button. Doing this will save the text onto the clipboard.

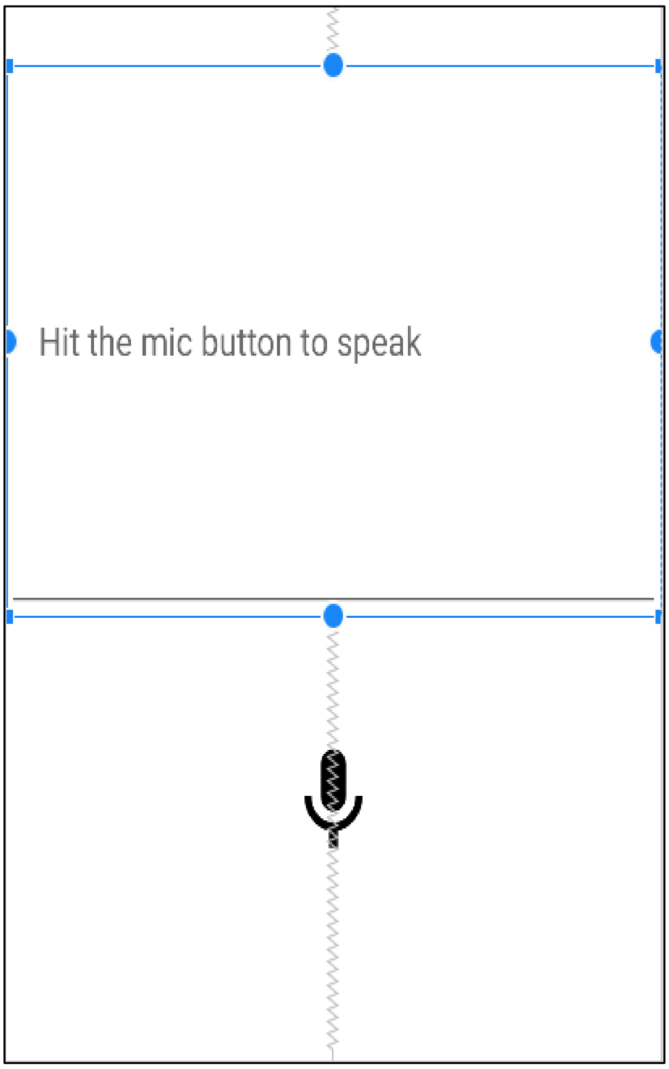


Figure 65: The speech to text page which functions similar to the microphone option in YouTube and Google Translate.

Text Message Maker Page:

In the add contact list page, clicking on the button titled "Send Text" will bring you to the Text Maker page. In a somewhat similar vein to the Email Maker page, you'll be able to notify a contact by text message. This will typically require both a contact and a message containing text, i.e., "Message: I just took my medication for the day/I finished my medication round".

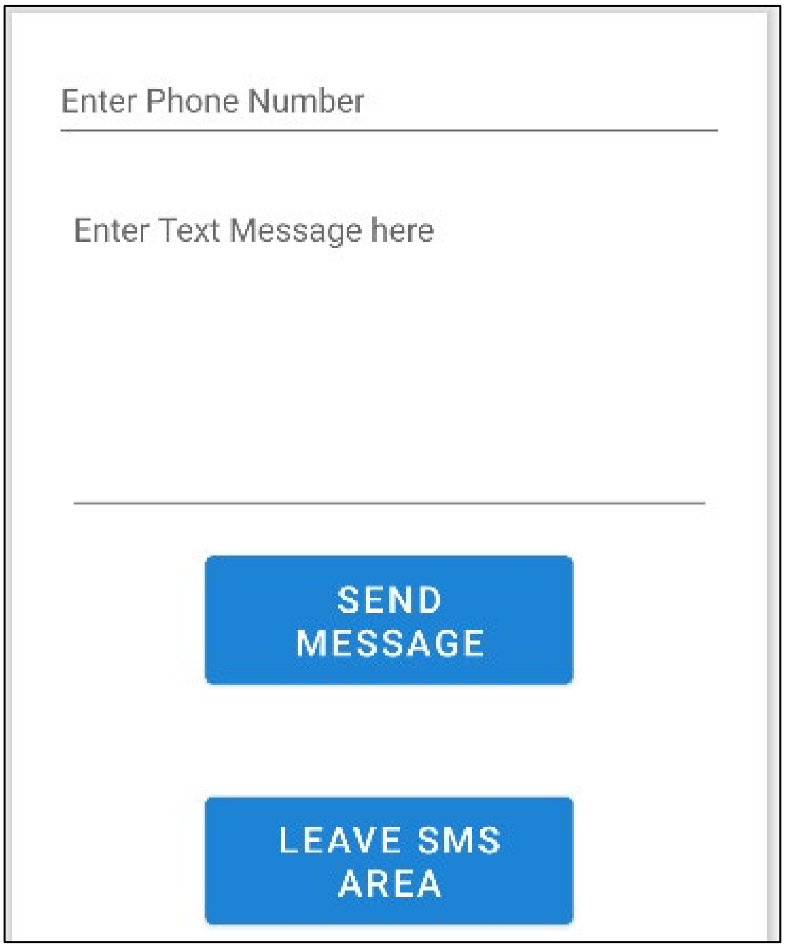
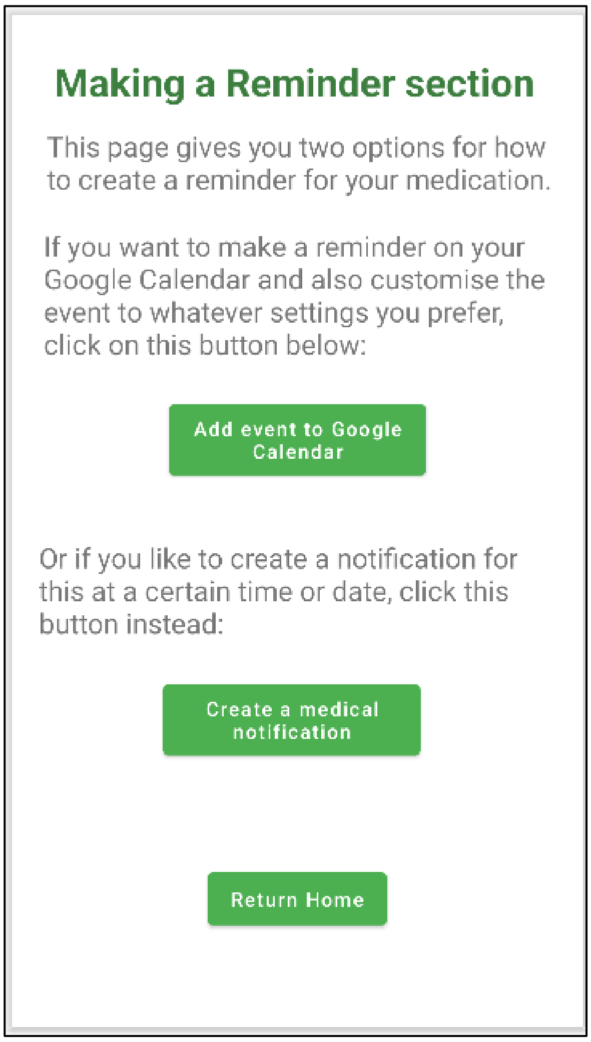


Figure 66: The text message maker page and a button to go back to the Add Contact List.

Add Reminder Page:

In the add reminder page, you'll be presented with two options, one where you can paste the medication text found in the Speech to Text page into a Google Calendar reminder and the other can allow you create a notification reminder that will activate when set to the desired time. The latter option was added to give the user some flexibility in terms of how they want to make a medication reminder for themselves.

In the former option, they'll taken to their Google Calendar and will be asked to sign into their Gmail to save and create a reminder on it. Fortunately, if you already signed in, you'll only be asked to do this once. The code in the Java page for AddReminder is designed to be set to a frequent reminder since realistically, medication rounds will happen in the span of a week or 2 depending on what their doctor/GP advised them.


Making a Reminder section

This page gives you two options for how to create a reminder for your medication.

If you want to make a reminder on your Google Calendar and also customise the event to whatever settings you prefer, click on this button below:

Or if you like to create a notification for this at a certain time or date, click this button instead:

Figure 67: The layout for the Reminder Creator page and the buttons provided.

Notification Creator Page:

When the user clicks on the button titled "Create a medical notification", they should see this page containing a reminder title and a Time Picker which is used to set the reminder to activate at a certain time (Trigger at 5:20pm for example). Once you're happy with everything, you can click on Finish to save the notification and it should activate at the specified time. The clock selector will function similar to the one found in the Real-time Bus App (Can be navigated from the Calendar Intent in Figure 67 and from the home page in Figure 59).

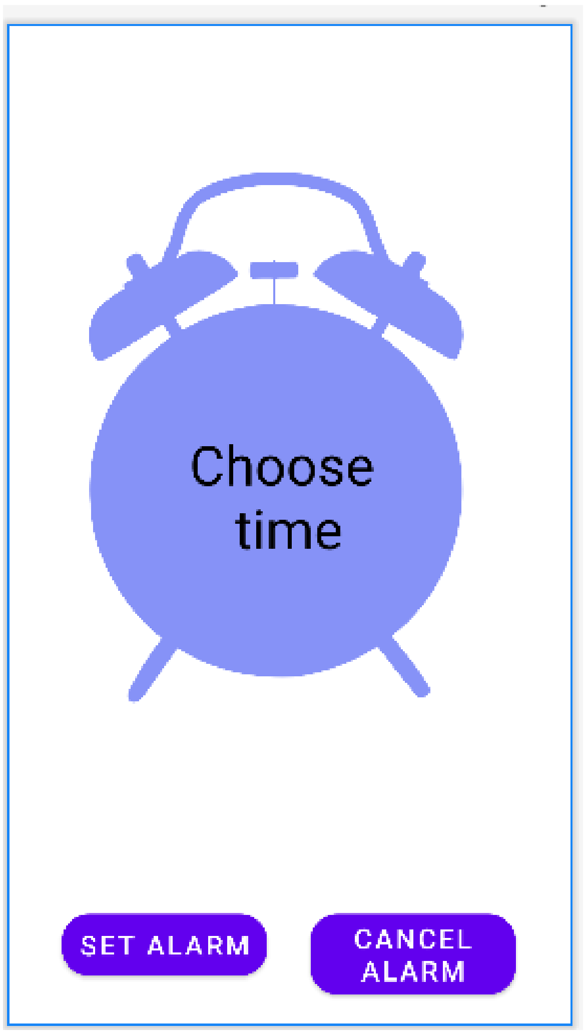


Figure 68: The page for creating a notification and the Time Picker used to create a timed alarm.

Add Diagnosis Page:

When the user clicks on the "Save Diagnosis" button in the home page, they'll be taken to this page where a user can record any known medical diagnosis/conditions into a table on the database. They can also remove an entry if they're in remission or no longer have said diagnosis.

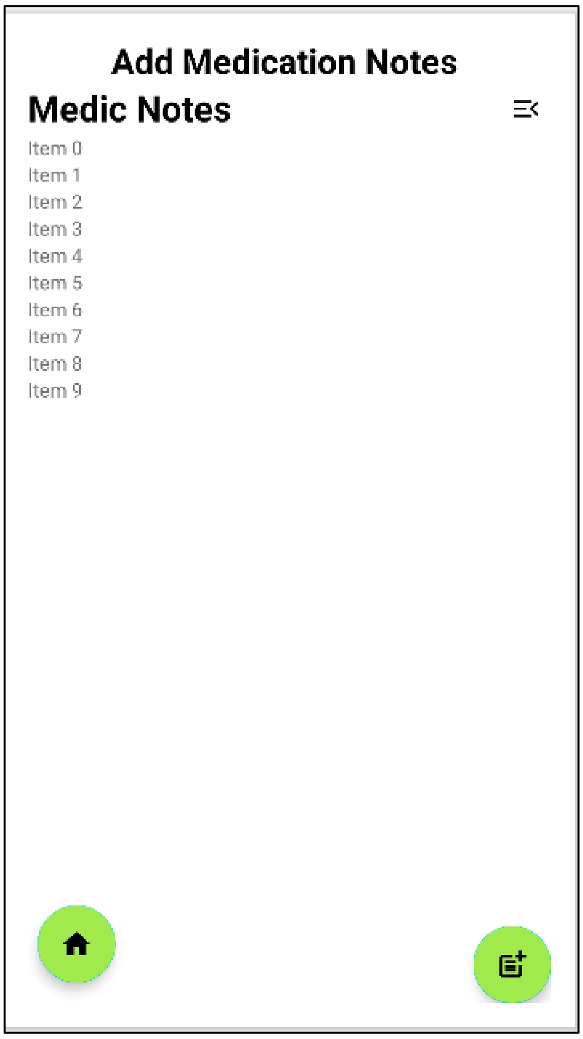
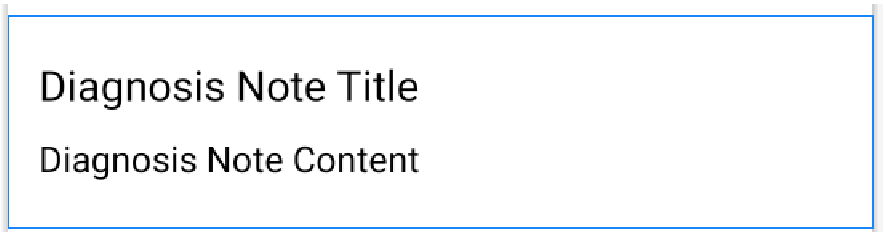


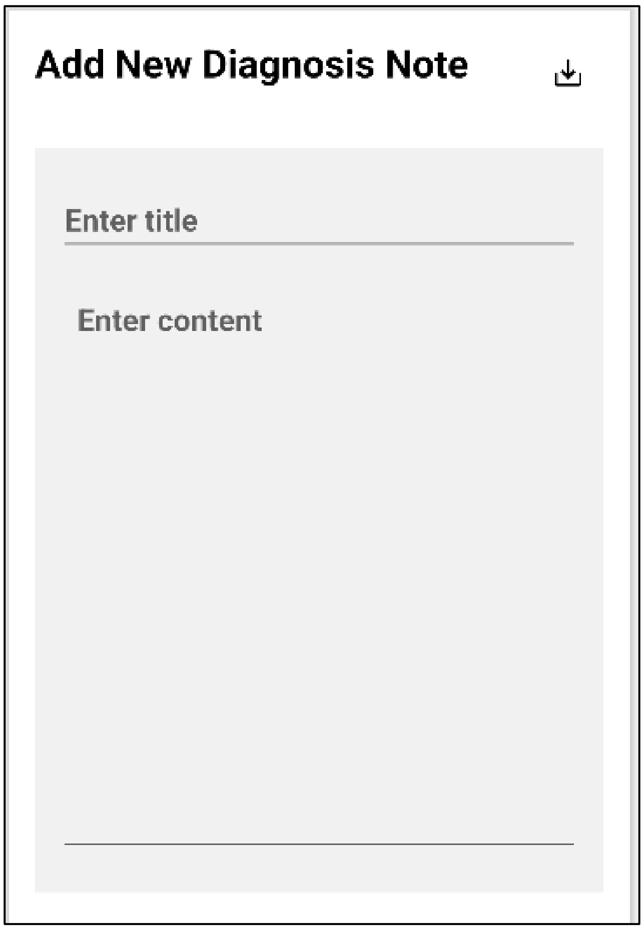
Figure 69: The page for adding medical diagnoses to the FireStore Database.

After clicking the add button, the user should see a pop up where they must enter a medical diagnosis they may have.


Diagnosis Note Title

Diagnosis Note Content

Figure 70: How a saved diagnosis entry will look when saved to the list


Add New Diagnosis Note

Enter title

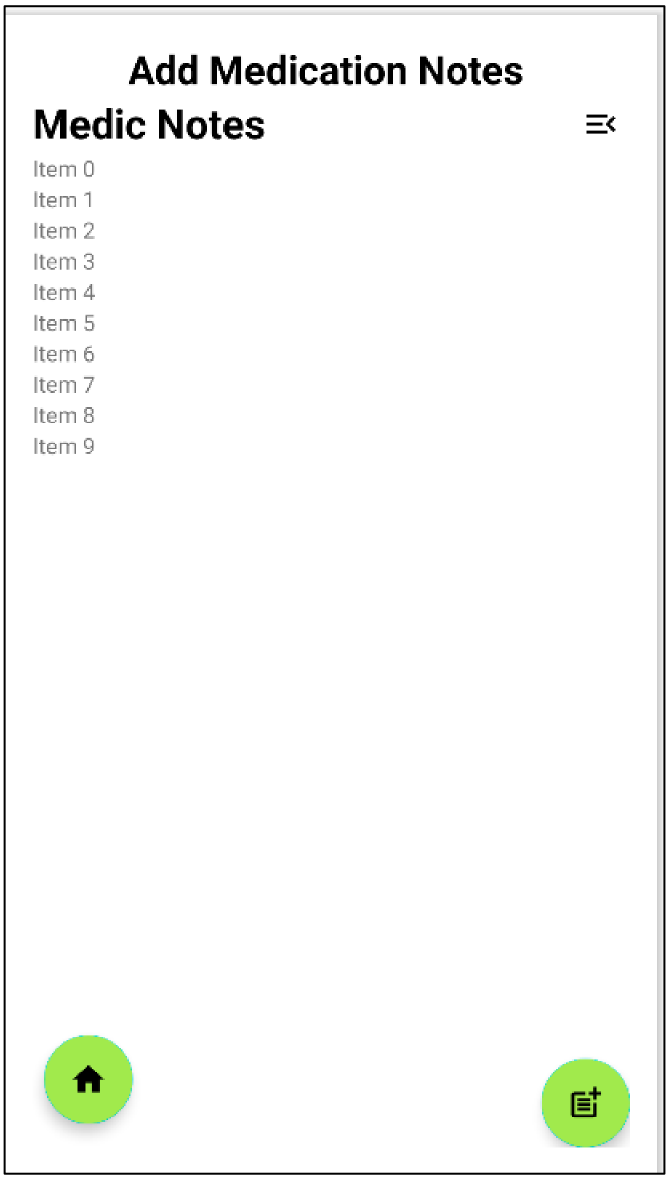
Enter content

Figure 71: The popup used to save a diagnosis entry to the database.

If a user wants to delete a diagnosis (if a user has had cancer and are in remission now for example), the message found at the bottom of the page will be used to confirm if the user wants to go through with this. If they click it, the diagnosis note will be permanently deleted, if they choose not to, the note will remain in the list.

Add Medication Page:

When the user clicks on the plus button in the home page, they'll be taken to this page where a user can record any medication and its instructions into a table on the database. They can also remove an entry if they're no longer taking said medication.


Add Medication Notes

Medic Notes

Item 0
Item 1
Item 2
Item 3
Item 4
Item 5
Item 6
Item 7
Item 8
Item 9

Figure 72: The page for adding medication information to the FireStore Database.

After clicking the add button, the user should see a pop up where they must enter a prescribed medication they may be taking, and the instructions assigned to them.

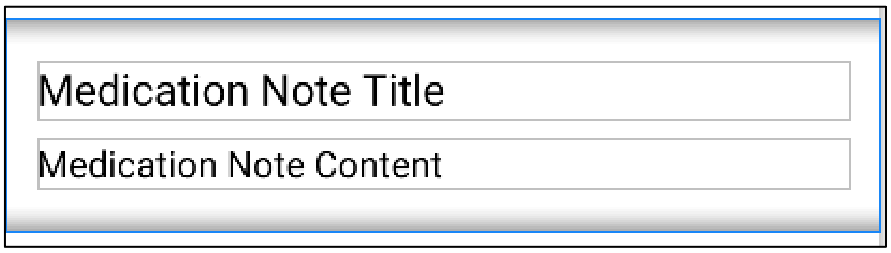


Figure 73: The pop up that users will use to save medication information to the database.

Saving the entry will lead to a list object created to be added to the Medication Entry List.

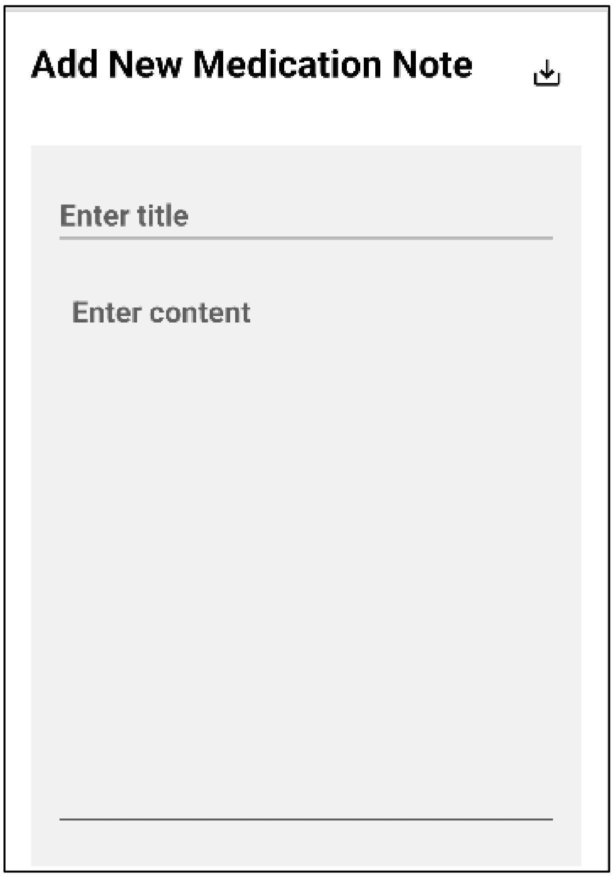


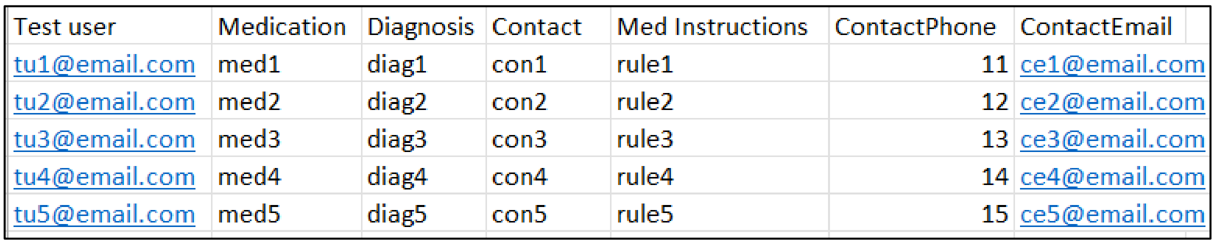
Figure 74: The medication entry list that will be shown when an entry has been saved to its respective table in the database.

If a user wants to delete a medication (if they're no longer taking it for example), the message found at the bottom of the page will be used to confirm if the user wants to go through with this. If they click it, the diagnosis note will be permanently deleted, if they choose not to, the note will remain in the list.

2.5 Testing

For the test plan, I tested the CRUD functionalities (adding notes of medication and diagnosis information and contact details), the Google Calendar Intent and finally the reminder alarm creator.

CRUD functionalities were tested via White-Box testing which involves executing all program statements. I started by setting up 5 examples of test users with made up account details and CRUD note information as seen in this screenshot below.

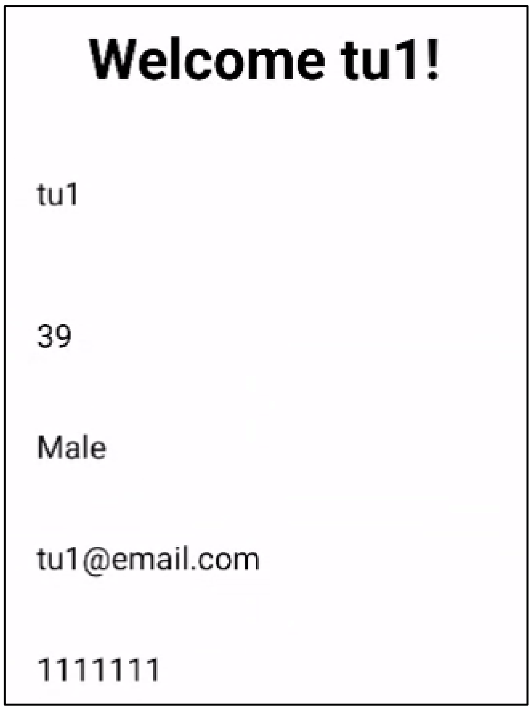


Test user	Medication	Diagnosis	Contact	Med Instructions	ContactPhone	ContactEmail
tu1@email.com	med1	diag1	con1	rule1	11	ce1@email.com
tu2@email.com	med2	diag2	con2	rule2	12	ce2@email.com
tu3@email.com	med3	diag3	con3	rule3	13	ce3@email.com
tu4@email.com	med4	diag4	con4	rule4	14	ce4@email.com
tu5@email.com	med5	diag5	con5	rule5	15	ce5@email.com

Figure 75: Series of test user cases and test data

Test case 1:

User profile:



Welcome tu1!

tu1

39

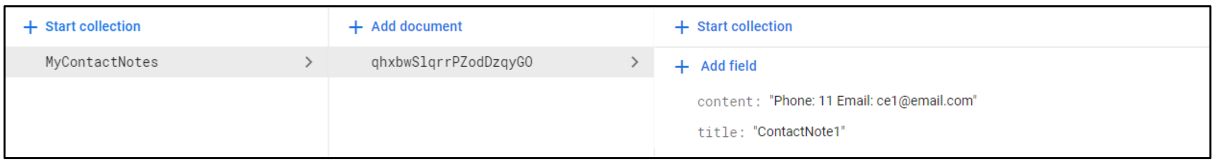
Male

tu1@email.com

1111111

Figure 76: Test user 1 profile with confirmed details

Contact note:



MyContactNotes

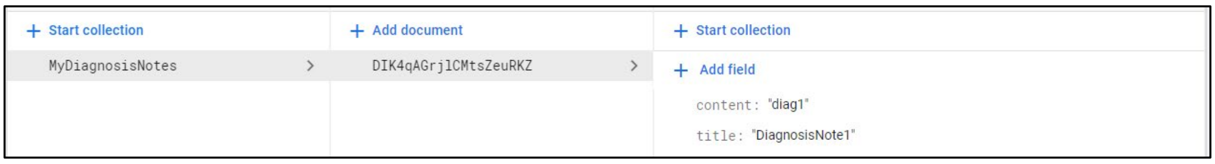
qhxbwSlqrrPZodDzqyG0

content: "Phone: 11 Email: ce1@email.com"

title: "ContactNote1"

Figure 77: Saved contact note for Test user case 1.

Diagnosis note:



MyDiagnosisNotes

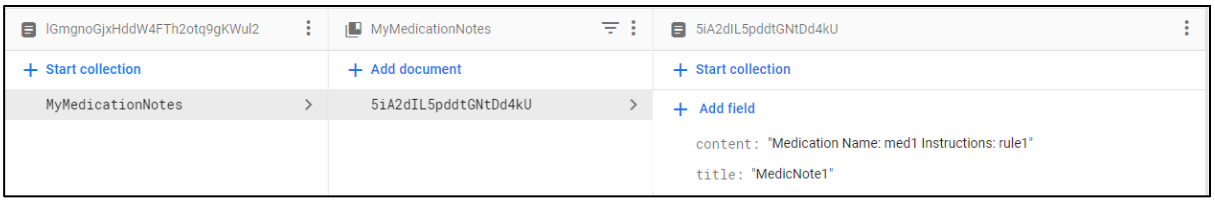
DIK4qAGrj1CMtsZeuRKZ

content: "diag1"

title: "DiagnosisNote1"

Figure 78: Saved diagnosis note for Test user case 1.

Medication note:



IGmgnoGjxHddW4FTh2otq9gKWul2

MyMedicationNotes

5iA2dIL5pddtGNtDd4kU

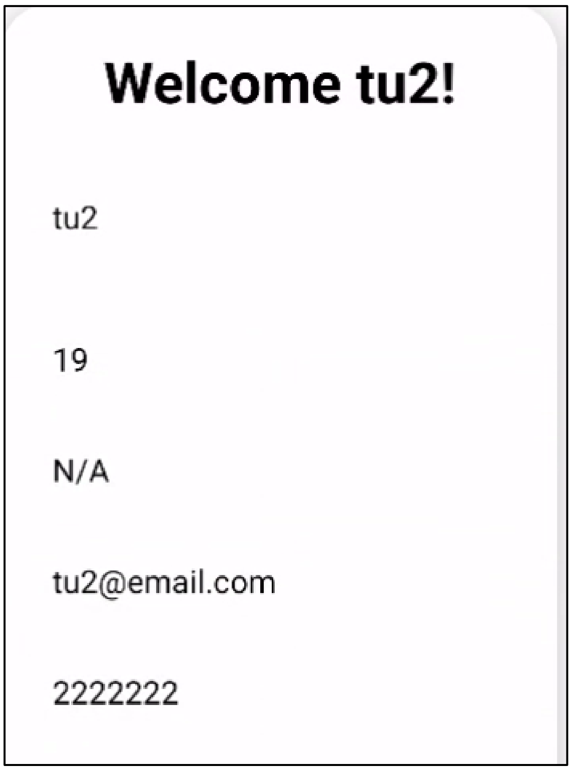
content: "Medication Name: med1 Instructions: rule1"

title: "MedicNote1"

Figure 79: Saved diagnosis note for Test user case 1.

<u>Test case 2:</u>

User profile:



Welcome tu2!

tu2

19

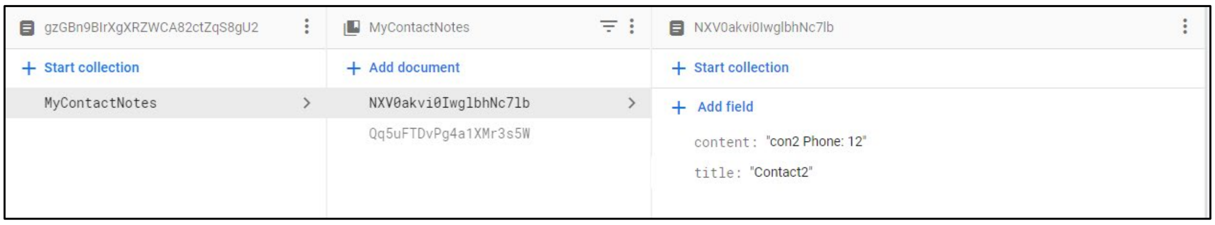
N/A

tu2@email.com

2222222

Figure 80: Saved user profile for Test user case 2.

Contact note:



gzGBn9BIrXgXRZWCA82ctZqS8gU2 | MyContactNotes | NXV0akvi0IwglbhNc7lb

+ Start collection | + Add document | + Start collection

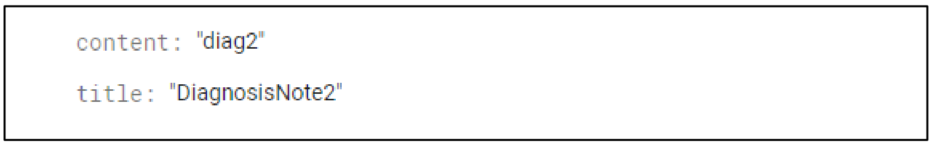
MyContactNotes | NXV0akvi0IwglbhNc7lb | + Add field

Qq5uFTDvPg4a1XMr3s5W

```
content: "con2 Phone: 12"
title: "Contact2"
```

Figure 81: Saved contact note for Test user case 2.

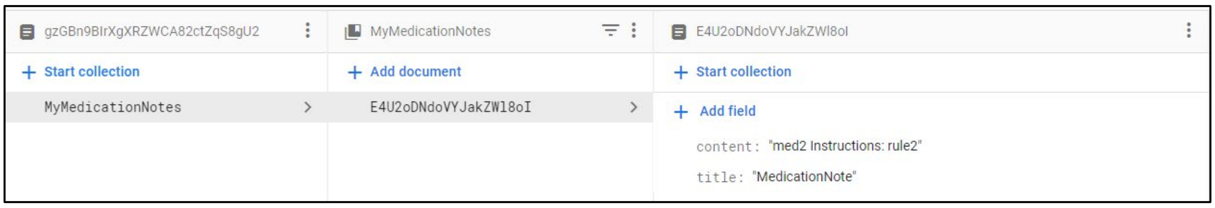
Diagnosis note:



```
content: "diag2"
title: "DiagnosisNote2"
```

Figure 82: Saved diagnosis note for Test user case 2.

Medication note:



gzGBn9BIrXgXRZWCA82ctZqS8gU2 | MyMedicationNotes | E4U2oDNdoVYJakZWl8oI

+ Start collection | + Add document | + Start collection

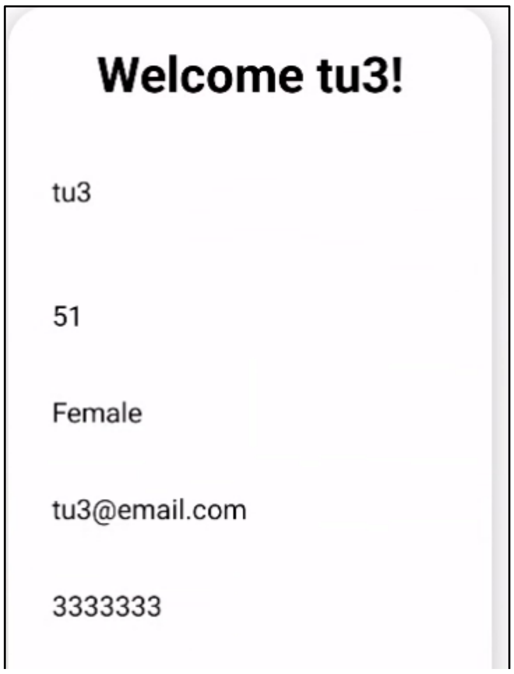
MyMedicationNotes | E4U2oDNdoVYJakZWl8oI | + Add field

```
content: "med2 Instructions: rule2"
title: "MedicationNote"
```

Figure 83: Saved medication note for Test user case 2.

Test case 3:

User profile:


Welcome tu3!

tu3

51

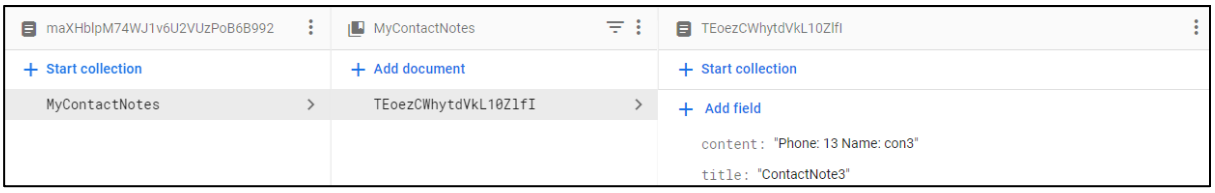
Female

tu3@email.com

3333333

Figure 84: User profile for Test user case 3.

Contact note:


maXHblpM74WJ1v6U2VUzPoB6B992 | MyContactNotes | TEoezCWhytdVkL10ZIfI

+ Start collection | + Add document | + Start collection

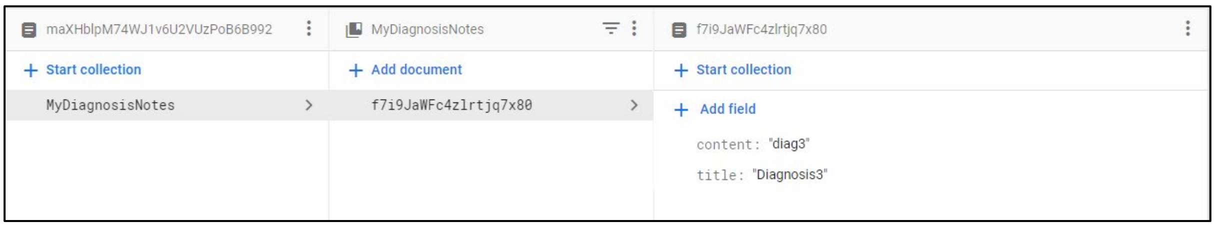
MyContactNotes > | TEoezCWhytdVkL10ZlfI > | + Add field

content: "Phone: 13 Name: con3"

title: "ContactNote3"

Figure 85: Saved contact note for Test user case 3.

Diagnosis note:


maXHblpM74WJ1v6U2VUzPoB6B992 | MyDiagnosisNotes | f7i9JaWFc4zlrtjq7x80

+ Start collection | + Add document | + Start collection

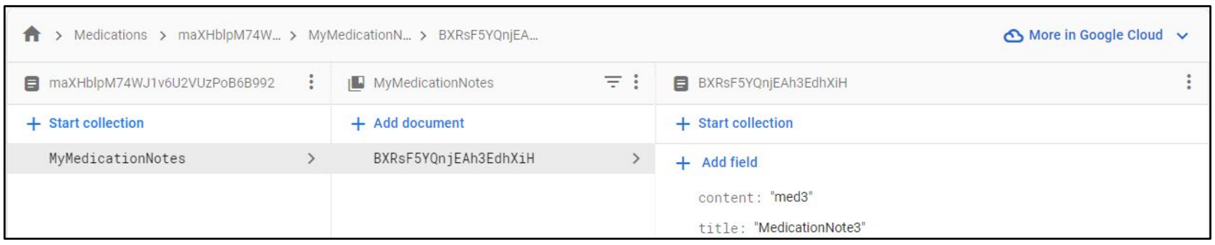
MyDiagnosisNotes > | f7i9JaWFc4zlrtjq7x80 > | + Add field

content: "diag3"

title: "Diagnosis3"

Figure 86: Saved diagnosis note for Test user case 3.

Medication note:


Medications > maXHblpM74W... > MyMedicationN... > BXRsF5YQnjEA...

More in Google Cloud

maXHblpM74WJ1v6U2VUzPoB6B992 | MyMedicationNotes | BXRsF5YQnjEAh3EdhXiH

+ Start collection | + Add document | + Start collection

MyMedicationNotes | BXRsF5YQnjEAh3EdhXiH | + Add field

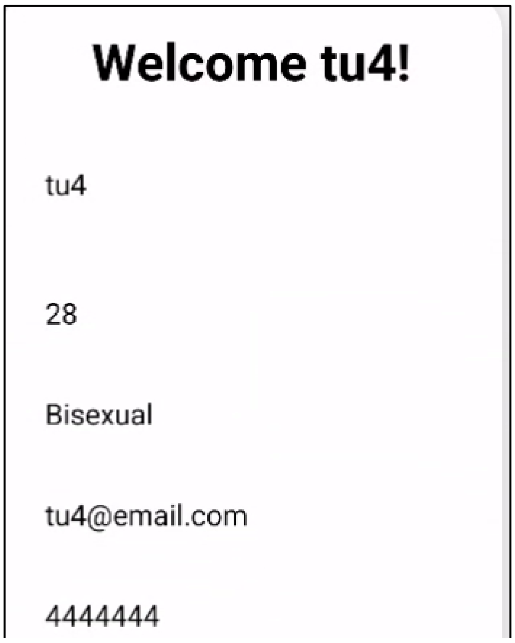
content: "med3"

title: "MedicationNote3"

Figure 87: Saved medication note for Test user case 3.

<u>Test case 4:</u>

User profile:


Welcome tu4!

tu4

28

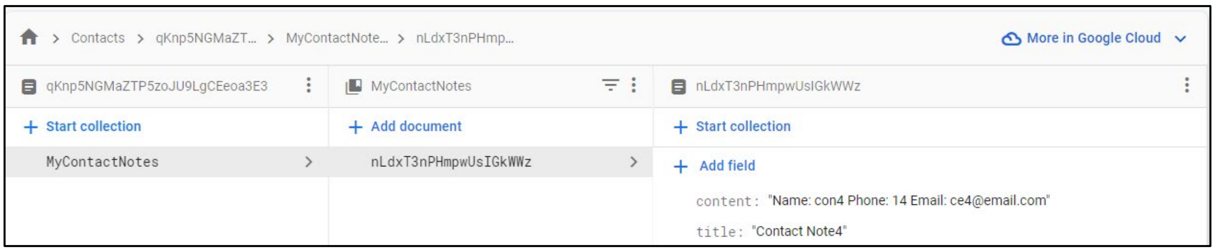
Bisexual

tu4@email.com

4444444

Figure 88: User profile for Test user case 4.

Contact note:


Contacts > qKnp5NGMaZT... > MyContactNote... > nLdxT3nPHmp...

More in Google Cloud

qKnp5NGMaZTP5zoJU9LgCEeoa3E3 | MyContactNotes | nLdxT3nPHmpwUsIGkWWz

+ Start collection | + Add document | + Start collection

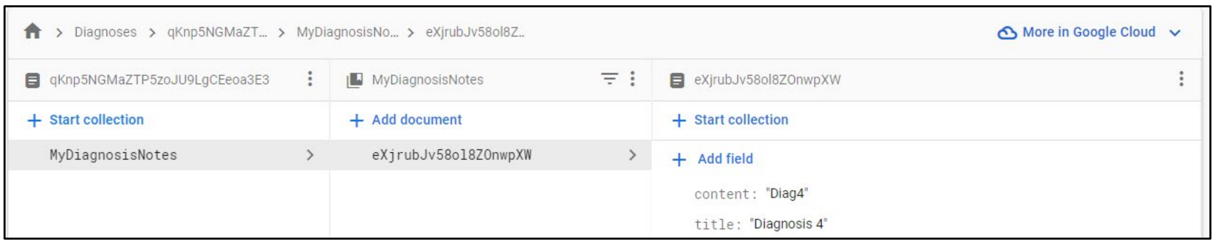
MyContactNotes | nLdxT3nPHmpwUsIGkWWz | + Add field

content: "Name: con4 Phone: 14 Email: ce4@email.com"

title: "Contact Note4"

Figure 89: Saved contact note for Test user case 4.

Diagnosis note:


Diagnoses > qKnp5NGMaZT... > MyDiagnosisNo... > eXjrubJv58ol8Z..

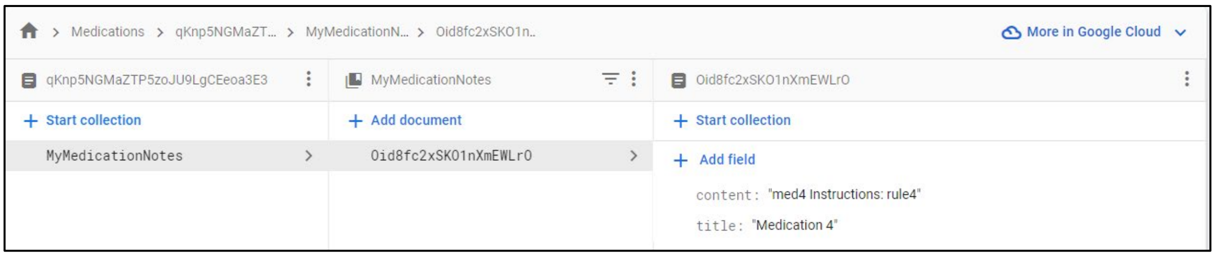
qKnp5NGMaZTP5zoJU9LgCEeoa3E3 | MyDiagnosisNotes | eXjrubJv58ol8ZOnwpXW

MyDiagnosisNotes | eXjrubJv58ol8ZOnwpXW

content: "Diag4"
title: "Diagnosis 4"

Figure 90: Saved diagnosis note for Test user case 4.

Medication note:


Medications > qKnp5NGMaZT... > MyMedicationN... > Oid8fc2xSKO1n..

qKnp5NGMaZTP5zoJU9LgCEeoa3E3 | MyMedicationNotes | Oid8fc2xSKO1nXmEWLrO

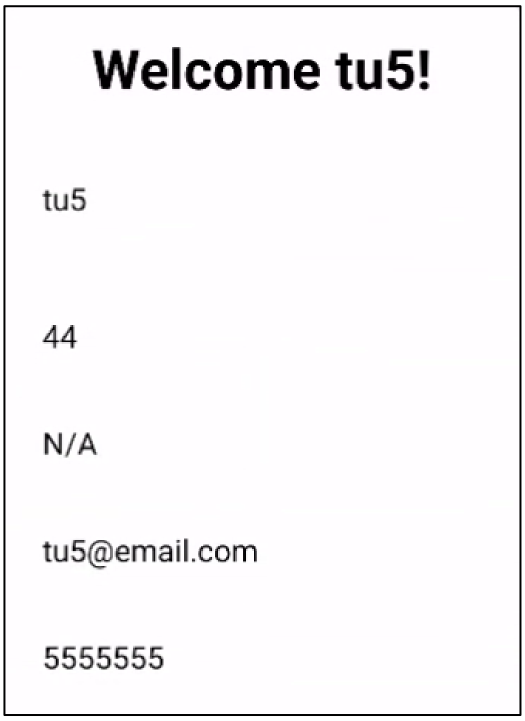
MyMedicationNotes | Oid8fc2xSKO1nXmEWLrO

content: "med4 Instructions: rule4"
title: "Medication 4"

Figure 91: Saved medication note for Test user case 4.

<u>Test case 5:</u>

User profile:


Welcome tu5!

tu5

44

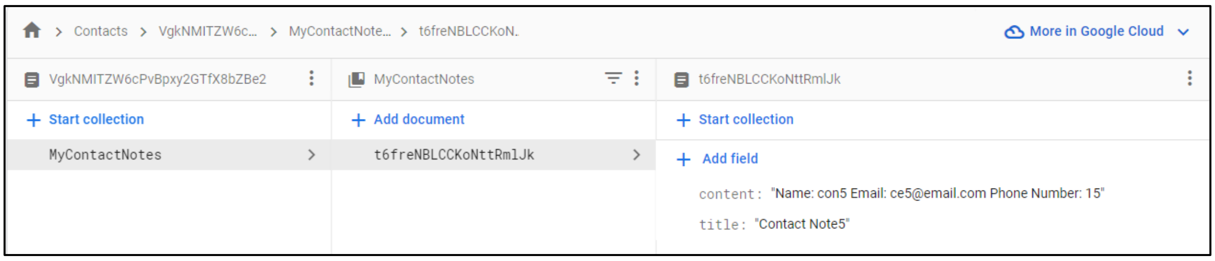
N/A

tu5@email.com

5555555

Figure 92: User profile for Test user case 5.

Contact note:


Contacts > VgkNMITZW6c... > MyContactNote... > t6freNBLCCKoN..

More in Google Cloud

VgkNMITZW6cPvBpxy2GTfX8bZBe2

+ Start collection

MyContactNotes >

MyContactNotes

+ Add document

t6freNBLCCKoNttRmlJk >

t6freNBLCCKoNttRmlJk

+ Start collection

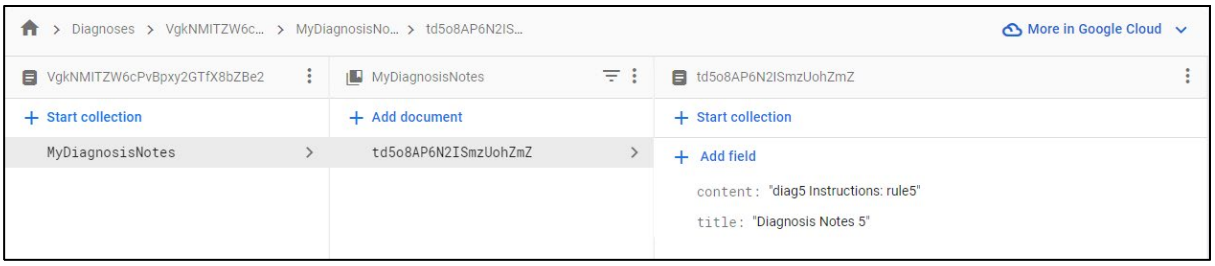
+ Add field

content: "Name: con5 Email: ce5@email.com Phone Number: 15"

title: "Contact Note5"

Figure 93: Saved contact note for Test user case 5.

Diagnosis note:


Diagnoses > VgkNMITZW6c... > MyDiagnosisNo... > td5o8AP6N2IS...

More in Google Cloud

VgkNMITZW6cPvBpxy2GTfX8bZBe2

+ Start collection

MyDiagnosisNotes >

MyDiagnosisNotes

+ Add document

td5o8AP6N2ISmzUohZmZ >

td5o8AP6N2ISmzUohZmZ

+ Start collection

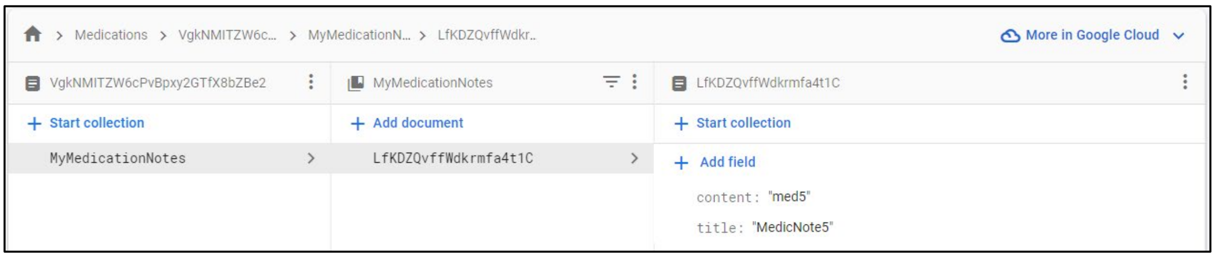
+ Add field

content: "diag5 Instructions: rule5"

title: "Diagnosis Notes 5"

Figure 94: Saved diagnosis note for Test user case 5.

Medication note:


Medications > VgkNMITZW6c... > MyMedicationN... > LfKDZQvffWdkr..

More in Google Cloud

VgkNMITZW6cPvBpxy2GTfX8bZBe2

+ Start collection

MyMedicationNotes >

MyMedicationNotes

+ Add document

LfKDZQvffWdkrmfa4t1C >

LfKDZQvffWdkrmfa4t1C

+ Start collection

+ Add field

content: "med5"

title: "MedicNote5"

Figure 95: Saved medication note for Test user case 5.

I also used Espresso to record functional tests. These involved starting from the login and registration pages and creating some test examples for the note maker functions, creating a reminder with the Google Calendar intent and editing a user profile.

In Figure 96, the following generated code from the 1st recording illustrates some of the saved edits made to the user profile which also reflected in the Firebase user table entry.

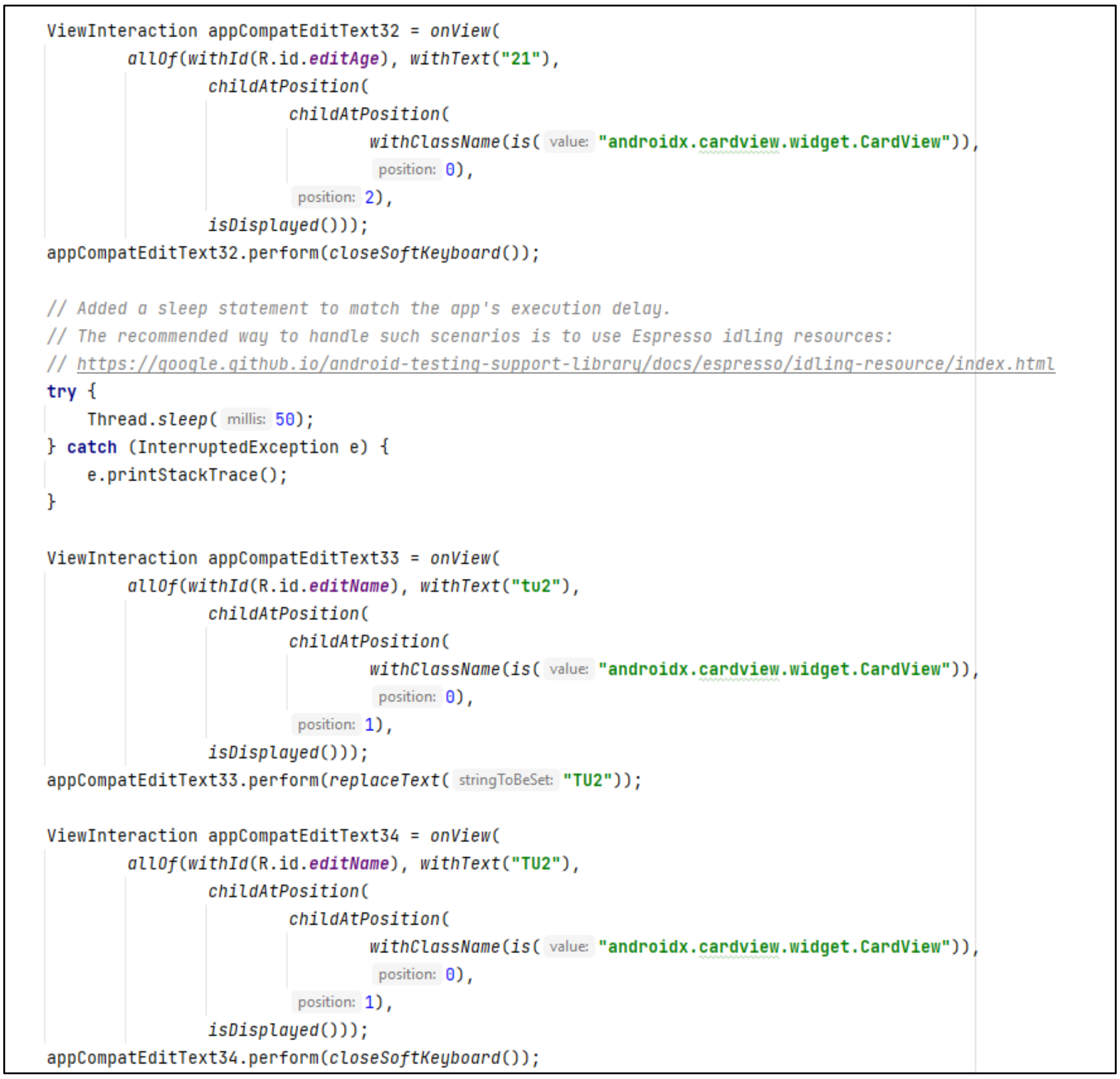

```
ViewInteraction appCompatEditText32 = onView(
        allOf(withId(R.id.editAge), withText("21"),
                childAtPosition(
                        childAtPosition(
                                withClassName(is( value: "androidx.cardview.widget.CardView")),
                                 position: 0),
                         position: 2),
                isDisplayed()));
appCompatEditText32.perform(closeSoftKeyboard());

// Added a sleep statement to match the app's execution delay.
// The recommended way to handle such scenarios is to use Espresso idling resources:
// https://google.github.io/android-testing-support-library/docs/espresso/idling-resource/index.html
try {
    Thread.sleep( millis: 50);
} catch (InterruptedException e) {
    e.printStackTrace();
}

ViewInteraction appCompatEditText33 = onView(
        allOf(withId(R.id.editName), withText("tu2"),
                childAtPosition(
                        childAtPosition(
                                withClassName(is( value: "androidx.cardview.widget.CardView")),
                                 position: 0),
                         position: 1),
                isDisplayed()));
appCompatEditText33.perform(replaceText( stringToBeSet: "TU2"));

ViewInteraction appCompatEditText34 = onView(
        allOf(withId(R.id.editName), withText("TU2"),
                childAtPosition(
                        childAtPosition(
                                withClassName(is( value: "androidx.cardview.widget.CardView")),
                                 position: 0),
                         position: 1),
                isDisplayed()));
appCompatEditText34.perform(closeSoftKeyboard());
```

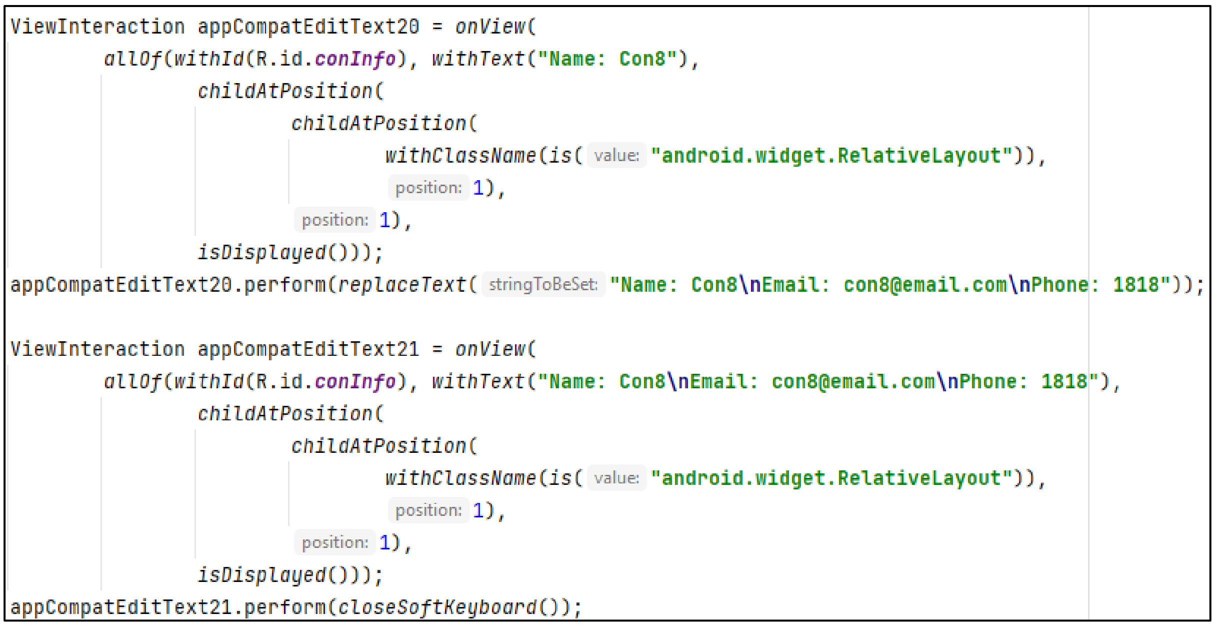
Figure 96: Profile entry edits made in the first Espresso recording.

And with the 2nd Espresso recording, according to Figure 97, the following code in the associated screenshot will replace the blank page entry with the confirmed note information for saving a contact note's title and details. This logic will also apply to the other CRUD note functionalities (Incidentally, medication and diagnosis all have identical functions to the contact note one).

I also carried out White Box Testing throughout the development of my application. This in turn consisted of testing out the application on my own phone and trying out the functions one at a time based on high and low priorities respectively.

Each time I added a new function page, I'd always test it to for any bugs, errors and if the code is carrying out the way it's declared in the application's code schematics.

During one of my Project supports, my lecturer noted how with RealTime Database, the CRUD entries saved from past user sessions would carry over each time, which was a close call since had I not noticed the error sooner, it could've potentially caused some dire cybersecurity risks.



```
ViewInteraction appCompatEditText20 = onView(
        allOf(withId(R.id.conInfo), withText("Name: Con8"),
                childAtPosition(
                        childAtPosition(
                                withClassName(is( value: "android.widget.RelativeLayout")),
                                 position: 1),
                         position: 1),
                isDisplayed()));
appCompatEditText20.perform(replaceText( stringToBeSet: "Name: Con8\nEmail: con8@email.com\nPhone: 1818"));

ViewInteraction appCompatEditText21 = onView(
        allOf(withId(R.id.conInfo), withText("Name: Con8\nEmail: con8@email.com\nPhone: 1818"),
                childAtPosition(
                        childAtPosition(
                                withClassName(is( value: "android.widget.RelativeLayout")),
                                 position: 1),
                         position: 1),
                isDisplayed()));
appCompatEditText21.perform(closeSoftKeyboard());
```

Figure 97: Added contact note and designated content of note body from the second Espresso recording.

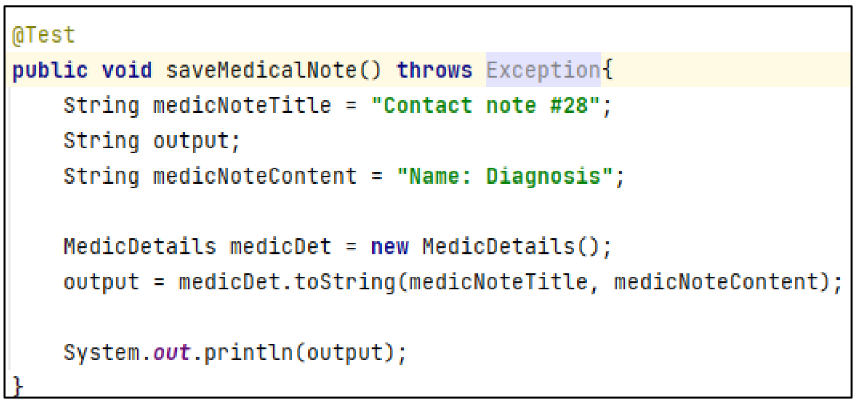
I also carried out White Box Testing throughout the development of my application. This in turn consisted of testing out the application on my own phone and trying out the functions one at a time based on high and low priorities respectively.

Each time I added a new function page, I'd always test it to for any bugs, errors and if the code is carrying out the way it's declared in the application's code schematics.

During one of my Project supports, my lecturer noted how with RealTime Database, the CRUD entries saved from past user sessions would carry over each time, which was a close call since had I not noticed the error sooner, it could've potentially caused some dire cybersecurity risks.

Unit testing involved testing out a few core functionalities with minor code modifications. For all 3 CRUD functionality pages, I decided to test out the method involving saving CRUD notes and returning them as a complete String object via the combining of both a note's title and its content.

The typical test case I coded out in Android Studio will look something like this:



```
@Test
public void saveMedicalNote() throws Exception{
    String medicNoteTitle = "Contact note #28";
    String output;
    String medicNoteContent = "Name: Diagnosis";

    MedicDetails medicDet = new MedicDetails();
    output = medicDet.toString(medicNoteTitle, medicNoteContent);

    System.out.println(output);
}
```

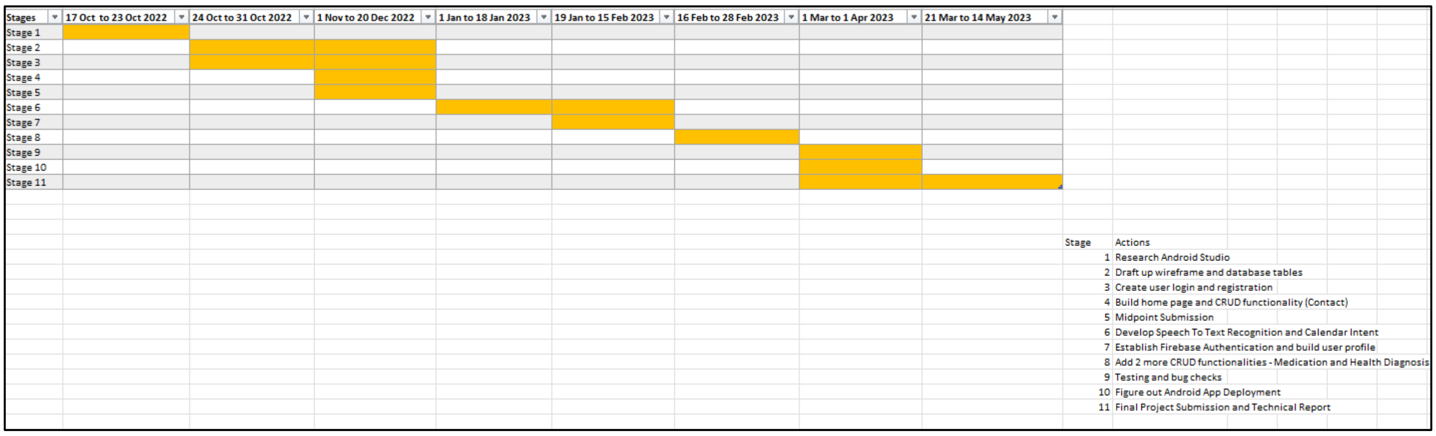
Figure 98: Test case example for CRUD Note

The basic idea in this test method is to use pre-written note information and combine them into a single String. When I run the test case, it will trigger the combined note content and title as one String value.

2.6 Evaluation

After conducting the 5 test cases, the 3 CRUD functionalities are working as they were intended and coded out. I also tested out the speech to text recogniser and the Calendar objects. For the email maker and Calendar Intent, the user will be prompted to log into their Gmail accounts for these functions, but they'd be only asked to do this once.

The alarm reminder function took some time to implement and code since the reminder will trigger 2 minutes after the intended time but I figured out from a tutorial about the functionality, that the reminder will still trigger even if it's in-between the seconds of a specified time; like how if the reminder is set for 12:00pm, the alarm will still trigger if it's 30 seconds after 12pm.



Stages	17 Oct to 23 Oct 2022	24 Oct to 31 Oct 2022	1 Nov to 20 Dec 2022	1 Jan to 18 Jan 2023	19 Jan to 15 Feb 2023	16 Feb to 28 Feb 2023	1 Mar to 1 Apr 2023	21 Mar to 14 May 2023
Stage 1	🟨							
Stage 2		🟨	🟨					
Stage 3		🟨	🟨					
Stage 4			🟨					
Stage 5			🟨					
Stage 6				🟨	🟨			
Stage 7					🟨			
Stage 8						🟨		
Stage 9							🟨	
Stage 10							🟨	
Stage 11							🟨	🟨

Stage	Actions
1	Research Android Studio
2	Draft up wireframe and database tables
3	Create user login and registration
4	Build home page and CRUD functionality (Contact)
5	Midpoint Submission
6	Develop Speech To Text Recognition and Calendar Intent
7	Establish Firebase Authentication and build user profile
8	Add 2 more CRUD functionalities - Medication and Health Diagnosis
9	Testing and bug checks
10	Figure out Android App Deployment
11	Final Project Submission and Technical Report

Figure 99: Project timeline and stages

I also used Snyk, which is a website that, according to the company's website (Snyk, 2023) allows users to import their application's code and cloud environments to scan and fix any security vulnerabilities and known issues found in the code. Through an analysis of

MedsUpApp's Git repository, it was discovered that there no issues in the project's code files, Gradle setups and dependency builds.

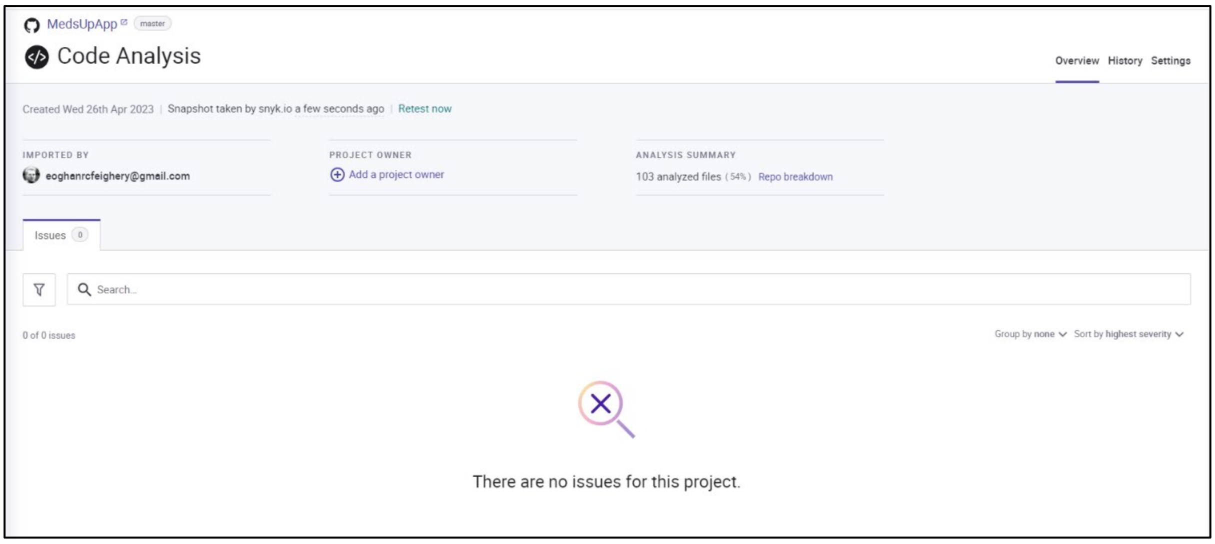

MedsUpApp master
Code Analysis
Overview History Settings
Created Wed 26th Apr 2023 | Snapshot taken by snyk.io a few seconds ago | Retest now
IMPORTED BY eoghanrcfeighery@gmail.com
PROJECT OWNER Add a project owner
ANALYSIS SUMMARY 103 analyzed files (54%) Repo breakdown
Issues 0
Search...
0 of 0 issues
Group by none Sort by highest severity
There are no issues for this project.

Figure 100: Code analysis results from Snyk.

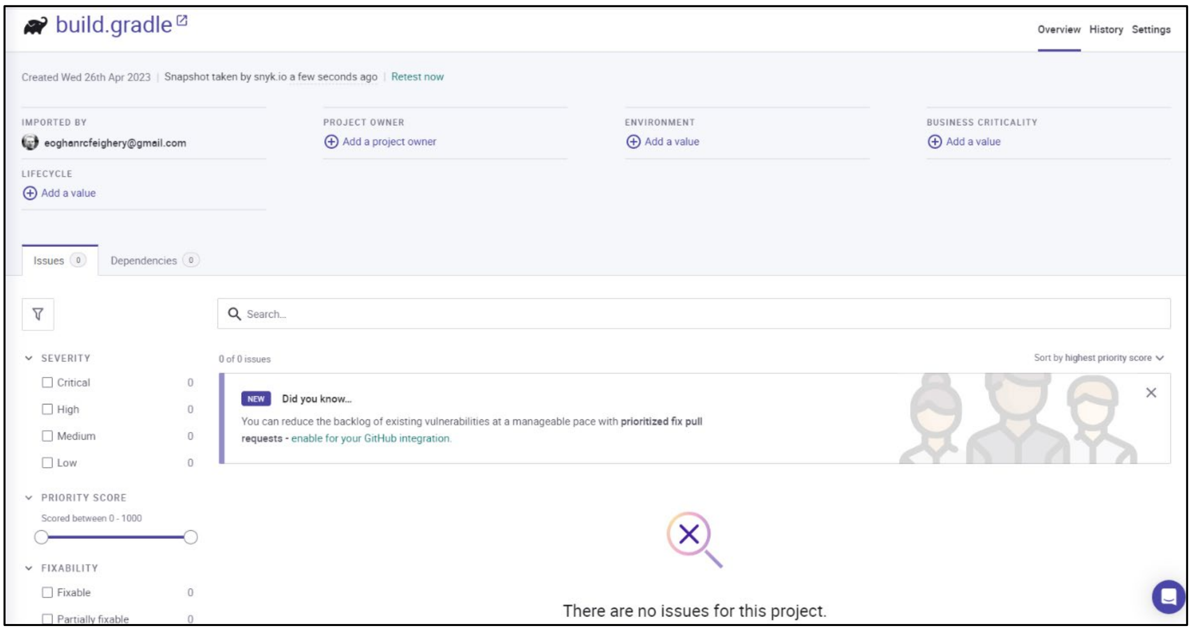

build.gradle
Overview History Settings
Created Wed 26th Apr 2023 | Snapshot taken by snyk.io a few seconds ago | Retest now
IMPORTED BY eoghanrcfeighery@gmail.com
PROJECT OWNER Add a project owner
ENVIRONMENT Add a value
BUSINESS CRITICALITY Add a value
LIFECYCLE Add a value
Issues 0 Dependencies 0
Search...
SEVERITY
Critical 0
High 0
Medium 0
Low 0
PRIORITY SCORE
Scored between 0 - 1000
FIXABILITY
Fixable 0
Partially fixable 0
0 of 0 issues
Sort by highest priority score
NEW Did you know...
You can reduce the backlog of existing vulnerabilities at a manageable pace with prioritized fix pull requests - enable for your GitHub integration.
There are no issues for this project.

Figure 101: Gradle build analysis result from Snyk.

In the Profiler screenshot down below, the application consumes around 256 megabytes worth of memory while the amount of energy consumption is on a medium-scale at the start of the application working but the more time spent on the application, the energy consumed grows smaller. As such, the energy gauge ranges from medium (application session start) to light (middle and end of application session). CPU consumption is on a similar observation since at the start of the application working, it goes from 21% to 0% which wildly varies depending on the functionality that the user will trigger.

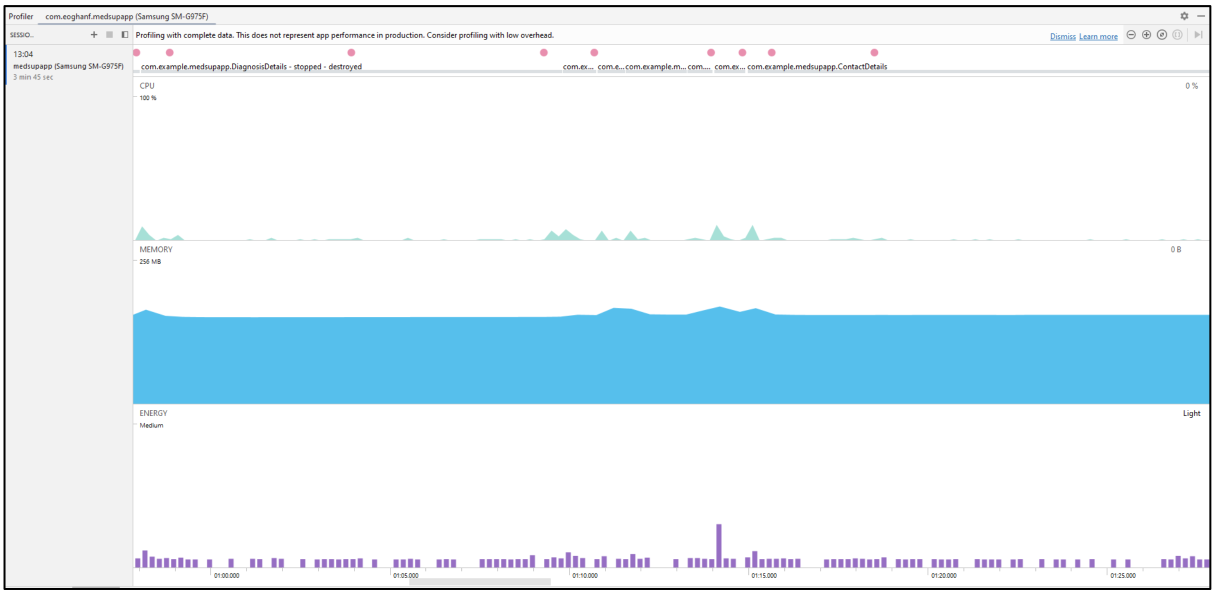

Profiler
com.eoghanf.medsupapp (Samsung SM-G975F)
SESSIO...
Profiling with complete data. This does not represent app performance in production. Consider profiling with low overhead.
Dismiss
Learn more
13:04
medsupapp (Samsung SM-G975F)
3 min 45 sec
com.example.medsupapp.DiagnosisDetails - stopped - destroyed
com.ex... com.e... com.example.m... com.... com.ex... com.example.medsupapp.ContactDetails
CPU
100 %
0 %
MEMORY
256 MB
0 B
ENERGY
Medium
Light
01:00.000
01:05.000
01:10.000
01:15.000
01:20.000
01:25.000

Figure 102: Profiler charter for MedsUpApp.

3 Conclusions

The advantage to using Android Studio was that I was able to better utilise my knowledge of Java since the former has a lot of similar code methods, I've studied previously such as instantiable classes converted into ArrayList objects. On top of that, I've had a lot of creative liberty in placing certain visual features and deciding on a more consistent color theme. This is why the app's color scheme mostly consisted of white and green, since they're a constant color scheme associated with hospitals and chemists.

The biggest limitation with Android Studio is that when it came to developing the application, it seems very delicate since one tiny change can lead to the entire project having errors in a few areas. On top of that, I had to work under some design constraints. For example, for the Add Contact page, the tutorial I found as part of my research used a LinearLayout format for the page's layout so that meant I had to work within that confinement which is why the buttons I added had to remain at the bottom or it would cause the layout of the page to look out of sync position-wise.

Finding tutorials for the Firebase RealTime Database utilisation had some difficulty since certain videos I found on YouTube had methods that would be outdated or may not work when I tested out the code. Fortunately, I did find out that any deprecated code was able to function regardless. Not to mention because Realtime Database is a NoSQL database, I had to rely on other methods to allow the app to be able find the right table entries for both the user and the CRUD pages. Fortunately, according to Odhiambo (Odhiambo, 2022), there're quite a few workaround ways when it comes to sorting and filtering Firebase data. With the use of the orderByValue, orderByKey and orderByChild methods, the application can sort out the data by both using child and key values as a reference point. This in turn, gives it a sort of semblance of linked database tables.

But during the final stages of development, I found out that saved CRUD data entries from past user sessions were bleeding into each other and could potentially pose a cyber-security risk. After finding no video tutorials for configuring user IDs for CRUD data with RealTime Database, I eventually attempted a tutorial that used Cloud FireStore Database which thankfully managed to work in my favour.

Then for the user login, for some reason despite an account being saved onto the users table, the app would remain at the login screen with a notification saying to check user credentials. As such, I had to create an extra if else statement with the login confirmation code pasted inside which was the only way that it was guaranteed to work 100% of the time. The speech to text recognition works better when testing it out on actual Android phones as when it's done in emulators, it stops when I click on the microphone button.

Overall, the application's development history has had its fair share of peaks and valleys from conception to implementation, though if nothing else, I was able to develop a fully functioning app ready made for Android smartphones.

My application's Git repository is in this link:
https://github.com/Efeighery/MedsUpReminder

4 Further Development or Research

If I had extra time and resources for MedsUpApp, it would've taken a direction of being on both Android and IOS platforms for cross-technological accessibility. I also feel that it would be an interesting challenge to have implemented both speech-to-text and image-to-text recognition; the latter of which could work like the likes of Cam Scanner and/or Google Translates camera transcribe feature.

Additionally, I'd have more time to deploy it to Google Play Store in the near future since while I gave it a shot to be sure, I realised it would take a lot more time to do that but I'll put that on the back burner for the time being.

5 References

Firebase, G., 2023. *Firebase Realtime Database.* [Online]
Available at: https://firebase.google.com/docs/database
[Accessed 4 March 2023].

Medicare.org, 2023. *5 Best Pill Reminder Apps.* [Online]
Available at: https://www.medicare.org/articles/5-best-pill-reminder-apps/
[Accessed 1 May 2023].

MediSafe, 2023. *Medisafe Pill & Med Reminder.* [Online]
Available at:
https://play.google.com/store/apps/details?id=com.medisafe.android.client&hl=en&gl=US
[Accessed 1 May 2023].

MyTherapy, 2023. *MyTherapy Pill Reminder.* [Online]
Available at:
https://play.google.com/store/apps/details?id=eu.smartpatient.mytherapy&hl=en&gl=US
[Accessed 1 May 2023].

Odhiambo, B., 2022. *Implementing Custom Searching and Filtering in Firebase Database in Android.* [Online]
Available at: https://www.section.io/engineering-education/custom-searching-and-filtering-in-firebase-database-in-android/
[Accessed 14 February 2023].

Snyk, 2023. *Introducing Snyk.* [Online]
Available at: https://docs.snyk.io/getting-started/introducing-snyk
[Accessed 25 April 2023].

Treichler, C., 2022. *THE 10 BEST MEDICATION REMINDER APPS.* [Online]
Available at: https://www.onlinedoctor.com/best-medicine-reminder-apps
[Accessed 1 May 2023].

6 Appendices

6.1 Project Proposal

The main objective that I want to achieve with this project is to design a scanner for mobile devices that can scan the front of a prescribed medication (cream, pills, medicine) to produce a list of instructions for how and when to take them.

Another objective is to investigate how to figure out how I can create a feature that can add the list to the user's Google Calendar. Also, I would need to research how to program a notification feature and implement to function properly on after every 3 or 4 hours.

Finally, I want to be able to figure out what I can do to synchronise the reminder to the user's relatives where it can send a message to notify them that the former has or hasn't taken their medication yet.

That last functionality would need to be able to send an automatic message to their phones which can produce updates in the same time interval that I outlined in the second objective above.

I chose to undertake the project because I feel that, in the aftermath of the Covid-19 pandemic, people have become a lot more self-conscious about their health conditions and naturally, they have been a lot more worried for their elderly relatives. Another common aspect is that with everyone gradually returning to college, work and/or school after almost two years of quarantines.

With that busy state of mind, we also may end up forgetting to take any prescribed medication amidst the hustle and bustle of day-to-day life. In fact, if we had a little reminder about taking them every day, people wouldn't be as stressed about forgetting about it.

Since one of my aims is to create a camera scanner like the kind that can scan documents, I would need to research how I can build a scanner program to work for smartphones and look at how the code functions that way. Additionally, I plan to research how to program a reusable pop-up notification and figure out how I link it with the scanner application.

For the Google Calendar synchronisation, that also needs a look into online examples to see if it's possible to code up the share options that usually seen in messaging apps like WhatsApp. The automatic text message will need to be researched as well to see how it works.

6.2 Reflective Journals

Reflective Journal – October:

Supervision & Reflection Template

Student Name	Eoghan Feighery
Student Number	X19413886
Course	Software Development BSHCSD
Supervisor	Adriana Chis

What? Reflect on what has happened in your project this month? So far, I've submitted my Project pitch and have decided to use the Flutter technology for building my application. But after talking with the Computing Project support supervisor, I decided to choose Ionic as my official technology framework since I've more familiarity with CSS, JavaScript, and HTML. For the testing phase, I decided to have certain members of my family to try out the app and give feedback on how it functions, for any errors that still exists and if it does what it was designed to do. I used Figma to chart out how the functions will play out, what the pages will look like and how they all connect to each stage by stage.
So What?

Consider what that meant for your project progress. What were your successes? What challenges still remain? In terms of my project success, it means that I managed to make a starting point for the project. Not to mention, since Ionic is entirely based on recognisable programming languages and it does have some recognisable code structures. Of course, I still need to figure out how Ionic code is done and how an app for it is developed with that technology. On top of that, I need to figure out how to store phone numbers in a way that respects data privacy especially for underage and vulnerable users of the app. Then, I need to look into how the camera scanner since I did find a guide for developing a camera and photo gallery using Ionic.
Now What? What can you do to address outstanding challenges? To resolve the challenges, I need to discuss my project idea further with my supervisor about how I can sort out how to store phone numbers safely and securely, how to make the scanner work in conjunction with the app and figure how to implement the log in system and data encryption. For Ionic, I plan to have a look at a few tutorials and documentations based on Ionic so that I can have a clear idea of how I can approach the code the application and what io expect with using the technology. For the pop-up notification, I'll need to investigate how I can make it when it scans an item to produce a small list of instructions to be copied and shared with a device's calendar app. For now, though, I'll need to make an ERD before I get to the complex coding schematics.

Student Signature	Eoghan Feighery

Reflective Journal – November:

Supervision & Reflection Template

Student Name	Eoghan Feighery
Student Number	X19413886
Course	Software Development BSHCSD
Supervisor	Adriana Chis

What? Reflect on what has happened in your project this month? So far, I had a look at Ionic, but I was building the prototype for the app, I had some difficulties in trying to implement the OCR technology and the Firebase database. After my talk with my supervisor, she suggested to think over what you're more familiar with.

And after a little bit of digging and thinking, I decided to stick with Android Studio and Java, so that I don’t overthink on how my app will work on multiple platforms.	
So What? Of course, since I have a prototype that’s due a few days before Christmas, I’ll need to focus on getting the OCR technology to work whilst also implementing a bit of the functionalities. Fortunately, my past attempt in Ionic will suffice as a proof of concept. Since this is a little tight, I will need to prioritize it as very important especially for the video demonstration and other project commitments.	
Now What? What can you do to address outstanding challenges? To better address the aforementioned potential hurdles, I’ll focus on the camera OCR function first since this would be the more difficult functionality I’ll ever implement in this project. Then, I can add a little function to the calendar page since I found out that you can add events via Google Calendar Intent. As for the login credentials, I may need to move the age, gender and name into the User Profile and figure out how to edit a User Profile like a reusable form. While this does seem like a tight schedule, at the very least it’s still doable. In hindsight, I probably should’ve just stuck to Android Studio from the very start, but at least I have a clear view on the project now.	
Student Signature	Eoghan Feighery

Reflective Journal – December:

Supervision & Reflection Template

Student Name	Eoghan Feighery
Student Number	X19413886
Course	Software Development BSHCSD
Supervisor	Adriana Chis

What? Reflect on what has happened in your project this month? After a chat with the college's Computing Support lecturer, I had to rebuild the application from the ground up to include the Realtime Firebase database. While I had some issues with confirming log-in functionality, I was able to have a functional prototype with the CRUD functionality for adding and removing contact inputted by the user.
So What? After a little bit of digging, I found a series of videos that have more clearer and concise guidelines for the text recogniser camera function, the Calendar intent object, and the notification creator. For the latter two, it may be possible to have a page with two buttons; one can used to add the medication reminder to the Google Calendar (if applicable) and another can be used to set a time for when the user can take their medication and will produce a notification for this. I also found a 4-video part playlist on YouTube where it's possible to let users add events to a calendar and display the notifications like that of Google Calendar.
Now What? What can you do to address outstanding challenges? For the time being, I'll start with the text recognition functionality and the copy to clipboard combination since that feature is the main focal point of MedsUp App. I'll also make a spare copy of my current application build which can be used for the aforementioned notification and calendar intent functionality page and will asked my supervisor if this is feasible for the final build. For the other build I have so far, I can test the calendar tutorial I found to check if it won't cause any errors when I run the emulator in Android Studio.

Student Signature	Eoghan Feighery

Reflective Journal – January:

Supervision & Reflection Template

Student Name	Eoghan Feighery
Student Number	X19413886
Course	Software Development BSHCSD
Supervisor	Adriana Chis

What?

Reflect on what has happened in your project this month?

With the midpoint submission over and done with, I got to work in trying to get the image to text OCR function to work but unfortunately I wasn't able to get it running on the emulator. After some thinking, I remembered that there's a functionality that can let you speak into a microphone and can produce the text captures from your recording. I tested it out on my phone and to my delight, it was working in the way that it should plus it can let users edit the text if a GP gave some changes to a medication prescription.

Additionally, I found out that people can add events to Google Calendar and compose emails with Intent objects. This was a bit of an awakening to me since it can allow reminders to be added in a more consistent fashion. Also, I added a timer as an alternative when people want to set a reminder without Google Calendar and added a splash screen to my app for visual fidelity.

So What?

How I got the speech to text recogniser functionality to work is that I found a tutorial that was created recently and then tried out a test version of it on my phone. Luckily it was functioning very well, and the text result can even be copied to the clipboard by holding on the text field.

Next, I had to watch some tutorials to get a better understanding of how the Calendar and Gmail Intents can work in Android Studio. Then I made a small test app for adding events to Google Calendar, texting a contact, and using Gmail Intents. Upon testing, I found that all three of these functionalities worked effectively. Not to mention, it would theoretically mean that the user would only need to log in to their Gmail account once when they activate it the first time.

While I did try a 4-video playlist about making a Google Calendar clone in Android Studio, it did use a sort of stylesheet type of thing that unfortunately I couldn't find so I decided to scrap it.

Now What?

What can you do to address outstanding challenges?

For the user profile, I noticed that when I tried to run the Edit Profile function, it caused the emulator to crash. After a little digging, I found a 3-part tutorial on YouTube where the login/registration leads into the user profile page and allows the chance to edit the profile. So, this would mean that if I changed the journey from the login page to the profile page and add a button to take users to the home page in the latter, I can avoid the app crashing when I run it.

As I plan on having a function that can let users save medical details and medication details, I need to talk to my supervisor about how I can approach. Ordinarily, that could mean taking the code I have for the add contact page and modifying it for adding medical information. I could do 2 separate entry boxes though that does run the risk of making the code looking all over the place; but it's best to talk to Adriana first before I go from there.

Student Signature	[illegible]

Reflective Journal – February:

Supervision & Reflection Template

Student Name	Eoghan Feighery
Student Number	X19413886
Course	Software Development BSHCSD
Supervisor	Adriana Chis

What?

Reflect on what has happened in your project this month?

At the start of the semester, I had an accidental run-in with some malware, and I thought my application was ruined but luckily, I was able to recreate all of the pages and functions. I also managed to add an authentication object to the registration and login systems as my supervisor suggested to help make user account details more secure in Firebase.

Then, I made sure to create two extra CRUD functionalities for adding diagnoses and medication items which wasn't so hard to do since their functions are near identical to the one where users can add contacts. Finally, I was able to test out a tutorial that allowed a user profile to display retrieved data from the database table and managed to work as intended alongside the Firebase Authentication object mentioned in the above paragraph.

I recently added my project onto a GitHub repository so that it can act like a spare backup and lets me frequently update it as time goes on.

So What?

Since I technically have all the functionalities in my application completed, the next step would be to get feedback on how the overall application functions, are all the functionalities working as they intended.

I would also need to figure out how I'll carry out the testing for the application especially there could be bugs that could severely cripple the app if left unchecked.

Now What?

What can you do to address outstanding challenges?

Since I have to demonstrate my application to my Computing Project support teacher next week, I'll ask for any feedback and suggestions since Android Studio can at times feel like a tightrope act when it comes to testing out functionalities and applications. I'll also ask my supervisor about how I can tackle the testing and integration of my application since it's essential for the Technical report documentation.

Student Signature	Eoghan Feighery

Reflective Journal – March:

Supervision & Reflection Template

Student Name	Eoghan Feighery
Student Number	X19413886
Course	Software Development BSHCSD
Supervisor	Adriana Chis

What?   I was able to implement the alarm clock feature and the text message and email creator functionalities. During a Computing Project Support, I noticed that saved CRUD data entries from past user sessions were actually bleeding into each other. After some research, I concluded that Firebase RealTime wasn't the correct database option for the CRUD functionalities.
So What?   Before I can start any testing with Espresso and dummy emails, I decided to investigate FireStore which is another database that Firebase offers. After correctly implementing FireStore as a database, I found an hour-long tutorial on YouTube which mentions how saved CRUD entries in FireStore can be tracked and uploaded to specific user IDs. This involves with creating notes which functions very similar to the RealTime version of CRUD functionalities.
Now What?   I've done the code for one of my CRUD entries which is saving notes about medication instructions. I'll need to do the same for the other two (diagnoses and contacts) but the overall code is pretty similar to my aforementioned prototype. Because I have all of the other non-CRUD functionalities working, I can focus on the FireStore database first and then worry about testing and using Espresso for my application.

Student Signature	Eoghan Feighery

Reflective Journal – April:

Supervision & Reflection Template

Student Name	Eoghan Feighery
Student Number	X19413886

Course	Software Development BSHCSD
Supervisor	Adriana Chis

Month: April

What?

I managed to complete some Whitebox testing for my Android application and also recorded a few Espresso sessions. With that out of the way, all of the functionalities are functioning as they were all declared. All that's left for me to do is the documentation and the deployment for my application.

So What?

As for deployment, while I still plan to deploy my application for the Google Play Store in the near future, I realised that it actually was technically optional. Not to mention the presentation video will take up to at least by the start of next week at most if things go according to plan.

Now What?

I asked on my college WhatsApp for any other alternatives with deploying my application. One of my friends suggested using F-Droid which is identical to Google Play Store except the former allows for free and open-source applications. For the time being, I'll look into the documentation my friend shared with me which should take about the guts of an hour. If I don't deploy it, it's no loss since it wasn't a mandatory requirement.

Although, the development of MedsUpApp was far from easy, I still feel proud knowing that I was able to at least create a fully functioning Android application regardless.

Student Signature	[illegible]