

Mobile Programming & Screen Design 2 Documentation - Medicare

by Goh Calvin Kai Boon

Submission date: 11-Aug-2022 10:38PM (UTC+0800)

Submission ID: 1881386460

File name: MPSD2_Documentation_turnitin.pdf (38.33M)

Word count: 4295

Character count: 21285

1.0 Introduction

In this day and age, with the prevalence of smartphones, everything we want to search for or need to know is at our fingertips. Smartphones have also allowed us to constantly track our health with built in applications. Therefore, for this project we would like to talk about our own application which also has the function of tracking our health and the ability to make appointments with associated clinics and hospitals that have collaborated with our application. The aim of our application is to make an application that fits an all-in-one system where people can do everything using it. For example, the ability to check their health, the ability to make appointments immediately and have consultations with doctors through the app itself for a diagnosis. With this new application, people can participate in their own health care seamlessly and effortlessly with everything located in just one place.

According to Pew Research Centre 97% of Americans owned a cell phone (2021) and that those that own a smartphone is 85% which is a huge leap compared to Pew Research Centre's first survey in 2011 where only 35% of Americans owned a smartphone. This is evident in surveys of healthcare professionals where many admit they own a smartphone which is used in both clinical practice and education (Aungst, 2013). One of the most important reasons why healthcare professionals use smartphones is because it allows for better communication and information resources (Mosa, Yoo and Sheets, 2012). On the average person's side however, it would lean more towards the fact that there have been rapid advancements in technology, which allows applications to have many more complex functions to accomplish a specific purpose thus leading to a rise in mobile device applications such as medical applications. Users now have a variety of applications to choose from catering to any of their needs (Mosa, Yoo and Sheets, 2012). Apart from that, smartphone compatible devices such blood pressure cuffs, and pulse oximeters can be found easily in a normal person's house (Batista and Gaglani, 2013). These smartphone compatible devices typically have a friendly user interface that are intuitive and provide health data which allows users to take their own healthcare into their own hands. They can also summarise the data into graphs that display the data in a concise form. Unfortunately, even with all these advances, they all still lack something, which is the ability to schedule/cancel an appointment, the ability to request prescriptions and easy access to medical records and only 11% of hospital apps offer one of those functionalities (georgiou, 2022).

1.1 Hypothesis & Objectives

8

Our hypothesis is will more people use our app if we were to offer functionalities like ability to schedule/cancel appointments, the ability to request prescriptions and easy access to medical records. Therefore, we aim to implement these functions into our application in a friendly user interface so that the user will have an easier time using our application. Delving into the objectives of developing the application and its significance in the current day and age is to provide convenient access to medical services, the main focus being the ability for patients to book an appointment online with any type of medical professional to perform virtual consultations, with the assistance of modern day mobile technology at a push of a button; to add, it also aims to increase the exposure of people to essential medical knowledge and awareness.

2.0 Methodology

12

2.1 Tools Used

2.1.1 Android Studio



Figure 1.1

6

Android Studio is the official integrated development environment (IDE) for Android application development. Android Studio uses a Gradle-based build system, emulator, has code templates as well as a GitHub integration. It incorporates Java to edit codes

2.1.2 Java

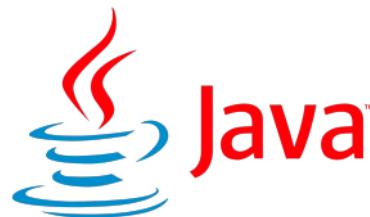


Figure 1.2

9

Java is an object-oriented programming language released by Sun Microsystems in 1995. Now, 3 billion devices run Java (w3school, n.d.). Java is widely used because Java has provided constant testing, updating and consistency for as long as it has been around. Not only that, Java has also constantly been maintained and tested by a community of Java developers

2.1.3 Firebase



Figure 1.3

Firebase is Google's mobile application development that is used to help in the development of an application. What Firebase does is that it helps developers out by taking care of many services that developers would normally have to build themselves. For example, authentication services, databases, and file storages. All these services are hosted online

2.1.4 GitHub



Figure 1.4

GitHub is a website that allows developers to store and manage their code online. GitHub also allows developers to track and control changes to their code. GitHub comprises two connected principles, version control and git. Version control is the part that helps developers work through branching and merging. Branching is when the code is duplicated where changes to code can be made without affecting the main code. Merging is when the developer submits the code back into the main code. The next principle git is the distributed version control system where the entire code is available on every developer's computer.

2.1.5 Figma



Figure 1.5

Figma is a UI and UX design application to create websites or applications. Figma is also a cloud-based design application tool that can be accessed by separate users to work on projects together whenever and wherever. Figma also has many features and services included such as Vector Networks. These functionalities give users the chance to do whatever they need to do.

2.2 Techniques and Methodologies Used

2.2.1 Waterfall Methodology

The methodology that we implemented was the Waterfall Methodology. The Waterfall methodology has a chronological process and works based on fixed dates, requirements and outcomes. In this methodology, each phase must be completed fully before being able to move onto the next phase.

7

The Waterfall Method

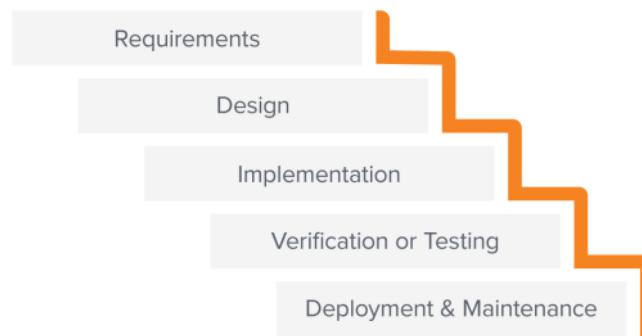


Figure 2.1 Waterfall Method

As seen in the picture above, before we started anything we had to complete the requirements phase. Here, we established whether all the project requirements could be gathered upfront where we did our best to understand what the assignment required.

After completing the requirements phase, we moved onto the design phase whereby we used a collaborative interface design tool known as Figma to come up with our mobile app designs. We spent a few days coming up with the design in order to come up with something because we had to come up with something that would fulfil the scope of the project. We also had to create scenarios and layouts of the project and how each component of the app would flow. After we all mutually agreed on the design, we moved onto the implementation phase.

In the implementation phase, we decided to split the project into smaller sections for each of us to complete as well as a due date for each of us to submit our work. Here, we coded the application based on the requirements and specifications. When we were done, we had to verify that the application was working meaning we had to conduct several instances of testing to ensure it

worked. If there were any changes, we would have had to go back to the design phase to do it properly.

Lastly, for the deployment and maintenance phase is when we would submit this application to the lecturer as it is already completed.

2.2.2 Data Binding Library

Data binding is the act of integrating data objects with views in an XML layout. The Data Binding Libraries responsibility is to generate the necessary classes for this method (Pandya, 2021).

Data Binding-layout XML files, unlike other types of layout XML files, begin with a root layout tag followed by a data element. Then, each layout file is coupled with a Data Binding class created by the library (Android Developers, 2022b).

2.2.3 Fragment Navigation

Android 3.0 (API level 11) added fragments to facilitate more dynamic and adaptable UI designs on big displays, such as tablets. Because a tablet's screen is far more extensive than a smartphone's, there is more area to mix and swap UI elements. Fragments enable such designs without requiring the management of extensive modifications to the view hierarchy. By breaking an activity's layout into pieces, we may edit the activity's appearance at runtime and save those changes in a back stack maintained by the activity (Android Developers, 2018).

2.2.4 Hashmap

It is a data structure used to organise data such that instead of mapping index to values as in an array, keys are used to pinpoint each row of data in memory, supported with the use of a hash function to connect the key to the location of the memory; therefore, this enables for random insertion of data into the data structure as opposed to an array which requires insertion according to positions within the array indicated by its index (Android Developers, 2022).

2.3 Prototype Design

The Figures below shows the high-fidelity prototype we designed before carrying out the development process.

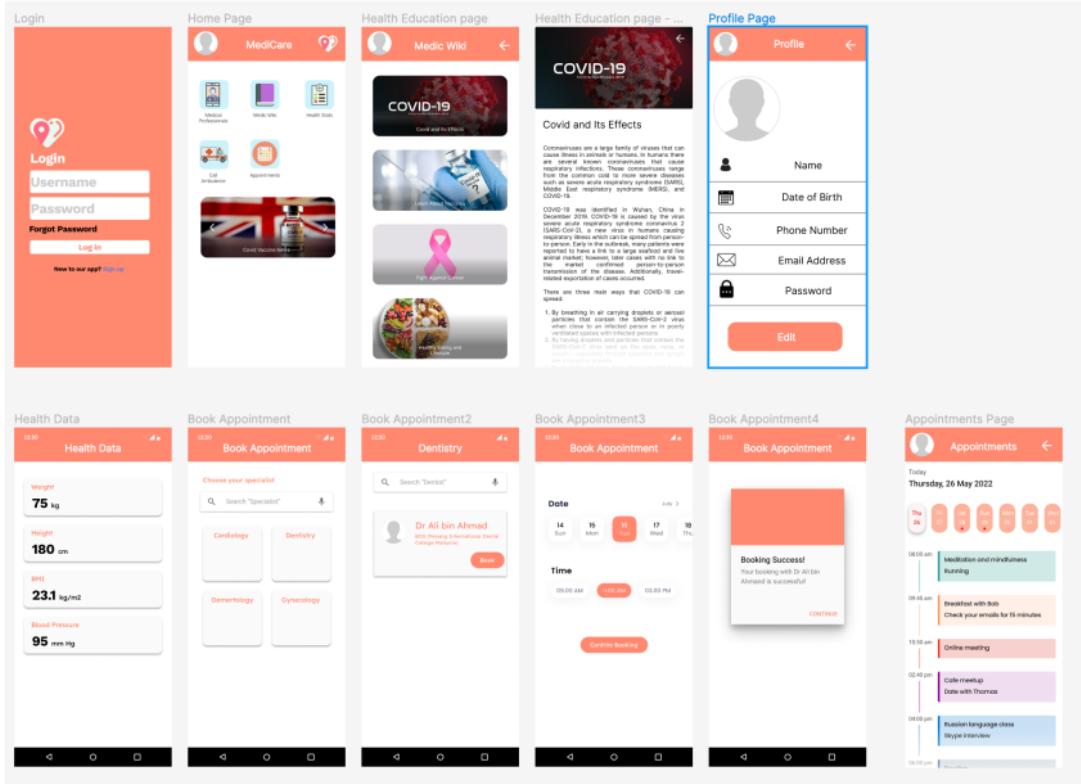


Figure 3.1

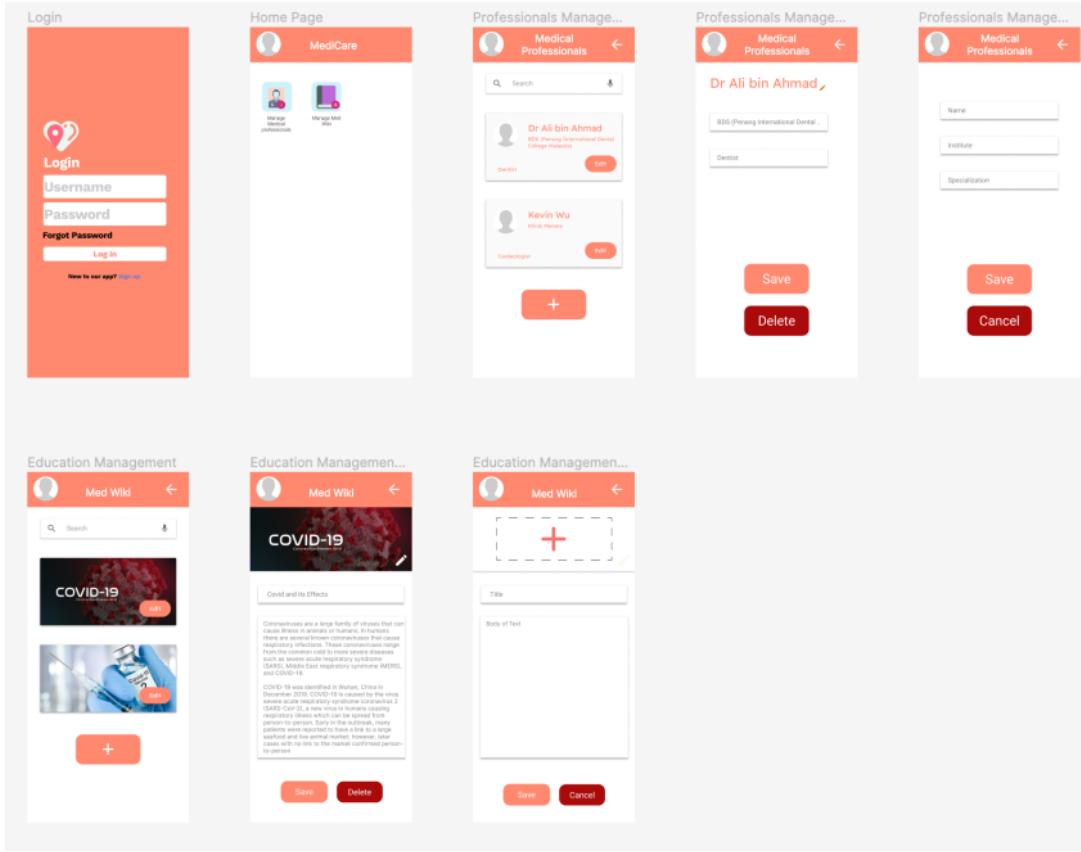


Figure 3.2

3.0 Results

The purpose of our study of the problem is to determine whether more people would use our app if we were to offer functionalities like ability to schedule/cancel appointments, the ability to request prescriptions and easy access to medical records. Thus, we have compiled the functionalities of our application with a breakdown on what each page does

3.1 Functionalities

3.1.1 Admin

3.1.1.1 Home

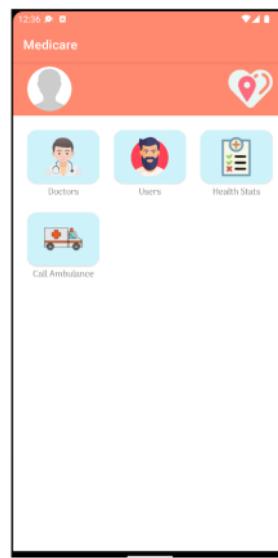


Figure 4.1

As the admin, when they access the main menu, they will see 4 options which are doctors, users, health stats and call an ambulance. Administrators will be able to select whichever button to conduct its corresponding activity

3.1.1.2 Add Doctors

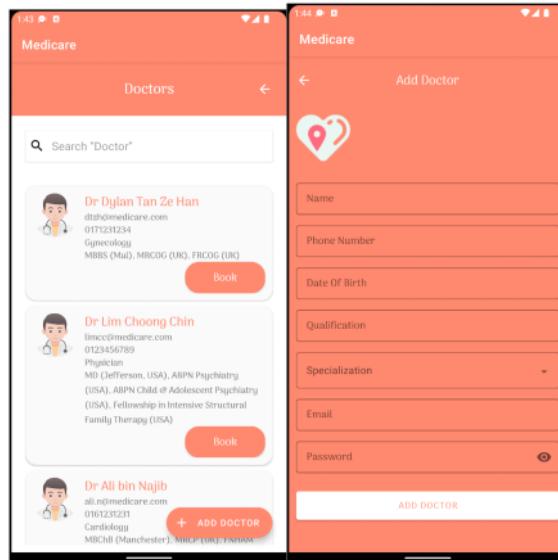


Figure 4.2

Administrators have the ability to add and register new doctors to the database. They can view all doctors that are registered into the system. In the other screen, they add doctors to the database and must include important things such as name, qualifications and specialisations.

3.1.1.3 Manage Users

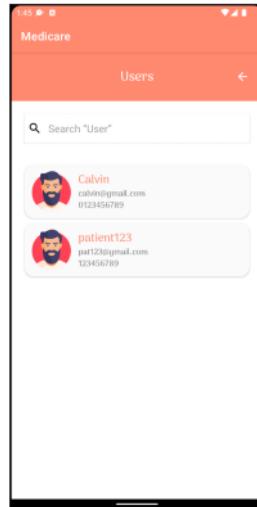


Figure 4.3

Next, administrators have the ability to search for any particular user which also displays all the user details

3.1.2 Users

3.1.2.1 Home



Figure 4.4

As the user, when we access the homepage, we have access to 5 options, which are medical professionals, medic wiki, health stats, call ambulance and appointments. Users will be able to click on any one of the options to conduct its corresponding activity

3.1.2.2 Medical Professionals

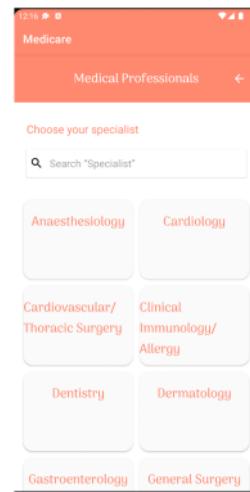


Figure 4.5

In the medical professionals' option, users will be able to search for any specialist they wish to see. In the image above, users can also scroll through the numbers of specialisations if they are unsure onto what they are looking for

3.1.2.3 Health Data

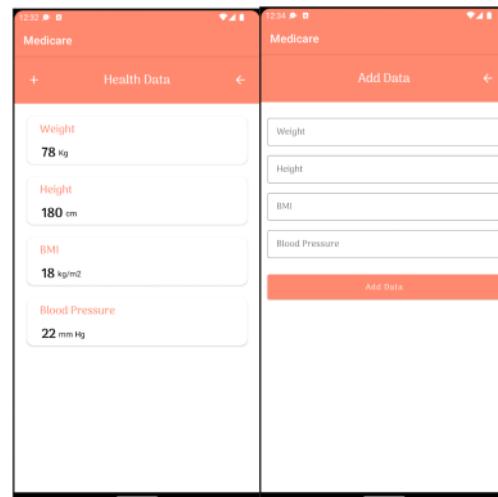


Figure 4.6

11

In the health data tab, users will be able to enter their weight, height, body mass index (BMI) and blood pressure. They can constantly track their progression this way

3.1.2.4 Book Appointment

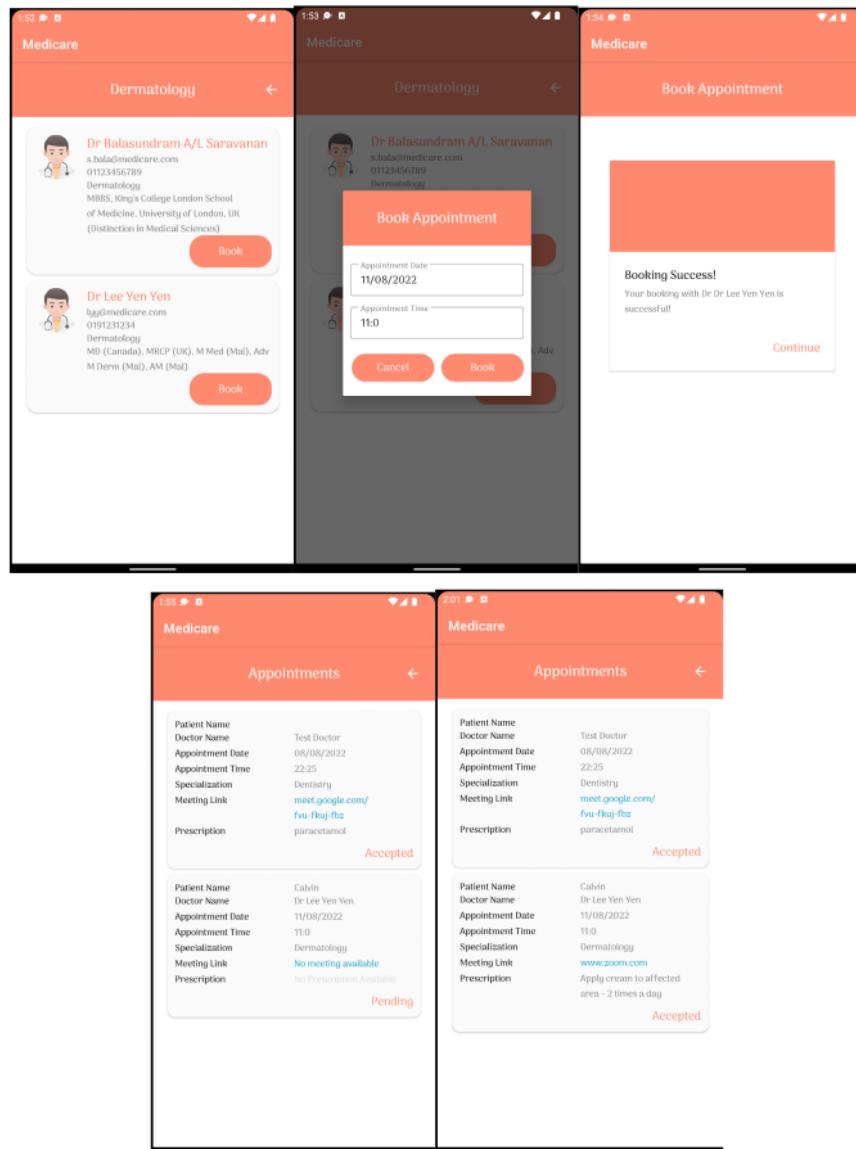


Figure 4.6

Users will be able to book appointments using our application with our built-in appointment registration system. First, from the medical professionals' page, they will select a specialisation. For example, dermatology. After picking dermatology, they will see a list of doctors who specialise in that particular field. Users will pick a doctor they wish to see and will then pick a time and date for their appointment. Patients will be informed that their appointment was registered successfully. Later, they can check on all their appointments which will specify important things such as appointment date and time as well as a link to an online meeting for their appointment. After the appointment, their prescription will be updated to the given prescription by the doctor along with the number of times they need to take it

3.1.3 Doctors

3.1.3.1 Home

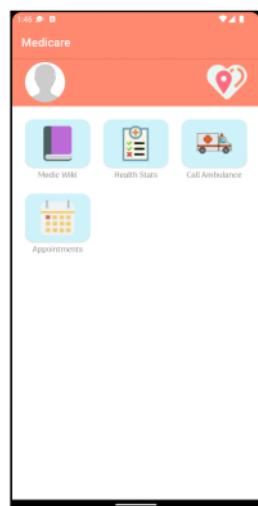


Figure 4.7

For the doctors, when they enter the main menu, they will also have 4 options which are medic wiki, health stats, call ambulance and appointments.

3.1.3.2 Appointment

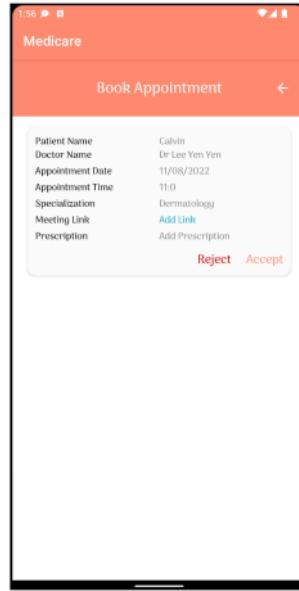


Figure 4.8

When the doctor clicks on the appointments option, it will bring him to this page where they can view any appointment that a user has made with them. The doctor will be able to reject or accept the appointment if it clashes with another appointment. The appointment card will also list details such as patient name

3.1.3.3 Create Virtual Consultation

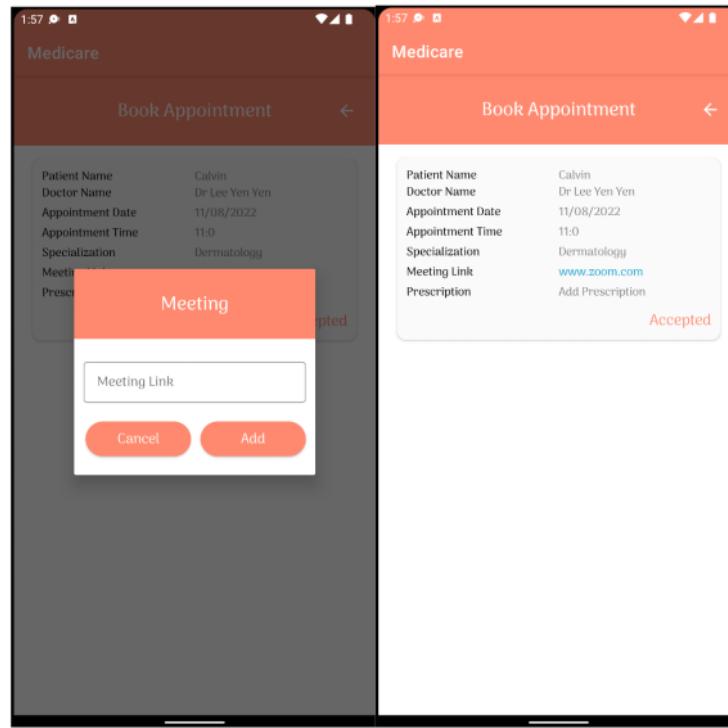


Figure 4.9

If the doctor does accept the appointment booking, a popup will be displayed in front of them where they have to add a meeting link for the patient to use. After adding the meeting link, the appointment card will change to an accepted status to show the doctor their accepted appointments

3.1.3.4 Prescriptions

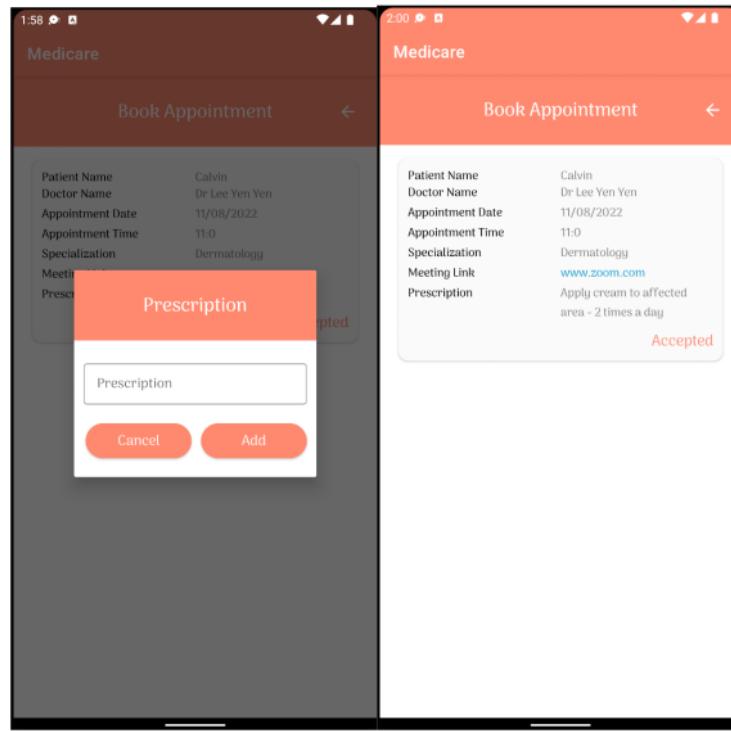


Figure 4.10

After the appointment and diagnosing the patient, the doctor will give a prescription to the page. Once again, a popup will be displayed in front of the doctor where he will enter the prescription. After adding it, the appointment card will be update with the prescription so that the patient will be reminded of the prescription whenever they check the application

3.1.4 Common Functions

3.1.4.1 Login

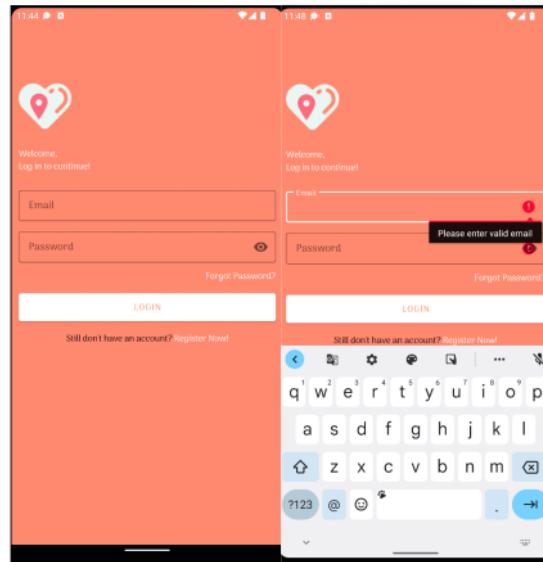


Figure 4.11

This is the login page for the application where users have to input email and password. If the user fails to enter an email or password, they will be alerted to enter an email. The login screen also allows users to view their password in case they mistyped something

3.1.4.2 Profile

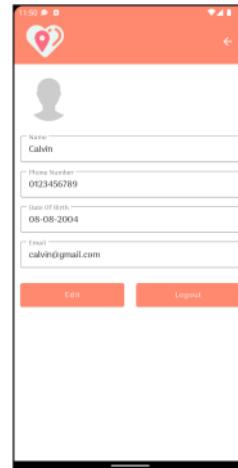


Figure 4.12

The profile page for the user. Users will enter their name, phone number, date of birth and their email which will all be used when making an appointment with a doctor

3.1.4.3 MedicWiki

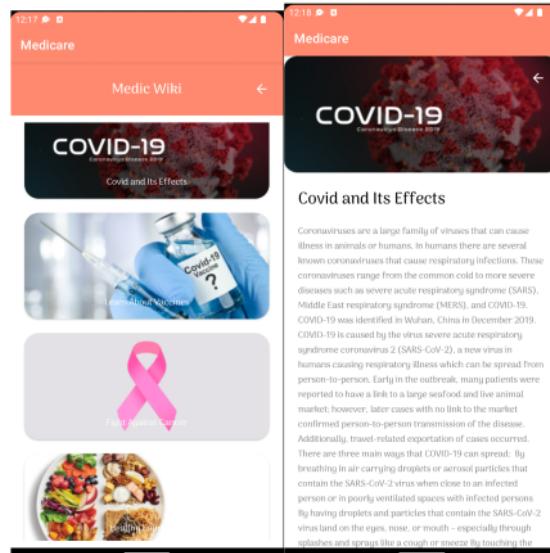


Figure 4.13

Users can also check on the medical wiki page where it displays things like the current Covid-19 situation along with its effects, vaccinations, or healthy eating. Users can look for self-improvement options here

3.1.4.4 Emergency Call



Figure 4.14

Users can register a number for an emergency call as well as call emergency services

3.1.4.5 Loading Screen

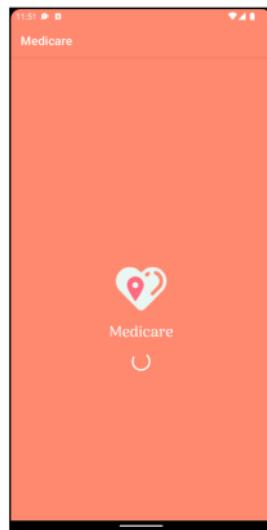


Figure 4.15

A simple loading screen to indicate when a page is loading.

Summary

Our applications main features are its easy-to-use functionalities on the users, doctors and administrators sides where everything is located in one place. The application also features a very clean design that makes navigating through easy and convenient to use. Along with its ability to track the user's health, users will be inclined to use the features provided by our application

4.0 Implementation

With regards to development, we used Android Studio for this project. We created a GitHub project and then imported the project to our Android Studio so that we could share any changes among us seamlessly and easily. This way, any changes made would also be documented.

4.1 Design

4.1.1 Edit Text View

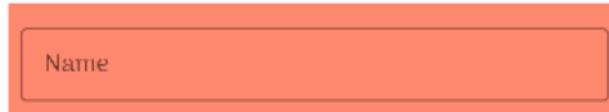


Figure 5.1

The Edit Text View is used wherever an editable text field is required such as in the case of the Registration or Login pages.

4.1.2 Text View

Still don't have an account? [Register Now!](#)

A screenshot of an Android application interface. It shows a single button with a solid red background and white text. The text on the button reads "Still don't have an account? [Register Now!](#)". The button has a slightly rounded rectangular shape and a thin white border.

Figure 5.2

The normal Text View is used to display a static text such as the one shown in the figure above, this can range from a short text to label an icon or even a whole paragraph of text to convey a message to the user. This component is used throughout the application

4.1.3 RecyclerView

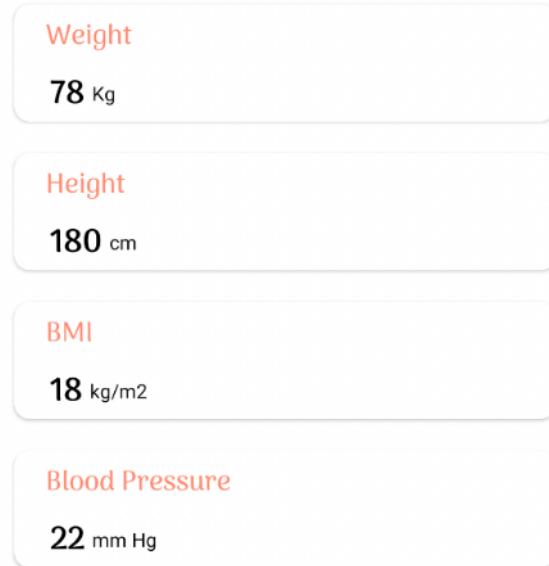


Figure 5.3

The RecyclerView is basically an extension of 2 existing views available in the android native development environment, ListView and GridView, however it provides the ability to react dynamically to changes in the list. Hence when an element is added or moved in the list it will reflect in the RecyclerView; such a component is used in the Health Data page.

4.1.4 Card View

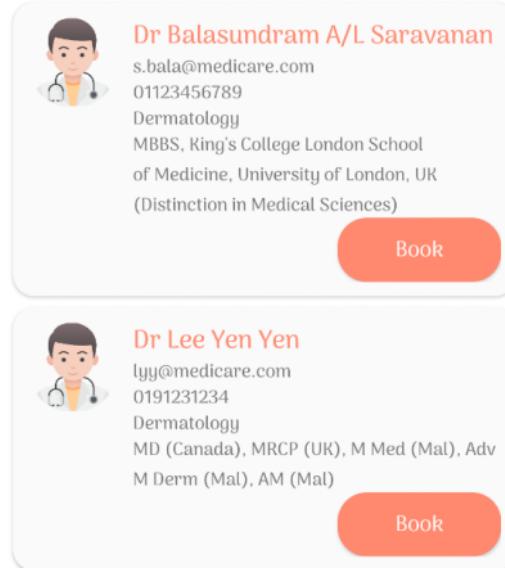


Figure 5.4

The CardView allows the developer to organise information for respective medical professionals into cards for easier browsing and searching by the patient user; such a component is mostly used in the medical professionals' page.

4.1.5 Image View

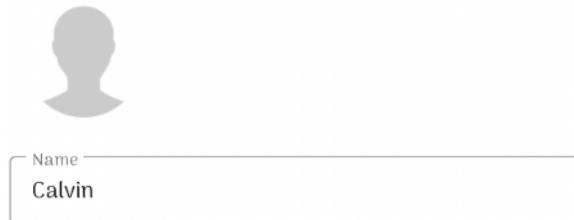


Figure 5.5

The ImageView is used to display images in any area of the pages by referencing the image source to a png or jpg; and this component can be seen almost throughout the app such as in the Edu Wiki Page and Medical Professionals page.

4.1.6 Button



Figure 5.6

The button allows the user of the app to invoke a function from the app such as redirecting to another page or submitting a request, i.e. when booking an appointment in the Medical professionals page.

4.1.7 Floating Button



Figure 5.7

Serves the same purpose as the button component but is given the attribute of floating over other elements within the page such as can be seen in the admin's side Doctors page.

4.1.8 Dialog

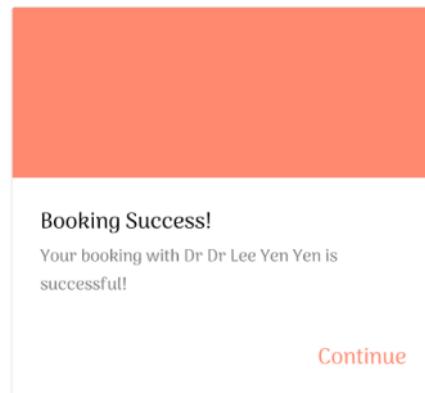


Figure 5.8

The dialog box is basically a component to prompt the user with some message as can be seen in the page after the patient user has clicked on any of the “Book” buttons.

4.1.9 Progress Indicator

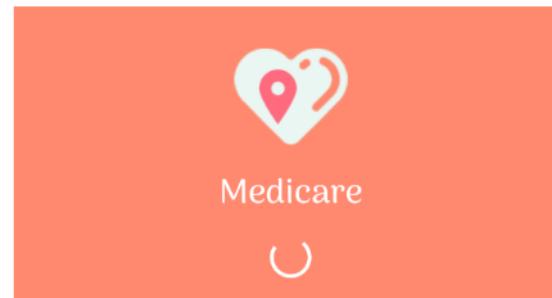


Figure 5.9

The indicator is to indicate to the user that the app is still loading so that the user knows whether the app is still running as intended while trying to load a page.

4.1.10 Calendar

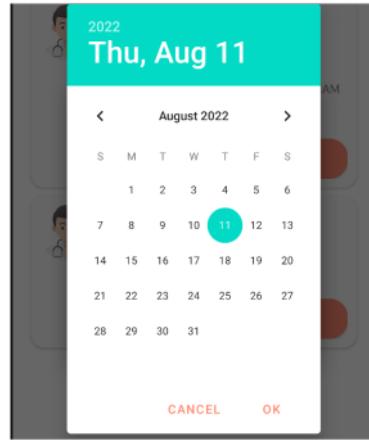


Figure 5.10

The calendar component basically facilitates the action of picking a date so that the user may have a more intuitive and efficient user experience; this component is used in the Booking page when the patient user is about to book an appointment with any medical professional.

4.2 Coding/Development

4.2.1 Register User Account

RegisterFragment.java

```
26  public class RegisterFragment extends Fragment {
27
28      private FragmentRegisterBinding binding;
29      private DatabaseReference userRef;
30      private FirebaseAuth mAuth;
31      private String profileType = "user";
32      private StringBuffer strBuf;
33      private String dobDate, formatDate;
34
35
36      @Override
37      public View onCreateView(LayoutInflater inflater, ViewGroup container,
38                             Bundle savedInstanceState) {
39          // Inflate the layout for this fragment
40          binding = FragmentRegisterBinding.inflate(inflater, container, false);
41
42          //initialize
43          userRef = FirebaseDatabase.getInstance().getReference("users");
44          mAuth = FirebaseAuth.getInstance();
45
46          //click listeners
47          binding.backArrow.setOnClickListener(view -> Navigation.findNavController(view).navigateUp());
48
49          binding.dobEt.setOnClickListener(v -> showDatePickerDialog());
50
51
52          binding.registerBtn.setOnClickListener(view -> {
53
54              //check validations
55              if (Objects.requireNonNull(binding.name.getText()).toString().isEmpty()) {
56                  binding.name.setError("Please enter user name");
57                  return;
58              }
59          });
59 }
```

```

63     if (Objects.requireNonNull(binding.emailEt.getText()).toString().isEmpty()) {
64         binding.emailEt.setError("Please enter valid email");
65         return;
66     }
67     if (Objects.requireNonNull(binding.passwordEt.getText()).toString().isEmpty()) {
68         binding.passwordEt.setError("Please enter password");
69         return;
70     }
71
72     if (Objects.requireNonNull(binding.dobEt.getText()).toString().isEmpty()) {
73         binding.dobEt.setError("Please select date of birth");
74         return;
75     }
76     //register user
77     register();
78 };
79
80     return binding.getRoot();
81 }
82
83 private void showDatePickerDialog() {
84     // Get open DatePickerDialog button.
85     // Create a new OnDateSetListener instance. This listener will be invoked when user click ok button in DatePickerDialog.
86     DatePickerDialog.OnDateSetListener onDateSetListener = (datePicker, year, month, dayOfMonth) -> {
87         strBuf = new StringBuffer();
88         // strBuf.append("You select date is ");
89
90
91         if ((month + 1) < 10) {
92             strBuf.append("0" + (month + 1));
93         } else {
94             strBuf.append(month + 1);
95         }
96
97         DatePickerDialog datePickerDialog = new DatePickerDialog(requireContext(), onDateSetListener, year, month, day);
98         now.add(Calendar.YEAR, amount: -18);
99
100        datePickerDialog.getDatePicker().setMaxDate(now.getTimeInMillis());
101        datePickerDialog.show();
102    }
103
104
105     //to get date of birth
106     @
107     private String getAge(int year, int month, int day) {
108
109         Calendar dob = Calendar.getInstance();
110         Calendar today = Calendar.getInstance();
111         int todaymonth = today.get(Calendar.MONTH);
112         int todayyear = today.get(Calendar.YEAR);
113         int today_date = today.get(Calendar.DAY_OF_MONTH);
114         today.set(todayyear, month: todaymonth + 1, date: today_date + 1);
115         dob.set(year, month, day);
116         int age = today.get(Calendar.YEAR) - dob.get(Calendar.YEAR);
117
118         System.out.println("DateFormat1 " + today.get(Calendar.YEAR) + "----" + dob.get(Calendar.YEAR));
119         System.out.println("DateFormat2 " + today.get(Calendar.DAY_OF_YEAR) + "----" + dob.get(Calendar.DAY_OF_YEAR));
120
121         if (today.get(Calendar.DAY_OF_YEAR) < dob.get(Calendar.DAY_OF_YEAR)) {
122             age--;
123         }
124
125         Integer ageInt = new Integer(age);
126         String ageS = ageInt.toString();
127         return ageS;
128     }
129

```

```
96         strBuf.append("-");
97
98     if (dayOfMonth < 10) {
99         strBuf.append("0" + dayOfMonth);
100    } else {
101        strBuf.append(dayOfMonth);
102    }
103    strBuf.append("-");
104
105    strBuf.append(year);
106
107    String ages = getAge(year, month, month + 1, dayOfMonth);
108
109    dobDate = strBuf.toString();
110
111    try {
112        DateFormat inputFormat = new SimpleDateFormat( pattern: "MM-dd-yyyy" );
113        DateFormat outputFormat = new SimpleDateFormat( pattern: "MMM-dd-yyyy" );
114        String inputDateStr = strBuf.toString();
115        Date date = inputFormat.parse(inputDateStr);
116        formateDate = outputFormat.format(date);
117    } catch (ParseException e) {
118        e.printStackTrace();
119    }
120
121
122    binding.dobEt.setText(ages + " " + "Years" + " / " + formateDate);
123};
124 // Get current year, month and day.
125 Calendar now = Calendar.getInstance();
126 int year = now.get(Calendar.YEAR);
127 int month = now.get(Calendar.MONTH);
128 int day = now.get(Calendar.DAY_OF_MONTH);
```

```
161     private void register() {
162         binding.loading.setVisibility(View.VISIBLE);
163         binding.registerBtn.setEnabled(false);
164         mAuth.createUserWithEmailAndPassword(Objects.requireNonNull(binding.emailEt.getText()).toString(),
165             Objects.requireNonNull(binding.passwordEt.getText()).toString())
166             .addOnCompleteListener(task -> {
167                 if (task.isSuccessful()) {
168                     //store user detail in firebase
169                     Users users;
170                     users = new Users(
171                         mAuth.getUid(),
172                         Objects.requireNonNull(binding.name.getText()).toString(),
173                         Objects.requireNonNull(binding.phoneNoEt.getText()).toString(),
174                         dobDate,
175                         profileType,
176                         binding.emailEt.getText().toString(),
177                         image: "",
178                         specialization: "",
179                         qualification: ""
180                     );
181                     userRef.child(Objects.requireNonNull(mAuth.getUid())).setValue(users).addOnCompleteListener(task1 -> {
182                         if (task1.isSuccessful()) {
183                             binding.loading.setVisibility(View.GONE);
184                             Snackbar snackbar = Snackbar
185                                 .make(binding.getRoot(), text: " Register Successfully", Snackbar.LENGTH_LONG);
186                             snackbar.show();
187                             mAuth.signOut();
188                             Navigation.findNavController(requireView()).navigateUp();
189                         }
190                     });
191                 }
192             }).addOnFailureListener(e -> {
193                 binding.loading.setVisibility(View.GONE);
194                 binding.registerBtn.setEnabled(true);
195                 Snackbar snackbar = Snackbar
196                     .make(binding.getRoot(), Objects.requireNonNull(e.getMessage()), Snackbar.LENGTH_LONG);
197                 snackbar.show();
198             });
199         });
200     }
```

Figure 6.1

The figure above shows the codes for account registration using `createUserWithEmailAndPassword` method provided by Firebase Authentication. As stated in the previous chapter, View Binding is used for this project. For each XML layout file existing in the module, it creates a binding class. All views with an ID in the relevant layout are directly referenced in an instance of a binding class. For example, in line 40, call the static `inflate()` method included in the generated binding class. This creates an instance of the binding class for the activity to use instead of using `findViewById`. The edit text (Et) basically would accept email as the username and the respective password. When registration is successful, the snackbar will display the 'Successful' toast message.

4.2.2 Login

LoginFragment.java

```
38     public class LoginFragment extends Fragment {
39         private FragmentLoginBinding binding;
40         private FirebaseAuth mAuth;
41         private DatabaseReference userRef;
42
43         @Nullable
44         @Override
45         public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {
46             // Inflate the layout for this fragment
47             binding = FragmentLoginBinding.inflate(inflater, container, attachToParent: false);
48
49             //initialize
50             mAuth = FirebaseAuth.getInstance();
51
52             userRef = FirebaseDatabase.getInstance().getReference(path: "users");
53
54             //listeners
55             binding.forgotPasswordTv.setOnClickListener(view -> Navigation.findNavController(view).navigate(R.id.action_LoginFragment_to_forgotPasswordFragment));
56             binding.registerTv.setOnClickListener(view -> Navigation.findNavController(view).navigate(R.id.action_LoginFragment_to_registerFragment));
57
58             //permissions
59             if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
60                 try {
61                     requestPermissions(new String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE,
62                         Manifest.permission.READ_EXTERNAL_STORAGE, Manifest.permission.CAMERA}, requestCode: 555);
63                 } catch (Exception e) {
64                     Log.e(tag: "userPer", e.toString());
65                 }
66             }
67
68         }
69
70         binding.loginBtn.setOnClickListener(view -> {
71             //checking validations
72             if (Objects.requireNonNull(binding.emailEt.getText()).toString().isEmpty()) {
73                 binding.emailEt.setError("Please enter valid email");
74                 return;
75             }
76
77             if (Objects.requireNonNull(binding.passwordEt.getText()).toString().isEmpty()) {
78                 binding.passwordEt.setError("Please enter password");
79                 return;
80             }
81
82             //login
83             login();
84         });
85
86
87
88         return binding.getRoot();
89     }
90
91
92     @Override
93     public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {
94         super.onRequestPermissionsResult(requestCode, permissions, grantResults);
95     }
96 }
```

The screenshot shows the Java code for a login function within an Android application. The code uses Firebase Authentication's `mAuth.signInWithEmailAndPassword` method to sign in users. If the sign-in is successful, it retrieves user data from the Firebase Realtime Database and stores it in SharedPreferences. It then checks the user's type ('admin', 'doctor', or 'user') and directs them to their respective home activity. If the sign-in fails, a toast message is displayed indicating the error.

```
98     private void login() {
99         binding.loading.setVisibility(View.VISIBLE);
100        binding.loginBtn.setEnabled(false);
101
102        mAuth.signInWithEmailAndPassword(binding.emailEt.getText().toString(), binding.passwordEt.getText().toString())
103            .addOnCompleteListener(task -> {
104                if (task.isSuccessful()) {
105                    userRef.child("Objects").requireNonNull(mAuth.getUid())
106                        .addListenerForSingleValueEvent(new ValueEventListener() {
107                            @Override
108                            public void onDataChange(@NonNull DataSnapshot snapshot) {
109                                if (snapshot.exists()) {
110
111                                    Snackbar snackbar = Snackbar
112                                        .make(binding.getRoot(), "Login Successfully", Snackbar.LENGTH_LONG);
113                                    snackbar.show();
114                                    binding.loading.setVisibility(View.GONE);
115                                    Users users = snapshot.getValue(Users.class);
116
117                                    // Storing data into Shared Preferences
118                                    SharedPreferences sharedPreferences = requireActivity().getSharedPreferences(Constants.sharedPrefName, MODE_PRIVATE);
119                                    SharedPreferences.Editor editor = sharedPreferences.edit();
120                                    editor.putString(Constants.userType, users.getType());
121                                    editor.apply();
122
123                                    if (users.getType().equals("admin")) {
124
125                                        SharedPreferences.Editor editor1 = sharedPreferences.edit();
126                                        editor1.putString(Constants.adminEmail, binding.emailEt.getText().toString());
127                                        editor1.putString(Constants.adminPassword, binding.passwordEt.getText().toString());
128                                        editor1.apply();
129
130                                        Intent intent = new Intent(getApplicationContext(), AdminHomeActivity.class);
131                                        startActivity(intent);
132                                        requireActivity().finish();
133                                    } else if (users.getType().equals("doctor")) {
134                                        Intent intent = new Intent(getApplicationContext(), DoctorHomeActivity.class);
135                                        startActivity(intent);
136                                        requireActivity().finish();
137                                    } else if (users.getType().equals("user")) {
138                                        Intent intent = new Intent(getApplicationContext(), UserHomeActivity.class);
139                                        startActivity(intent);
140                                        requireActivity().finish();
141                                    }
142                                }
143                            }
144                            @Override
145                            public void onCancelled(@NonNull DatabaseError error) {
146
147                            }
148                        });
149                    }).addOnFailureListener(e -> {
150                        binding.loading.setVisibility(View.GONE);
151                        binding.loginBtn.setEnabled(true);
152                        Snackbar snackbar = Snackbar
153                            .make(binding.getRoot(), e.getMessage(), Snackbar.LENGTH_LONG);
154                        snackbar.show();
155                    });
156                }
157            }
158        }
    }
```

Figure 6.2

The figure above shows the codes for Login Function. The username and password requires validation from the firebase in order to login users into their respective account. Hence, we implement `mAuth.signInWithEmailAndPassword` linked to the firebase where if the validation is successful, the firebase would fetch users' accounts according to their username and password. Users will be directed to the home page after login. However, if the validation process fails, a toast message will pop up to inform the users that their username or password is invalid. From line 123, it will also check on the user's type before directing the respective users to their respective home page. Besides, `addOnFailureListener()` is to ensure users' meet the requirements of inputs as the validation would fail and an error message will be displayed if either textbox is left blank.

4.2.3 Home

AdminHomeActivity.java

UserHomeActivity.java

DoctorHomeActivity.java

```
33     public class AdminHomeActivity extends AppCompatActivity implements HomeAdapter.OnItemClickListener {
34         private ActivityAdminHomeBinding binding;
35         private HomeAdapter mAdapter;
36         private ArrayList<Home> homeArrayList = new ArrayList<>();
37         private FirebaseAuth mAuth;
38         private DatabaseReference userRef;
39         @Override
40         protected void onCreate(Bundle savedInstanceState) {
41             super.onCreate(savedInstanceState);
42             binding = ActivityAdminHomeBinding.inflate(getLayoutInflater());
43             setContentView(binding.getRoot());
44
45             //initialize
46             mAuth = FirebaseAuth.getInstance();
47             userRef = FirebaseDatabase.getInstance().getReference("users");
48             binding.userImage.setOnClickListener(v -> startActivity(new Intent(getApplicationContext(), AdminHomeActivity.this, ProfileActivity.class)));
49
50             setHomeMenu();
51         }
52         @Override
53         protected void onStart() {
54             super.onStart();
55             getUserData();
56         }
57     }
58
59     private void getUserData() {
60         userRef.child("Objects").requireNonNull(mAuth.getUid())
61             .addValueEventListener(new ValueEventListener() {
62                 @Override
63                 public void onDataChange(@NonNull DataSnapshot snapshot) {
64                     if (snapshot.exists()) {
65
66                         Users users = snapshot.getValue(Users.class);
67
68                         assert users != null;
69                         if (!users.getImage().equals("")) {
70                             Glide.with(activity: AdminHomeActivity.this)
71                                 .load(users.getImage())
72                                 .into(binding.userImage);
73                         }
74                     }
75                 }
76             });
77     }
78
79     @Override
80     public void onCancelled(@NonNull DatabaseError error) {
81         Snackbar snackbars = Snackbar
82             .make(binding.getRoot(), error.getMessage(), Snackbar.LENGTH_LONG);
83         snackbars.show();
84     }
85 }
86
87 }
```

```
88     private void setHomeMenu() {
89         homeArrayList.add(new Home( id: "1", title: "Doctors", R.drawable.doctor));
90         homeArrayList.add(new Home( id: "2", title: "Users", R.drawable.user));
91         homeArrayList.add(new Home( id: "3", title: "Health Stats", R.drawable.stats));
92         homeArrayList.add(new Home( id: "4", title: "Call Ambulance", R.drawable.ambul));
93         mAdapter = new HomeAdapter( context: this, homeArrayList);
94         binding.homeMenuRv.setLayoutManager(new GridLayoutManager( context: this, spanCount: 3));
95         binding.homeMenuRv.setAdapter(mAdapter);
96         mAdapter.setOnItemClickListener(this);
97     }
98
99     @Override
100    public void onListItemClick(Home home) {
101        if (home.getId().equals("1")){
102            startActivity(new Intent( packageName: AdminHomeActivity.this, DoctorsActivity.class));
103        }if (home.getId().equals("2")){
104            startActivity(new Intent( packageName: AdminHomeActivity.this, UsersActivity.class));
105        }if (home.getId().equals("3")){
106            startActivity(new Intent( packageName: AdminHomeActivity.this, HealthStatsActivity.class));
107        }if (home.getId().equals("4")){
108            // Use format with "tel:" and phoneNumber created is
109            // stored in u.
110            Uri u = Uri.parse("tel://" + "999");
111
112            // Create the intent and set the data for the
113            // intent as the phone number.
114            Intent i = new Intent(Intent.ACTION_DIAL, u);
115            try {
116                // Launch the Phone app's dialer with a phone
117                // number to dial a call.
118                startActivity(i);
119            } catch (SecurityException s) {
120                // show() method display the toast with
121                // exception message.
122                Toast.makeText( context: this, text: "An error occurred", Toast.LENGTH_LONG)
123                    .show();
124            }
125        }
126    }
127
128 }
129
130 }
```

Figure 6.3

The figure above shows the codes for the admin home page but for users and doctors home page, the codes will be similar to the admin. In the home page, there will be menus for users to navigate to another page. These menu items are stored into `homeArrayList` and set into respective ids. The menu items are then added to the `onListItemClick` and assign intents to their respective layout.

HomeAdapter.java

```
16  public class HomeAdapter extends RecyclerView.Adapter<HomeAdapter.ViewHolder> {
17
18      private Context context;
19      ArrayList<Home> homeArrayList;
20      OnItemClickListener mListener;
21
22      public HomeAdapter(Context context, ArrayList<Home> homeArrayList) {
23
24          this.context = context;
25          this.homeArrayList = homeArrayList;
26      }
27
28      public interface OnItemClickListener {
29          void onListItemClick(Home home);
30      }
31
32      public void setOnItemClickListener(OnItemClickListener listener) { mListener = listener; }
33
34      @NotNull
35      @Override
36      public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
37          CustomUserHomeLayoutBinding binding = CustomUserHomeLayoutBinding
38              .inflate(
39                  LayoutInflater.from(parent.getContext()),
40                  parent,
41                  attachToParent: false);
42
43
44          return new ViewHolder(binding);
45      }
46
47      @Override
48      public void onBindViewHolder(@NotNull final ViewHolder holder, int position) {
49          Home home = homeArrayList.get(position);
50          holder.bind(home);
51      }
52
53      @Override
54      public int getItemCount() { return homeArrayList.size(); }
55
56      public class ViewHolder extends RecyclerView.ViewHolder {
57          private final CustomUserHomeLayoutBinding binding;
58
59          public ViewHolder(CustomUserHomeLayoutBinding binding) {
60              super(binding.getRoot());
61              this.binding = binding;
62
63          @Override
64          public void onClick(View view) {
65              binding.linear.setOnClickListener(view -> {
66                  int position = getAdapterPosition();
67                  if (position != RecyclerView.NO_POSITION) {
68                      Home home = homeArrayList.get(position);
69                      mListener.onListItemClick(home);
70                  }
71              });
72          }
73
74      }
75
76      @Override
77      public void bind(Home home) {
78          binding.title.setText(home.getTitle());
79          Glide.with(context)
80              .load(home.getImage())
81              .into(binding.homeIcon);
82      }
83  }
```

Figure 6.4

Figure above shows the codes for Home Adapter. It is to display the menu items into the recycler view using ViewHolder.

4.2.3 Profile & Logout

ProfileActivity.java

```
54 | public class ProfileActivity extends AppCompatActivity {
55 |     private ActivityProfileBinding binding;
56 |
57 |     private FirebaseAuth mAuth;
58 |
59 |     private DatabaseReference userRef;
60 |
61 |     private Uri resultUri;
62 |
63 |     private Bitmap bitmap;
64 |
65 |     private StringBuffer strBuf;
66 |
67 |     @Override
68 |     protected void onCreate(Bundle savedInstanceState) {
69 |         super.onCreate(savedInstanceState);
70 |         binding = ActivityProfileBinding.inflate(getLayoutInflater());
71 |         setContentView(binding.getRoot());
72 |
73 |         //initialize
74 |         mAuth = FirebaseAuth.getInstance();
75 |         userRef = FirebaseDatabase.getInstance().getReference("users");
76 |
77 |         binding.backBtn.setOnClickListener(v -> onBackPressed());
78 |         binding.logoutBtn.setOnClickListener(v -> logoutUser());
79 |         specialization = getResources().getStringArray(R.array.specialization_array);
80 |         ArrayAdapter arrayAdapter = new ArrayAdapter(
81 |             context: this,
82 |             android.R.layout.simple_dropdown_item_1line,
83 |             specialization
84 |         );
85 |         //set spinner adapter
```

```
85     //set spinner adapter
86     binding.specializationSpinner.setAdapter(arrayAdapter);
87     //getting profile type
88     binding.specializationSpinner.setOnItemSelectedListener(new MaterialSpinner.OnItemSelectedListener() {
89         @Override
90         public void onItemSelected(@NonNull MaterialSpinner materialSpinner, @Nullable View view, int i, long l) {
91             specializationValue = Objects.requireNonNull(materialSpinner.getSelectedItem()).toString();
92         }
93
94         @Override
95         public void onNothingSelected(@NonNull MaterialSpinner materialSpinner) {
96
97         }
98     });
99
100    binding.editBtn.setOnClickListener(v -> {
101        if (binding.editBtn.getText().toString().equals("Edit")) {
102
103            binding.name.setEnabled(true);
104            binding.dobEt.setEnabled(true);
105            binding.phoneNoEt.setEnabled(true);
106            binding.userImage.setEnabled(true);
107            binding.specializationSpinner.setEnabled(true);
108            binding.qualificationEt.setEnabled(true);
109            binding.editBtn.setText("Update");
110        } else {
111
112            //check validations
113            if (Objects.requireNonNull(binding.name.getText()).toString().isEmpty()) {
114                binding.name.setError("Please enter user name");
115                return;
116            }
117            if (Objects.requireNonNull(binding.phoneNoEt.getText()).toString().isEmpty()) {
118                binding.phoneNoEt.setError("Please enter phone number");
119                return;
120            }
121            if (Objects.requireNonNull(binding.dobEt.getText()).toString().isEmpty()) {
122                binding.dobEt.setError("Please select date of birth");
123                return;
124            }
125        }
126        updateProfile();
127    });
128
129    binding.userImage.setOnClickListener(v -> checkAndroidVersion());
130    binding.dobEt.setOnClickListener(v -> showDatePickerDialog());
131    getUserData();
132
133
134 }
```

```
136     private void logoutUser() {
137         new AlertDialog.Builder( context: this)
138             .setTitle("Logout")
139             .setMessage("Do you really want to logout?")
140             .setIcon(android.R.drawable.ic_dialog_alert)
141             .setPositiveButton(android.R.string.yes, (dialog, whichButton) -> {
142                 SharedPreference settings = getSharedPreference(Constants.sharedPrefName, Context.MODE_PRIVATE);
143                 settings.edit().clear().apply();
144                 mAuth.signOut();
145                 Intent intent = new Intent( packageContext: ProfileActivity.this, AuthActivity.class);
146                 startActivity(intent);
147                 finishAffinity();
148             })
149             .setNegativeButton(android.R.string.no, listener: null).show();
150     }
151
152     private void showDatePickerDialog() {
153         // Create a new OnDateSetListener instance. This listener will be invoked when user click ok button in DatePickerDialog.
154         DatePickerDialog.OnDateSetListener onDateSetListener = (datePicker, year, month, dayOfMonth) -> {
155             strBuf = new StringBuffer();
156             // strBuf.append("You select date is ");
157             if ((month + 1) < 10) {
158                 strBuf.append("0" + (month + 1));
159             } else {
160                 strBuf.append(month + 1);
161             }
162             strBuf.append("-");
163
164             if (dayOfMonth < 10) {
165                 strBuf.append("0" + dayOfMonth);
166             } else {
167                 strBuf.append(dayOfMonth);
168             }
169             strBuf.append("-");
170
171             strBuf.append(year);
172
173             String ages = getAge(year, month: month + 1, dayOfMonth);
174
175             dobDate = strBuf.toString();
176
177             try {
178                 DateFormat inputFormat = new SimpleDateFormat(pattern: "MM-dd-yyyy");
179                 DateFormat outputFormat = new SimpleDateFormat(pattern: "MMM-dd-yyyy");
180                 String inputDateStr = strBuf.toString();
181                 Date date = inputFormat.parse(inputDateStr);
182                 formatDate = outputFormat.format(date);
183             } catch (ParseException e) {
```

```
184         e.printStackTrace();
185     }
186     binding.dobEt.setText(ages + " " + "Years" + " / " + formateDate);
187 }
188 // Get current year, month and day.
189 Calendar now = Calendar.getInstance();
190 int year = now.get(Calendar.YEAR);
191 int month = now.get(Calendar.MONTH);
192 int day = now.get(Calendar.DAY_OF_MONTH);
193 DatePickerDialog datePickerDialog = new DatePickerDialog(context: ProfileActivity.this, onDateSetListener, year, month, day);
194 now.add(Calendar.YEAR, amount: -18);
195
196 datePickerDialog.getDatePicker().setMaxDate(now.getTimeInMillis());
197 datePickerDialog.show();
198 }
199
200 @ private String getAge(int year, int month, int day) {
201
202     Calendar dob = Calendar.getInstance();
203     Calendar today = Calendar.getInstance();
204     int todaymonth = today.get(Calendar.MONTH);
205     int todayyear = today.get(Calendar.YEAR);
206     int today_date = today.get(Calendar.DAY_OF_MONTH);
207     today.set(todayyear, month: todaymonth + 1, date: today_date + 1);
208     dob.set(year, month, day);
209     int age = today.get(Calendar.YEAR) - dob.get(Calendar.YEAR);
210
211     System.out.println("DateFormat1 " + today.get(Calendar.YEAR) + "----" + dob.get(Calendar.YEAR));
212     System.out.println("DateFormat2 " + today.get(Calendar.DAY_OF_YEAR) + "----" + dob.get(Calendar.DAY_OF_YEAR));
213
214     if (today.get(Calendar.DAY_OF_YEAR) < dob.get(Calendar.DAY_OF_YEAR)) {
215         age--;
216     }
217
218     Integer ageInt = new Integer(age);
219     String ageS = ageInt.toString();
220     return ageS;
221 }
222
223 @Override
224 public void onActivityResult(int requestCode, int resultCode, Intent data) {
225
226     //RESULT FROM SELECTED IMAGE
227     super.onActivityResult(requestCode, resultCode, data);
228
229
230     if (resultCode == Activity.RESULT_OK && requestCode == ImagePicker.REQUEST_CODE && data != null) {
231         //Image Uri will not be null for RESULT_OK
232         Uri uri = data.getData();
233
234         resultUri = uri;
235
236         try {
237             bitmap = MediaStore.Images.Media.getBitmap(getApplicationContext(), uri);
238             binding.userImage.setImageBitmap(bitmap);
239         } catch (IOException e) {
240             e.printStackTrace();
241         }
242         // Use Uri object instead of File to avoid storage permissions
243
244     }
245 }
```

```
247     private void getUserData() {
248         binding.loading.setVisibility(View.VISIBLE);
249         userRef.child(Objects.requireNonNull(mAuth.getUid()))
250             .addListenerForSingleValueEvent(new ValueEventListener() {
251                 @Override
252                 public void onDataChange(@NonNull DataSnapshot snapshot) {
253                     if (snapshot.exists()) {
254                         binding.loading.setVisibility(View.GONE);
255                         binding.emailEt.setEnabled(false);
256                         binding.name.setEnabled(false);
257                         binding.dobEt.setEnabled(false);
258                         binding.phoneNoEt.setEnabled(false);
259                         binding.userImage.setEnabled(false);
260                         binding.qualificationEt.setEnabled(false);
261                         binding.specializationSpinner.setEnabled(false);
262
263                         Users users = snapshot.getValue(Users.class);
264
265                         assert users != null;
266                         if (users.getType().equals("doctor")) {
267                             binding.qualificationLayout.setVisibility(View.VISIBLE);
268                             binding.qualificationEt.setText(users.getQualification());
269                             binding.specializationSpinner.setVisibility(View.VISIBLE);
270                             specializationValue = users.getSpecialization();
271                         }
272
273                         binding.emailEt.setText(users.getEmail());
274                         binding.name.setText(users.getName());
275                         binding.dobEt.setText(users.getDob());
276                         binding.phoneNoEt.setText(users.getPhone());
277                         dobDate = users.getDob();
278
279                         if (!users.getImage().equals("")) {
280                             Glide.with(ProfileActivity.this)
281                                 .load(users.getImage())
282                                 .into(binding.userImage);
283                         }
284                     }
285
286                 }
287             });
288
289             @Override
290             public void onCancelled(@NonNull DatabaseError error) {
291                 Snackbar snackbar = Snackbar
292                     .make(binding.getRoot(), error.getMessage(), Snackbar.LENGTH_LONG);
293                 snackbar.show();
294                 binding.loading.setVisibility(View.GONE);
295             });
296
297         public void checkAndroidversion() {
298             if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
299                 try {
300                     requestPermissions(new String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE, Manifest.permission.READ_EXTERNAL_STORAGE, Manifest.permission.CAMERA}, requestCode: 555);
301                 } catch (Exception e) {
302                 }
303             }
304         } else {
305             ImagePicker.with(ProfileActivity.this)
306                 .crop() //Crop image(Optional), Check Customization for more option
307                 .compress(1024) //Final image size will be less than 1 MB(Optional)
308                 .maxResultSize(width: 1080, height: 1080) //Final image resolution will be less than 1080 x 1080(Optional)
309                 .start();
310         }
311     }

```

```
328     private void updateProfile() {
329         if (resultUri != null) {
330             binding.loading.setVisibility(View.VISIBLE);
331
332             final StorageReference filePath = FirebaseStorage.getInstance().getReference().child("Users").child(resultUri.getLastPathSegment());
333             Bitmap bitmap = null;
334             try {
335                 bitmap = MediaStore.Images.Media.getBitmap(this.getContentResolver(), resultUri);
336             } catch (IOException e) {
337                 e.printStackTrace();
338             }
339
340             ByteArrayOutputStream baos = new ByteArrayOutputStream();
341             bitmap.compress(Bitmap.CompressFormat.JPEG, quality: 20, baos);
342             byte[] data = baos.toByteArray();
343             UploadTask uploadTask = filePath.putBytes(data);
344
345             uploadTask.addOnFailureListener(e -> {
346                 binding.loading.setVisibility(View.GONE);
347                 Toast.makeText(context: ProfileActivity.this, text: "" + e.getMessage(), Toast.LENGTH_SHORT).show();
348                 return;
349             });
350             uploadTask.addOnSuccessListener(taskSnapshot -> filePath.getDownloadUrl().addOnSuccessListener(uri -> {
351
352                 Map<String, Object> updateValue = new HashMap<>();
353                 updateValue.put("name", binding.name.getText().toString());
354                 updateValue.put("phone", binding.phoneNoEt.getText().toString());
355                 updateValue.put("dob", dobDate);
356                 updateValue.put("image", uri.toString());
357                 updateValue.put("specialization", specializationValue);
358                 if (!binding.qualificationEt.getText().toString().isEmpty()) {
359                     updateValue.put("qualification", binding.qualificationEt.getText().toString());
360
361                 } else {
362                     updateValue.put("qualification", qualifications);
363                 }
364
365                 userRef.child(auth.getUid()).updateChildren(updateValue).addOnCompleteListener(task -> {
366                     binding.loading.setVisibility(View.GONE);
367                     Toast.makeText(context: ProfileActivity.this, text: "Profile Updated Successfully", Toast.LENGTH_SHORT).show();
368                     onBackPressed();
369
370                 }).addOnFailureListener(e -> {
371                     binding.loading.setVisibility(View.GONE);
372                     Toast.makeText(context: ProfileActivity.this, text: "" + e.getMessage(), Toast.LENGTH_SHORT).show();
373                 });
374                 return;
375
376             }).addOnFailureListener(exception -> {
377                 binding.loading.setVisibility(View.GONE);
378                 Toast.makeText(context: ProfileActivity.this, text: "" + exception.getMessage(), Toast.LENGTH_SHORT).show();
379             }));
380         } else {
381             binding.loading.setVisibility(View.VISIBLE);
382
383             Map<String, Object> updateValue = new HashMap<>();
384             updateValue.put("name", binding.name.getText().toString());
385             updateValue.put("phone", binding.phoneNoEt.getText().toString());
386             updateValue.put("dob", dobDate);
387             updateValue.put("specialization", specializationValue);
388             if (!binding.qualificationEt.getText().toString().isEmpty()) {
389                 updateValue.put("qualification", binding.qualificationEt.getText().toString());
390             } else {
391                 updateValue.put("qualification", qualifications);
392             }
393
394         }
395     }
396 }
```



```
395     userRef.child(mAuth.getUid()).updateChildren(updateValue)
396         .addOnCompleteListener(task -> {
397             binding.loading.setVisibility(View.GONE);
398             Toast.makeText( context: ProfileActivity.this, text: "Profile Updated Successfully", Toast.LENGTH_SHORT).show();
399             onBackPressed();
400         }
401         .addOnFailureListener(e -> {
402             binding.loading.setVisibility(View.GONE);
403             Toast.makeText( context: ProfileActivity.this, text: "" + e.getMessage(), Toast.LENGTH_SHORT).show();
404         });
405     }
406 }
407 }
```

Figure 6.5

Figure above shows the code for the profile page of admin, doctors, and users. `ProfileActivity.java` is implemented to show and update the information of admin, doctors, and users. `logoutUser()` function is used to sign users out when the button is clicked.

4.2.4 Medic Wiki

MedicWikiActivity.java

```
16  public class MedicWikiActivity extends AppCompatActivity implements MedicWikiAdapter.OnItemClickListener {
17      private ActivityMedicWikiBinding binding;
18      private MedicWikiAdapter mAdapter;
19      private ArrayList<MedicWiki> medicWikiArrayList = new ArrayList<>();
20      @Override
21      protected void onCreate(Bundle savedInstanceState) {
22          super.onCreate(savedInstanceState);
23          binding = ActivityMedicWikiBinding.inflate(getLayoutInflater());
24          setContentView(binding.getRoot());
25
26          binding.backBtn.setOnClickListener(v -> onBackPressed());
27
28          getWikiData();
29      }
30
31      private void getWikiData() {
32          medicWikiArrayList.add(new MedicWiki( id: "1", title: "Covid and Its Effects", R.drawable.covid_image));
33          medicWikiArrayList.add(new MedicWiki( id: "2", title: "Learn About Vaccines", R.drawable.covid_image_2));
34          medicWikiArrayList.add(new MedicWiki( id: "3", title: "Fight Against Cancer", R.drawable.cancer_image));
35          medicWikiArrayList.add(new MedicWiki( id: "4", title: "Healthy Eating and\nLifestyle", R.drawable.health_image));
36
37          mAdapter = new MedicWikiAdapter( context: this, medicWikiArrayList);
38          binding.medicWikiRv.setLayoutManager(new LinearLayoutManager( context: MedicWikiActivity.this));
39          binding.medicWikiRv.setAdapter(mAdapter);
40          mAdapter.setOnItemClickListener(this);
41      }
42
43      @Override
44      public void onListItemClick(MedicWiki home) {
45          Intent intent = new Intent( packageContext: MedicWikiActivity.this, MedicWikiDetailActivity.class);
46          intent.putExtra( name: "title", home.getTitle());
47          intent.putExtra( name: "image", home.getImage());
48          startActivity(intent);
49      }
50  }
```

Figure 6.6

The figure above shows the codes for the Medic Wiki feature. `getWikiData()` is implemented to get information to store into `medicWikiArrayList`.

```
MedicWikiAdapter.java
16  public class MedicWikiAdapter extends RecyclerView.Adapter<MedicWikiAdapter.ViewHolder> {
17      private Context context;
18      ArrayList<MedicWiki> homeArrayList;
19      OnItemClickListener mListener;
20
21      public MedicWikiAdapter(Context context, ArrayList<MedicWiki> homeArrayList) {
22          this.context = context;
23          this.homeArrayList = homeArrayList;
24      }
25
26      public interface OnItemClickListener {
27          void onListItemClick(MedicWiki home);
28      }
29
30      public void setOnItemClickListener(OnItemClickListener listener) { mListener = listener; }
31
32      @NotNull
33      @Override
34      public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
35          CustomMedicWikilayoutBinding binding = CustomMedicWikilayoutBinding
36              .inflate(
37                  LayoutInflater.from(parent.getContext()),
38                  parent,
39                  attachToParent: false);
40
41          return new ViewHolder(binding);
42      }
43
44      @Override
45      public void onBindViewHolder(@NotNull final ViewHolder holder, int position) {
46          MedicWiki home = homeArrayList.get(position);
47          holder.bind(home);
48      }
49
50      @Override
51      public int getItemCount() { return homeArrayList.size(); }
52
53      public class ViewHolder extends RecyclerView.ViewHolder {
54          private final CustomMedicWikilayoutBinding binding;
55
56          @Override
57          public ViewHolder(CustomMedicWikilayoutBinding binding) {
58              super(binding.getRoot());
59              this.binding = binding;
60
61              binding.cardView.setOnClickListener(view -> {
62                  int position = getAdapterPosition();
63                  if (position != RecyclerView.NO_POSITION) {
64                      MedicWiki home = homeArrayList.get(position);
65                      mListener.onListItemClick(home);
66                  }
67              });
68          }
69
70          public void bind(MedicWiki home) {
71              binding.title.setText(home.getTitle());
72              Glide.with(context)
73                  .load(home.getImage())
74                  .into(binding.image);
75          }
76      }
77  }
```

Figure 6.7

The figure above shows the codes for the MedicWikiAdapter. The adapter is used to display the information into a cardview using a viewholder.

MedicWikiDetailActivity.java

```
10  public class MedicWikiDetailActivity extends AppCompatActivity {
11      private ActivityMedicWikiDetailBinding binding;
12
13      @Override
14      protected void onCreate(Bundle savedInstanceState) {
15          super.onCreate(savedInstanceState);
16          binding = ActivityMedicWikiDetailBinding.inflate(getLayoutInflater());
17          setContentView(binding.getRoot());
18
19          binding.backBtn.setOnClickListener(v -> onBackPressed());
20
21          Bundle bundle = getIntent().getExtras();
22          if (bundle != null) {
23              Glide.with( activity: this)
24                  .load(bundle.getInt( key: "image"))
25                  .into(binding.wikiImage);
26
27              binding.title.setText(bundle.getString( key: "title"));
28          }
29
30      }
31  }
```

Figure 6.8

The figure above shows the codes for the MedicWikiDetailActivity. It is the main activity where the information and articles are published.

4.2.5 Loading Screen

SplashActivity.java

```
16 17  public class SplashActivity extends AppCompatActivity {
18 19      private ActivitySplashBinding binding;
20      private FirebaseAuth mAuth;
21
22 23      @Override
24 25      protected void onCreate(Bundle savedInstanceState) {
26          super.onCreate(savedInstanceState);
27          binding = ActivitySplashBinding.inflate(getLayoutInflater());
28          setContentView(binding.getRoot());
29
30          mAuth = FirebaseAuth.getInstance();
31          new Handler().postDelayed(() -> {
32              //if user logged in
33              if (mAuth.getCurrentUser() != null){
34                  SharedPreferences sharedpreferences = getSharedPreferences(Constants.sharedPrefName, MODE_PRIVATE);
35                  String userType = sharedpreferences.getString(Constants.userType, defaultValue: "");
36                  if (userType.equals("admin")) {
37                      Intent intent = new Intent(packageContext: SplashActivity.this, AdminHomeActivity.class);
38                      startActivity(intent);
39                      finish();
40                  } else if (userType.equals("doctor")) {
41                      Intent intent = new Intent(packageContext: SplashActivity.this, DoctorHomeActivity.class);
42                      startActivity(intent);
43                      finish();
44                  } else if (userType.equals("user")) {
45                      Intent intent = new Intent(packageContext: SplashActivity.this, UserHomeActivity.class);
46                      startActivity(intent);
47                      finish();
48                  }
49              }else {
50                  startActivity(new Intent(packageContext: SplashActivity.this, AuthActivity.class));
51              }
52              finish();
53          }, delayMillis: 3000);
54 }
```

Figure 6.9

The figure above shows the codes for the SplashActivity. It is the loading screen when the application is launched, and the duration is set to 3 seconds.

4.2.6 Add Doctors

AddDoctorActivity.java

```
31  public class AddDoctorActivity extends AppCompatActivity {
32      private ActivityAddDoctorBinding binding;
33      private String[] specialization;
34      private String specializationValue = "";
35      private StringBuffer strBuf;
36      private String dobDate, formatDate;
37      private DatabaseReference userRef;
38      private FirebaseAuth mAuth;
39
40      @Override
41      protected void onCreate(Bundle savedInstanceState) {
42          super.onCreate(savedInstanceState);
43          binding = ActivityAddDoctorBinding.inflate(getLayoutInflater());
44          setContentView(binding.getRoot());
45
46          //initialize
47          userRef = FirebaseDatabase.getInstance().getReference("users");
48          mAuth = FirebaseAuth.getInstance();
49
50          specialization = getResources().getStringArray(R.array.specialization_array);
51          //click listeners
52          binding.backArrow.setOnClickListener(view -> onBackPressed());
53
54          binding.dobEt.setOnClickListener(v -> showDatePickerDialog());
55          ArrayAdapter arrayAdapter = new ArrayAdapter(
56              context, this,
57              android.R.layout.simple_dropdown_item_1line,
58              specialization
59          );
60          //set spinner adapter
61          binding.specializationSpinner.setAdapter(arrayAdapter);
62
63      }
```

```
63     //getting profile type
64     binding.specializationSpinner.setOnItemSelectedListener(new MaterialSpinner.OnItemSelectedListener() {
65         @Override
66         public void onItemSelected(@NonNull MaterialSpinner materialSpinner, @Nullable View view, int i, long l) {
67             specializationValue = Objects.requireNonNull(materialSpinner.getSelectedItem()).toString();
68         }
69
70         @Override
71         public void onNothingSelected(@NonNull MaterialSpinner materialSpinner) {
72
73         }
74     });
75
76     binding.registerDoctorBtn.setOnClickListener(view -> {
77
78         //check validations
79         if (Objects.requireNonNull(binding.name.getText()).toString().isEmpty()) {
80             binding.name.setError("Please enter user name");
81             return;
82         }
83         if (Objects.requireNonNull(binding.phoneNoEt.getText()).toString().isEmpty()) {
84             binding.phoneNoEt.setError("Please enter phone number");
85             return;
86         }
87         if (Objects.requireNonNull(binding.emailEt.getText()).toString().isEmpty()) {
88             binding.emailEt.setError("Please enter valid email");
89             return;
90         }
91         if (Objects.requireNonNull(binding.passwordEt.getText()).toString().isEmpty()) {
92             binding.passwordEt.setError("Please enter password");
93             return;
94         }
95     });

```

```

95     if (Objects.requireNonNull(binding.dobEt.getText()).toString().isEmpty()) {
96         binding.dobEt.setError("Please select date of birth");
97         return;
98     }
99     if (Objects.requireNonNull(binding.qualificationEt.getText()).toString().isEmpty()) {
100        binding.dobEt.setError("Please select qualification");
101        return;
102    }
103    if (specializationValue.equals("")) {
104        Toast.makeText( context: this, text: "Please select specialization", Toast.LENGTH_SHORT).show();
105        return;
106    }
107    //register doctor
108    registerDoctor();
109 });
110 }
111 }
112
113 private void registerDoctor() {
114     binding.loading.setVisibility(View.VISIBLE);
115     binding.registerDoctorBtn.setEnabled(false);
116     mAuth.createUserWithEmailAndPassword(Objects.requireNonNull(binding.emailEt.getText()).toString(),
117                                         Objects.requireNonNull(binding.passwordEt.getText()).toString())
118         .addOnCompleteListener(task -> {
119             if (task.isSuccessful()) {
120
121                 //store user detail in firebase
122                 Users users;
123                 users = new Users(
124                     mAuth.getUid(),
125                     Objects.requireNonNull(binding.name.getText()).toString(),
126                     Objects.requireNonNull(binding.phoneNoEt.getText()).toString(),
127                     dobDate,
128
129                     type: "doctor",
130                     binding.emailEt.getText().toString(),
131                     image: "",
132                     specializationValue,
133                     binding.qualificationEt.getText().toString()
134                 );
135
136                 userRef.child(Objects.requireNonNull(mAuth.getUid())).setValue(users).addOnCompleteListener(task1 -> {
137                     if (task1.isSuccessful()) {
138                         binding.loading.setVisibility(View.GONE);
139
140                         mAuth.signOut();
141
142                         SharedPreferences sharedpreferences = getSharedPreferences(Constants.sharedPrefName, MODE_PRIVATE);
143
144                         mAuth.signInWithEmailAndPassword(sharedpreferences.getString(Constants.adminEmail, defaultValue: ""));
145
146                         sharedpreferences.getString(Constants.adminPassword, defaultValue: "").addOnCompleteListener(task2 -> {
147                             Snackbar snackbar = Snackbar
148                                 .make(binding.getRoot(), text: "Doctor Added Successfully", Snackbar.LENGTH_LONG);
149                             snackbar.show();
150                             onBackPressed();
151                         });
152                     });
153
154                 });
155             }.addOnFailureListener(e -> {
156                 binding.loading.setVisibility(View.GONE);
157                 binding.registerDoctorBtn.setEnabled(true);
158                 Snackbar snackbar = Snackbar
159                     .make(binding.getRoot(), Objects.requireNonNull(e.getMessage()), Snackbar.LENGTH_LONG);
160                 snackbar.show();
161             });
162         });
163     });
164 }

```

```

166     private void showDatePickerDialog() {
167         // Create a new OnDateSetListener instance. This listener will be invoked when user click ok button in DatePickerDialog.
168         DatePickerDialog.OnDateSetListener onDateSetListener = (datePicker, year, month, dayOfMonth) -> {
169             strBuf = new StringBuffer();
170             // strBuf.append("You select date is ");
171             if ((month + 1) < 10) {
172                 strBuf.append("0" + (month + 1));
173             } else {
174                 strBuf.append(month + 1);
175             }
176             strBuf.append("-");
177             if (dayOfMonth < 10) {
178                 strBuf.append("0" + dayOfMonth);
179             } else {
180                 strBuf.append(dayOfMonth);
181             }
182             strBuf.append("-");
183             strBuf.append(year);
184             String ages = getAge(year, month, month + 1, dayOfMonth);
185             dobDate = strBuf.toString();
186
187             try {
188                 DateFormat inputFormat = new SimpleDateFormat( pattern: "MM-dd-yyyy");
189                 DateFormat outputFormat = new SimpleDateFormat( pattern: "MMM-dd-yyyy");
190                 String inputDateStr = strBuf.toString();
191                 Date date = inputFormat.parse(inputDateStr);
192                 formateDate = outputFormat.format(date);
193             } catch (ParseException e) {
194                 e.printStackTrace();
195             }
196
197             binding.dobEt.setText(ages + " " + "Years" + " / " + formateDate);
198         };
199         // Get current year, month and day.
200         Calendar now = Calendar.getInstance();
201         int year = now.get(Calendar.YEAR);
202         int month = now.get(Calendar.MONTH);
203         int day = now.get(Calendar.DAY_OF_MONTH);
204         DatePickerDialog datePickerDialog = new DatePickerDialog( context: AddDoctorActivity.this, onDateSetListener, year, month, day);
205         now.add(Calendar.YEAR, amount: -18);
206
207         datePickerDialog.getDatePicker().setMaxDate(now.getTimeInMillis());
208         datePickerDialog.show();
209     }
210
211     @
212     private String getAge(int year, int month, int day) {
213
214         Calendar dob = Calendar.getInstance();
215         Calendar today = Calendar.getInstance();
216         int todaymonth = today.get(Calendar.MONTH);
217         int todayyear = today.get(Calendar.YEAR);
218         int today_date = today.get(Calendar.DAY_OF_MONTH);
219         today.set(todayyear, month: todaymonth + 1, date: today_date + 1);
220         dob.set(year, month, day);
221         int age = today.get(Calendar.YEAR) - dob.get(Calendar.YEAR);
222
223         System.out.println("DateFormat1 " + today.get(Calendar.YEAR) + "----" + dob.get(Calendar.YEAR));
224         System.out.println("DateFormat2 " + today.get(Calendar.DAY_OF_YEAR) + "----" + dob.get(Calendar.DAY_OF_YEAR));
225
226         if (today.get(Calendar.DAY_OF_YEAR) < dob.get(Calendar.DAY_OF_YEAR)) {
227             age--;
228         }
229
230         Integer ageInt = new Integer(age);
231         String ageS = ageInt.toString();
232         return ages;
233     }

```

Figure 6.10

The figure above shows the codes for the AddDoctorActivity. It is where the admin adds doctors into the database using registerDoctor() function. It uses similar logic as the register account for users which is createUserWithEmailAndPassoword. However, the user type will be 'doctor' and if the doctor is registered successfully, it will display the toast message "Doctor Added Successfully".

DoctorActivity.java

UserActivity.java

```
26 public class DoctorsActivity extends AppCompatActivity {
27     private ActivityDoctorsBinding binding;
28     private ArrayList<Users> usersArrayList = new ArrayList<>();
29     private UserAdapter mAdapter;
30     private FirebaseAuth mAuth;
31     private DatabaseReference userRef;
32     @Override
33     protected void onCreate(Bundle savedInstanceState) {
34         super.onCreate(savedInstanceState);
35         binding = ActivityDoctorsBinding.inflate(getLayoutInflater());
36         setContentView(binding.getRoot());
37
38         //initialize
39         mAuth = FirebaseAuth.getInstance();
40         userRef = FirebaseDatabase.getInstance().getReference("users");
41         binding.backBtn.setOnClickListener(v -> onBackPressed());
42
43         binding.addDoctorBtn.setOnClickListener(v -> startActivity(new Intent(getApplicationContext(), DoctorsActivity.this, AddDoctorActivity.class)));
44
45         binding.searchEt.addTextChangedListener(new TextWatcher() {
46             @Override
47             public void beforeTextChanged(CharSequence charSequence, int i, int i1, int i2) {
48
49             }
50             @Override
51             public void onTextChanged(CharSequence charSequence, int i, int i1, int i2) {
52                 if (!binding.searchEt.getText().toString().isEmpty()) {
53                     final ArrayList<Users> filterModList = filter(usersArrayList, binding.searchEt.getText().toString());
54                     if (!filterModList.isEmpty())
55                         mAdapter.setFilter(filterModList);
56                 } else {
57                     getDoctors();
58                 }
59             }
60         });
61     }
62
63     private void getDoctors() {
64         userRef.addValueEventListener(new ValueEventListener() {
65             @Override
66             public void onDataChange(@NonNull DataSnapshot snapshot) {
67                 usersArrayList.clear();
68                 for (DataSnapshot dataSnapshot : snapshot.getChildren()) {
69                     Users users = dataSnapshot.getValue(Users.class);
70                     usersArrayList.add(users);
71                 }
72                 mAdapter.notifyDataSetChanged();
73             }
74             @Override
75             public void onCancelled(@NonNull DatabaseError error) {
76                 Log.e("TAG", "Error getting data: " + error.getMessage());
77             }
78         });
79     }
80
81     private ArrayList<Users> filter(ArrayList<Users> usersArrayList, String query) {
82         ArrayList<Users> filteredModList = new ArrayList<>();
83         for (Users users : usersArrayList) {
84             if (users.getName().toLowerCase().contains(query.toLowerCase()))
85                 filteredModList.add(users);
86         }
87         return filteredModList;
88     }
89
90     @Override
91     public void onBackPressed() {
92         super.onBackPressed();
93         Intent intent = new Intent(getApplicationContext(), MainActivity.class);
94         intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
95         intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
96         startActivity(intent);
97     }
98 }
```

```

61         @Override
62     public void afterTextChanged(Editable editable) {
63
64     }
65 });
66 }
67 }
68 }
69
70 @Override
71 protected void onStart() {
72     super.onStart();
73     getDoctors();
74 }
75
76 private void getDoctors() {
77     usersArrayList.clear();
78     binding.loading.setVisibility(View.VISIBLE);
79     userRef.orderByChild("type").equalTo("doctor").addValueEventListener(new ValueEventListener() {
80
81     @Override
82     public void onDataChange(@NonNull DataSnapshot snapshot) {
83         if (snapshot.exists()) {
84             binding.loading.setVisibility(View.GONE);
85
86             for (DataSnapshot ds: snapshot.getChildren()){
87                 Users users = ds.getValue(Users.class);
88                 usersArrayList.add(users);
89             }
90             mAdapter = new UserAdapter(context: DoctorsActivity.this, usersArrayList);
91             binding.doctorRv.setLayoutManager(new LinearLayoutManager(context: DoctorsActivity.this));
92             binding.doctorRv.setAdapter(mAdapter);
93
94         } else {
95             binding.loading.setVisibility(View.GONE);
96         }
97     }
98
99     @Override
100    public void onCancelled(@NonNull DatabaseError error) {
101        binding.loading.setVisibility(View.GONE);
102
103        Snackbar snackbar = Snackbar
104            .make(binding.getRoot(), error.getMessage(), Snackbar.LENGTH_LONG);
105        snackbar.show();
106    };
107 };
108 }
109
110 //getting filter list
111 @
112 private ArrayList<Users> filter(ArrayList<Users> hi, String query) {
113     query = query.toLowerCase();
114     final ArrayList<Users> filterModelList = new ArrayList<>();
115     for (Users model : hi) {
116
117         final String sTitle = model.getName().toLowerCase();
118
119         if (sTitle.startsWith(query)) {
120             filterModelList.add(model);
121         }
122     }
123     return filterModelList;
124 }
125 }
126 }

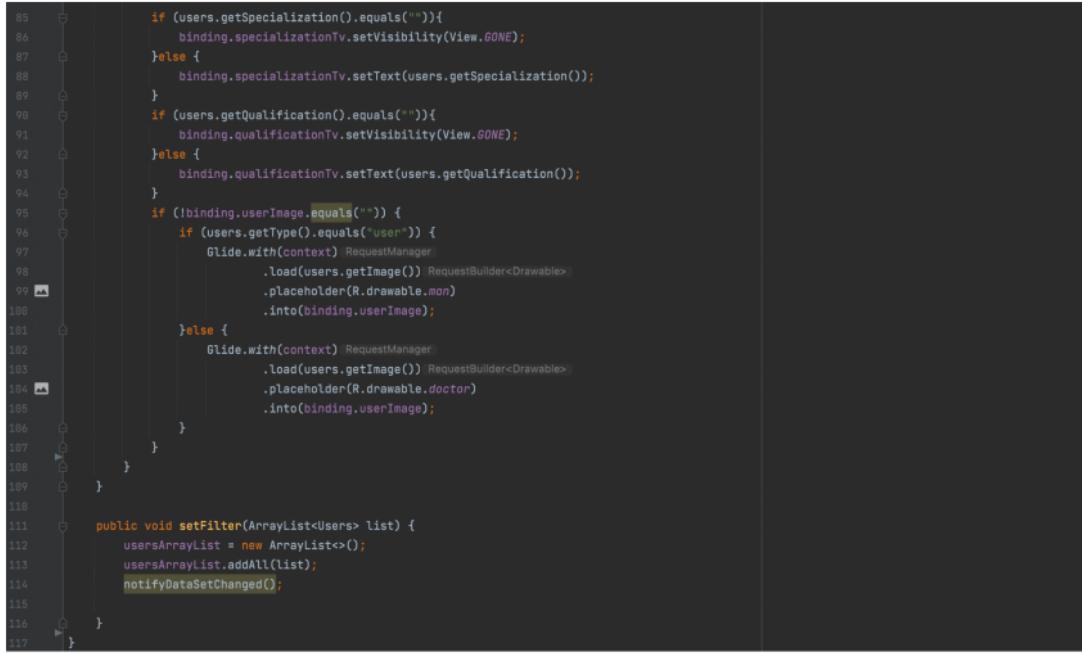
```

Figure 6.11

The figure above shows the codes for the DoctorActivity but UserActivity is similar. It is the main activity where it handles doctors and users in the admin page. getDoctors() function is used to fetch data from the database and display them into the array. Similarly for UserActivity, getUsers() function is used to fetch data from the database and display them into the array.

UserAdapter.java

```
17
18  public class UserAdapter extends RecyclerView.Adapter<UserAdapter.ViewHolder> {
19      private Context context;
20      ArrayList<Users> usersArrayList;
21      OnItemClickListener mListener;
22
23      public UserAdapter(Context context, ArrayList<Users> usersArrayList) {
24          this.context = context;
25          this.usersArrayList = usersArrayList;
26      }
27
28      public interface OnItemClickListener {
29          void onListItemClick(Users users);
30      }
31
32      public void setOnItemClickListener(OnItemClickListener listener) { mListener = listener; }
33
34      @NonNull
35      @Override
36      public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
37          CustomUsersLayoutBinding binding = CustomUsersLayoutBinding
38              .inflate(
39                  LayoutInflater.from(parent.getContext()),
40                  parent,
41                  false);
42
43          return new ViewHolder(binding);
44      }
45
46      @Override
47      public void onBindViewHolder(@NonNull final ViewHolder holder, int position) {
48          Users users = usersArrayList.get(position);
49          holder.bind(users);
50      }
51
52      @Override
53      public int getItemCount() { return usersArrayList.size(); }
54
55
56      public class ViewHolder extends RecyclerView.ViewHolder {
57          private final CustomUsersLayoutBinding binding;
58
59          public ViewHolder(CustomUsersLayoutBinding binding) {
60              super(binding.getRoot());
61              this.binding = binding;
62
63              binding.bookCardView.setOnClickListener(view -> {
64                  int position = getAdapterPosition();
65                  if (position != RecyclerView.NO_POSITION) {
66                      Users users = usersArrayList.get(position);
67                      mListener.onListItemClick(users);
68                  }
69              });
70          }
71
72          public void bind(Users users) {
73              binding.userName.setText(users.getName());
74              binding.userEmail.setText(users.getEmail());
75              binding.userPhone.setText(users.getPhone());
76
77              if(users.getType().equals("doctor")){
78                  binding.bookCardView.setVisibility(View.VISIBLE);
79              }
80          }
81      }
82
83  }
```



```
85         if (users.getSpecialization().equals("")){
86             binding.specializationTv.setVisibility(View.GONE);
87         }else {
88             binding.specializationTv.setText(users.getSpecialization());
89         }
90         if (users.getQualification().equals("")){
91             binding.qualificationTv.setVisibility(View.GONE);
92         }else {
93             binding.qualificationTv.setText(users.getQualification());
94         }
95         if (!binding.userImage.equals("")) {
96             if (users.getType().equals("user")) {
97                 Glide.with(context) RequestManager
98                     .load(users.getImage()) RequestBuilder<Drawable>
99                         .placeholder(R.drawable.mon)
100                         .into(binding.userImage);
101             }else {
102                 Glide.with(context) RequestManager
103                     .load(users.getImage()) RequestBuilder<Drawable>
104                         .placeholder(R.drawable.doctor)
105                         .into(binding.userImage);
106             }
107         }
108     }
109 }
110
111 public void setFilter(ArrayList<Users> list) {
112     usersArrayList = new ArrayList<>();
113     usersArrayList.addAll(list);
114     notifyDataSetChanged();
115 }
116
117 }
```

Figure 6.12

The figure above shows the codes for the UserAdapter. It is to display the users information from the array into their respective cardview using a viewholder.

4.2.5 Health Data

AddHealthDataActivity.java

```
17 public class AddHealthDataActivity extends AppCompatActivity {
18     private ActivityAddHealthDataBinding binding;
19     private FirebaseAuth mAuth;
20     private DatabaseReference healthRef;
21
22     @Override
23     protected void onCreate(Bundle savedInstanceState) {
24         super.onCreate(savedInstanceState);
25         binding = ActivityAddHealthDataBinding.inflate(getLayoutInflater());
26         setContentView(binding.getRoot());
27
28         //initialize
29         mAuth = FirebaseAuth.getInstance();
30         healthRef = FirebaseDatabase.getInstance().getReference("path: \"health\"").child(mAuth.getUid());
31
32         binding.backBtn.setOnClickListener(v -> onBackPressed());
33         binding.addDataBtn.setOnClickListener(v -> {
34             //check validations
35             if (Objects.requireNonNull(binding.weightEt.getText()).toString().isEmpty()) {
36                 binding.weightEt.setError("Please enter weight");
37                 return;
38             }
39             if (Objects.requireNonNull(binding.heightEt.getText()).toString().isEmpty()) {
40                 binding.heightEt.setError("Please enter height");
41                 return;
42             }
43             if (Objects.requireNonNull(binding.bmiEt.getText()).toString().isEmpty()) {
44                 binding.bmiEt.setError("Please enter valid BMI");
45                 return;
46             }
47             if (Objects.requireNonNull(binding.bloodPressureEt.getText()).toString().isEmpty()) {
48                 binding.bloodPressureEt.setError("Please enter blood pressure");
49                 return;
50             }
51             addHealthData();
52         });
53     }
54
55     private void addHealthData() {
56         binding.loading.setVisibility(View.VISIBLE);
57
58         Health health = new Health(
59             binding.weightEt.getText().toString(),
60             binding.heightEt.getText().toString(),
61             binding.bmiEt.getText().toString(),
62             binding.bloodPressureEt.getText().toString()
63         );
64
65         healthRef.setValue(health)
66             .addOnCompleteListener(task -> {
67                 if (task.isSuccessful()) {
68                     binding.loading.setVisibility(View.GONE);
69
70                     Toast.makeText(context: AddHealthDataActivity.this, text: "Health data added successfully", Toast.LENGTH_SHORT).show();
71                     onBackPressed();
72
73                 }
74             }).addOnFailureListener(e -> {
75                 binding.loading.setVisibility(View.GONE);
76                 Toast.makeText(context: AddHealthDataActivity.this, e.getMessage(), Toast.LENGTH_SHORT);
77             });
78
79     }
80 }
```

Figure 6.13

The figure above shows the codes for the AddHealthDataActivity. It is the activity that allows the users to add their health data in the application using addHealthData() function.

HealthStatsActivity.java

```
21  public class HealthStatsActivity extends AppCompatActivity {
22      private ActivityHealthStatusBinding binding;
23      private FirebaseAuth mAuth;
24      private DatabaseReference healthRef;
25
26      @Override
27      protected void onCreate(Bundle savedInstanceState) {
28          super.onCreate(savedInstanceState);
29          binding = ActivityHealthStatusBinding.inflate(getLayoutInflater());
30          setContentView(binding.getRoot());
31
32          //initialize firebase
33          mAuth = FirebaseAuth.getInstance();
34          healthRef = FirebaseDatabase.getInstance().getReference("health").child(Objects.requireNonNull(mAuth.getUid()));
35
36          binding.backBtn.setOnClickListener(v -> onBackPressed());
37          binding.addDataBtn.setOnClickListener(v -> startActivity(new Intent(getApplicationContext(), AddHealthDataActivity.class)));
38      }
39
40      @Override
41      protected void onStart() {
42          super.onStart();
43          getHealthData();
44      }
45
46      private void getHealthData() {
47          binding.loading.setVisibility(View.VISIBLE);
48          healthRef.addValueEventListener(new ValueEventListener() {
49              @Override
50              public void onDataChange(@NonNull DataSnapshot snapshot) {
51                  binding.loading.setVisibility(View.GONE);
52
53                  if (snapshot.exists()) {
54
55                      Health health = snapshot.getValue(Health.class);
56                      binding.weightTv.setText(health.getWeight());
57                      binding.heightTv.setText(health.getHeight());
58                      binding.bmiTv.setText(health.getBmi());
59                      binding.bloodPressureTv.setText(health.getBloodPressure());
60
61                  }
62
63              }
64
65              @Override
66              public void onCancelled(@NonNull DatabaseError error) {
67                  binding.loading.setVisibility(View.GONE);
68              }
69          });
70      }
71  }
```

Figure 6.14

The figure above shows the codes for the HealthStatsActivity. It is the main activity that handles the health data feature in the application. If the current user id is validated, it will display the health data accordingly. To simplify the fetching of data, Health Class is created where we can getWeight(), getHeight, getBmi() and getBloodPressure() just by accessing the Class.

4.2.7 Appointment

MedicalProfessionalsActivity.java

```
17  public class MedicalProfessionalsActivity extends AppCompatActivity implements SpecialistAdapter.OnItemSelectedListener {
18      private ActivityMedicalProfessionalsBinding binding;
19      private SpecialistAdapter mAdapter;
20      private ArrayList<Specialist> specialistArrayList = new ArrayList<>();
21
22      @Override
23      protected void onCreate(Bundle savedInstanceState) {
24          super.onCreate(savedInstanceState);
25          binding = ActivityMedicalProfessionalsBinding.inflate(LayoutInflater.from(this));
26          setContentView(binding.getRoot());
27
28          binding.backBtn.setOnClickListener(v -> onBackPressed());
29
30          setUpSpecialist();
31
32          //search
33          binding.searchEt.addTextChangedListener(new TextWatcher() {
34              @Override
35              public void beforeTextChanged(CharSequence charSequence, int i, int i1, int i2) {
36
37                  }
38
39                  @Override
40                  public void onTextChanged(CharSequence charSequence, int i, int i1, int i2) {
41                      if (!binding.searchEt.getText().toString().isEmpty()) {
42                          final ArrayList<Specialist> filterModList = filter(specialistArrayList, binding.searchEt.getText().toString());
43                          if (!filterModList.isEmpty())
44                              mAdapter.setFilter(filterModList);
45                      } else {
46                          setUpSpecialist();
47                      }
48                  }
49          });
50      }
51
52      private void setUpSpecialist() {
53          mAdapter = new SpecialistAdapter(this, specialistArrayList);
54          binding.list.setAdapter(mAdapter);
55      }
56
57      @Override
58      public void onItemSelected(Specialist specialist) {
59          Intent intent = new Intent(this, SpecialistDetailActivity.class);
60          intent.putExtra("SPECIALIST", specialist);
61          startActivity(intent);
62      }
63  }
```

```
67
50     @Override
51     public void afterTextChanged(Editable editable) {
52     }
53   });
54 }
55
56 private void setUpSpecialist() {
57   specialistArrayList.clear();
58   specialistArrayList.add(new Specialist(id: "1", title: "Anesthesiology"));
59   specialistArrayList.add(new Specialist(id: "2", title: "Cardiology"));
60   specialistArrayList.add(new Specialist(id: "3", title: "Cardiovascular/Thoracic Surgery"));
61   specialistArrayList.add(new Specialist(id: "4", title: "Clinical Immunology/Allergy"));
62   specialistArrayList.add(new Specialist(id: "5", title: "Dentistry"));
63   specialistArrayList.add(new Specialist(id: "6", title: "Dermatology"));
64   specialistArrayList.add(new Specialist(id: "7", title: "Gastroenterology"));
65   specialistArrayList.add(new Specialist(id: "7", title: "General Surgery"));
66   specialistArrayList.add(new Specialist(id: "8", title: "General/Clinical Pathology"));
67   specialistArrayList.add(new Specialist(id: "9", title: "Gynecology"));
68   specialistArrayList.add(new Specialist(id: "10", title: "Otolaryngology"));
69   specialistArrayList.add(new Specialist(id: "11", title: "Physician"));
70   specialistArrayList.add(new Specialist(id: "12", title: "Plastic Surgery"));
71   specialistArrayList.add(new Specialist(id: "13", title: "Psychiatry"));
72   specialistArrayList.add(new Specialist(id: "14", title: "Neurology"));
73   specialistArrayList.add(new Specialist(id: "15", title: "Urology"));
74
75   mAdapter = new SpecialistAdapter(context: this, specialistArrayList);
76   binding.specialistRv.setLayoutManager(new GridLayoutManager(context: this, spanCount: 2));
77   binding.specialistRv.setAdapter(mAdapter);
78   mAdapter.setOnItemClickListener(this);
79 }
80
81
82 @Override
83 public void onListItemClick(Specialist specialist) {
84   Intent intent = new Intent(packageContext: MedicalProfessionalsActivity.this, ProfessionalsActivity.class);
85   intent.putExtra(name: "title", specialist.getTitle());
86   startActivity(intent);
87 }
88
89 //getting filter list
90 private ArrayList<Specialist> filter(ArrayList<Specialist> hi, String query) {
91   query = query.toLowerCase();
92   final ArrayList<Specialist> filterModelList = new ArrayList<>();
93   for (Specialist modal : hi) {
94
95     final String sTitle = modal.getTitle().toLowerCase();
96
97     if (sTitle.startsWith(query)) {
98       filterModelList.add(modal);
99     }
100   }
101   return filterModelList;
102 }
103
104 }
```

Figure 6.15

The figure above shows the codes for the `MedicalProfessionalsActivity`. It is the main activity that handles the viewing of medical professionals featured in the application. To set up the department or specialisation, `setUpSpecialist()` is implemented where they are hard-coded and stored into an array due to time constraints.

SpecialistAdapter.java

```
15  public class SpecialistAdapter extends RecyclerView.Adapter<SpecialistAdapter.ViewHolder> {
16      private Context context;
17      ArrayList<Specialist> specialistArrayList;
18      OnItemClickListener mListener;
19
20      public SpecialistAdapter(Context context, ArrayList<Specialist> specialistArrayList) {
21          this.context = context;
22          this.specialistArrayList = specialistArrayList;
23      }
24
25      public interface OnItemClickListener {
26          void onListItemClick(Specialist home);
27      }
28
29      public void setOnItemClickListener(OnItemClickListener listener) { mListener = listener; }
30
31      @NotNull
32      @Override
33      public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
34          CustomSpecialistLayoutBinding binding = CustomSpecialistLayoutBinding
35              .inflate(
36                  LayoutInflater.from(parent.getContext()),
37                  parent,
38                  attachToParent: false);
39
40          return new ViewHolder(binding);
41      }
42
43      @Override
44      public void onBindViewHolder(@NotNull final ViewHolder holder, int position) {
45          Specialist home = specialistArrayList.get(position);
46          holder.bind(home);
47
48          @Override
49          public int getItemCount() { return specialistArrayList.size(); }
50
51          public class ViewHolder extends RecyclerView.ViewHolder {
52              private final CustomSpecialistLayoutBinding binding;
53
54              @Override
55              public ViewHolder(CustomSpecialistLayoutBinding binding) {
56                  super(binding.getRoot());
57                  this.binding = binding;
58
59                  binding.linear.setOnClickListener(view -> {
60                      int position = getAdapterPosition();
61                      if (position != RecyclerView.NO_POSITION) {
62                          Specialist home = specialistArrayList.get(position);
63                          mListener.onListItemClick(home);
64                      }
65                  });
66              }
67
68              @Override
69              public void bind(Specialist home) {
70                  binding.title.setText(home.getTitle());
71              }
72
73          }
74
75          public void setFilter(ArrayList<Specialist> list) {
76              specialistArrayList = new ArrayList<>();
77              specialistArrayList.addAll(list);
78              notifyDataSetChanged();
79          }
80
81      }
82
83  }
```

Figure 6.16

The figure above shows the codes for the SpecialistAdapter. It is to display the specialists/doctors a recyclerview using a viewholder.

ProfessionalsActivity.java

```
38 public class ProfessionalsActivity extends AppCompatActivity implements UserAdapter.OnItemClickListener {
39     private ActivityProfessionalsBinding binding;
40     private FirebaseAuth mAuth;
41     private UserAdapter mAdapter;
42     private DatabaseReference userRef, appointmentRef;
43     private ArrayList<Users> doctorArrayList = new ArrayList<>();
44     final Calendar myCalendar = Calendar.getInstance();
45     private String specialization;
46     private String patientName = "";
47     @Override
48     protected void onCreate(Bundle savedInstanceState) {
49         super.onCreate(savedInstanceState);
50         binding = ActivityProfessionalsBinding.inflate(getLayoutInflater());
51         setContentView(binding.getRoot());
52
53         binding.backBtn.setOnClickListener(v -> onBackPressed());
54         //initialize
55         mAuth = FirebaseAuth.getInstance();
56         userRef = FirebaseDatabase.getInstance().getReference("users");
57         appointmentRef = FirebaseDatabase.getInstance().getReference("appointments");
58
59         Bundle bundle = getIntent().getExtras();
60         if (bundle != null) {
61             specialization = bundle.getString(key: "title");
62             binding.title.setText(specialization);
63             getDoctors(specialization);
64         }
65         getUserData();
66     }
67
68     private void getUserData() {
69         userRef.child(Objects.requireNonNull(mAuth.getUid()))
70             .addValueEventListener(new ValueEventListener() {
71                 @Override
72                 public void onDataChange(@NonNull DataSnapshot snapshot) {
73                     if (snapshot.exists()) {
74                         Users users = snapshot.getValue(Users.class);
75                         patientName = users.getName();
76                     }
77                 }
78                 @Override
79                 public void onCancelled(@NonNull DatabaseError error) {
80                     Snackbar snackbar = Snackbar
81                         .make(binding.getRoot(), error.getMessage(), Snackbar.LENGTH_LONG);
82                     snackbar.show();
83                 }
84             });
85     }
86
87     private void getDoctors(String specialization) {
88         doctorArrayList.clear();
89         binding.loading.setVisibility(View.VISIBLE);
90         userRef.orderByChild("specialization").equalTo(specialization).addValueEventListener(new ValueEventListener() {
91             @Override
92             public void onDataChange(@NonNull DataSnapshot snapshot) {
93                 if (snapshot.exists()) {
94                     binding.loading.setVisibility(View.GONE);
95
96                     for (DataSnapshot ds : snapshot.getChildren()) {
97                         Users users = ds.getValue(Users.class);
98                         doctorArrayList.add(users);
99                         Log.e(tag: "doctorrrr", doctorArrayList.toString());
100                }
101            }
102        });
103    }
104}
```

```
102         mAdapter = new UserAdapter(context: ProfessionalsActivity.this, doctorArrayList);
103         binding.doctorsRv.setLayoutManager(new LinearLayoutManager(context: ProfessionalsActivity.this));
104         binding.doctorsRv.setAdapter(mAdapter);
105         mAdapter.setOnItemTouchListener(ProfessionalsActivity.this);
106
107     } else {
108         binding.loading.setVisibility(View.GONE);
109     }
110 }
111
112 @Override
113 void onCancelled(@NonNull DatabaseError error) {
114     binding.loading.setVisibility(View.GONE);
115
116     Snackbar snackbar = Snackbar
117         .make(binding.getRoot(), error.getMessage(), Snackbar.LENGTH_LONG);
118     snackbar.show();
119 }
120 });
121 }
122
123 @Override
124 void onListItemClick(Users users) {
125     final Dialog dialog = new Dialog(context: this);
126     dialog.setContentView(R.layout.custom_dialog_layout);
127
128     TextInputEditText selectDate = dialog.findViewById(R.id.booking_date_et);
129     TextInputEditText selectTime = dialog.findViewById(R.id.booking_time_et);
130     CardView bookingBtn = dialog.findViewById(R.id.book_card_view);
131     CardView cancelBtn = dialog.findViewById(R.id.cancel_card_view);
132     cancelBtn.setOnClickListener(v -> dialog.dismiss());
133
134     DatePickerDialog.OnDateSetListener date = (view, year, month, day) -> {
135         myCalendar.set(Calendar.YEAR, year);
136         myCalendar.set(Calendar.MONTH, month);
137         myCalendar.set(Calendar.DAY_OF_MONTH, day);
138         String myFormat = "dd/MM/yyyy";
139         SimpleDateFormat dateFormat = new SimpleDateFormat(myFormat, Locale.US);
140         selectDate.setText(dateFormat.format(myCalendar.getTime()));
141     };
142     selectDate.setOnClickListener(v -> new DatePickerDialog(context: ProfessionalsActivity.this, date, myCalendar.get(Calendar.YEAR),
143             myCalendar.get(Calendar.MONTH), myCalendar.get(Calendar.DAY_OF_MONTH)).show());
144
145     selectTime.setOnClickListener(v -> {
146         Calendar mCurrentTime = Calendar.getInstance();
147         int hour = mCurrentTime.get(Calendar.HOUR_OF_DAY);
148         int minute = mCurrentTime.get(Calendar.MINUTE);
149         TimePickerDialog mTimePicker;
150         mTimePicker = new TimePickerDialog(context: ProfessionalsActivity.this, (timePicker, selectedHour, selectedMinute) ->
151             selectTime.setText(selectedHour + ":" + selectedMinute), hour, minute, true); //Yes 24 hour time
152         mTimePicker.setTitle("Select Time");
153         mTimePicker.show();
154     });
155
156     bookingBtn.setOnClickListener(v -> {
157         if (selectDate.getText().toString().isEmpty()) {
158             selectDate.setError("Please select date");
159             return;
160         }
161         if (selectTime.getText().toString().isEmpty()) {
162             selectTime.setError("Please select time");
163             return;
164         }
165     });
166 }
```

```

165         bookAppointment(dialog, selectDate.getText().toString(), selectTime.getText().toString(),
166                         users.getUserId(), users.getName());
167     });
168     dialog.show();
169 }
170
171 private void bookAppointment(Dialog dialog, String selectedData, String selectedTime, String doctorId, String doctorName) {
172     String appointmentId = appointmentRef.push().getKey();
173     Map<String, Object> appointment = new HashMap<>();
174     appointment.put("appointmentId", appointmentId);
175     appointment.put("doctorId", doctorId);
176     appointment.put("userId", mAuth.getUid());
177     appointment.put("date", selectedData);
178     appointment.put("time", selectedTime);
179     appointment.put("meetingLink", "");
180     appointment.put("doctorName", doctorName);
181     appointment.put("patientName", patientName);
182     appointment.put("status", "pending");
183     appointment.put("specialization", specialization);
184     appointment.put("prescription", "");

185     appointmentRef.child(appointmentId)
186         .setValue(appointment)
187         .addOnCompleteListener(task -> {
188             if (task.isSuccessful()){
189                 dialog.dismiss();
190                 Intent intent = new Intent( packageContext: ProfessionalsActivity.this, ConfirmBookingActivity.class);
191                 intent.putExtra( name: "doctorName", doctorName);
192                 startActivity(intent);
193             }
194         })
195     }.addOnFailureListener(e -> Toast.makeText( context: ProfessionalsActivity.this, text: ""+ e.toString(), Toast.LENGTH_SHORT).show());

```

Figure 6.17

The figure above shows the codes for the `ProfessionalsActivity`. It is the main activity for users to book an appointment with the specialist/medical professional. When the user clicks on the book button in line124, a dialog will show up for the users to enter appointment details such as date and time. `bookAppointment()` function is implemented to save the data into the database under reference of `appointmentRef`. Users are then redirected to the confirmation page.

ConfirmBookingActivity.java

```

11 public class ConfirmBookingActivity extends AppCompatActivity {
12     private ActivityConfirmBookingBinding binding;
13
14     @Override
15     protected void onCreate(Bundle savedInstanceState) {
16         super.onCreate(savedInstanceState);
17         binding = ActivityConfirmBookingBinding.inflate(LayoutInflater);
18         setContentView(binding.getRoot());
19
20         Bundle bundle = getIntent().getExtras();
21         if (bundle != null){
22             binding.des.setText("Your booking with Dr " + bundle.getString(key: "doctorName") + " is successful!");
23         }
24
25         binding.continueTv.setOnClickListener(v -> {
26             Intent intent = new Intent( packageContext: ConfirmBookingActivity.this, UserHomeActivity.class);
27             startActivity(intent);
28             finishAffinity();
29         });
30
31     }
32 }

```

Figure 6.18

The figure above shows the codes for the ConfirmBookingActivity. It is when users are done with the booking and a dialog will show up to indicate the booking is successful and will be redirected to the home page.

AppointmentActivity.java

```

24  public class AppointmentActivity extends AppCompatActivity implements AppointmentAdapter.OnItemClickListener {
25      private ActivityAppointmentBinding binding;
26      private FirebaseAuth mAuth;
27      private DatabaseReference userRef, appointmentRef;
28      private ArrayList<Appointment> appointmentArrayList = new ArrayList<>();
29      private AppointmentAdapter mAdapter;
30      @Override
31      protected void onCreate(Bundle savedInstanceState) {
32          super.onCreate(savedInstanceState);
33          binding = ActivityAppointmentBinding.inflate(getLayoutInflater());
34          setContentView(binding.getRoot());
35
36          binding.backBtn.setOnClickListener(v -> onBackPressed());
37
38          //initialize
39          mAuth = FirebaseAuth.getInstance();
40          userRef = FirebaseDatabase.getInstance().getReference("path: \"users\"");
41          appointmentRef = FirebaseDatabase.getInstance().getReference("path: \"appointments\"");
42
43          getAppointments();
44      }
45      private void getAppointments() {
46          appointmentArrayList.clear();
47          binding.loading.setVisibility(View.VISIBLE);
48          appointmentRef.orderByChild("userId").equalTo(mAuth.getUid())
49              .addValueEventListener(new ValueEventListener() {
50                  @Override
51                  public void onDataChange(@NonNull DataSnapshot snapshot) {
52                      if (snapshot.exists()) {
53                          binding.loading.setVisibility(View.GONE);
54
55                          for (DataSnapshot ds: snapshot.getChildren()){
56                              Appointment appointment = ds.getValue(Appointment.class);
57                              appointmentArrayList.add(appointment);
58                          }
59                          mAdapter = new AppointmentAdapter(context: AppointmentActivity.this, appointmentArrayList);
60                          binding.appointmentRv.setLayoutManager(new LinearLayoutManager(context: AppointmentActivity.this));
61                          binding.appointmentRv.setAdapter(mAdapter);
62                          mAdapter.setOnItemClickListener(AppointmentActivity.this);
63                      } else {
64                          binding.loading.setVisibility(View.GONE);
65                      }
66
67                      @Override
68                      public void onCancelled(@NonNull DatabaseError error) {
69                      }
70                  });
71
72
73          public void onAcceptClick(Appointment home) {
74          }
75
76          public void onRejectClick(Appointment home) {
77          }
78          public void onMeetingLinkClick(Appointment home) {
79              String data = home.getMeetingLink();
80              Intent defaultBrowser = Intent.makeMainSelectorActivity(Intent.ACTION_MAIN, Intent.CATEGORY_APP_BROWSER);
81              defaultBrowser.setData(Uri.parse(data));
82              startActivity(defaultBrowser);
83          }
84
85          public void onPrescriptionClick(Appointment home) {
86      }

```

Figure 6.19

The figure above shows the codes for the AppointmentActivity. It is the main activity where it handles the appointment feature and gets appointments according to the user id.

AppointmentAdapter.java

```
16
17  public class AppointmentAdapter extends RecyclerView.Adapter<AppointmentAdapter.ViewHolder> {
18      private Context context;
19      ArrayList<Appointment> appointmentArrayList;
20      OnItemClickListener mListener;
21      private FirebaseAuth mAuth = FirebaseAuth.getInstance();
22
23      public AppointmentAdapter(Context context, ArrayList<Appointment> appointmentArrayList) {
24
25          this.context = context;
26          this.appointmentArrayList = appointmentArrayList;
27      }
28
29      public interface OnItemClickListener {
30          void onAcceptClick(Appointment home);
31          void onRejectClick(Appointment home);
32          void onMeetingLinkClick(Appointment home);
33          void onPrescriptionClick(Appointment home);
34      }
35
36      public void setOnItemClickListener(OnItemClickListener listener) { mListener = listener; }
37
38
39
40      @NonNull
41      @Override
42      public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
43          CustomAppointmentLayoutBinding binding = CustomAppointmentLayoutBinding
44              .inflate(
45                  LayoutInflater.from(parent.getContext()),
46                  parent,
47                  attachToParent: false);
48
49
50
51      @Override
52      public void onBindViewHolder(@NonNull final ViewHolder holder, int position) {
53          Appointment home = appointmentArrayList.get(position);
54          holder.bind(home);
55      }
56
57
58      @Override
59      public int getItemCount() { return appointmentArrayList.size(); }
60
61
62      public class ViewHolder extends RecyclerView.ViewHolder {
63          private final CustomAppointmentLayoutBinding binding;
64
65          public ViewHolder(CustomAppointmentLayoutBinding binding) {
66              super(binding.getRoot());
67              this.binding = binding;
68
69
70              binding.acceptBtn.setOnClickListener(view -> {
71                  int position = getAdapterPosition();
72                  if (position != RecyclerView.NO_POSITION) {
73                      Appointment home = appointmentArrayList.get(position);
74                      mListener.onAcceptClick(home);
75                  }
76              });
77
78              binding.rejectBtn.setOnClickListener(view -> {
79                  int position = getAdapterPosition();
80                  if (position != RecyclerView.NO_POSITION) {
81                      Appointment home = appointmentArrayList.get(position);
82                      mListener.onRejectClick(home);
83                  }
84              });
85          }
86
87      }
88
89
90
91
92
93
94
95
96
97
98
99
```

```
78     binding.rejectBtn.setOnClickListener(view -> {
79         int position = getAdapterPosition();
80         if (position != RecyclerView.NO_POSITION) {
81             Appointment home = appointmentArrayList.get(position);
82             mListener.onRejectClick(home);
83         }
84     });
85     binding.meetingLink.setOnClickListener(view -> {
86         int position = getAdapterPosition();
87         if (position != RecyclerView.NO_POSITION) {
88             Appointment home = appointmentArrayList.get(position);
89             mListener.onMeetingLinkClick(home);
90         }
91     });
92     binding.addPrescription.setOnClickListener(view -> {
93         int position = getAdapterPosition();
94         if (position != RecyclerView.NO_POSITION) {
95             Appointment home = appointmentArrayList.get(position);
96             mListener.onPrescriptionClick(home);
97         }
98     });
99 }
100 }
101 @
102 public void bind(Appointment home) {
103     binding.patientName.setText(home.getPatientName());
104     binding.doctorName.setText(home.getDoctorName());
105     binding.bookDate.setText(home.getDate());
106     binding.bookTime.setText(home.getTime());
107     binding.bookSpecialization.setText(home.getSpecialization());
108     if (!home.getMeetingLink().equals("")){
109         binding.meetingLink.setText(home.getMeetingLink());
110     }
111 }
```

```
111     if (!home.getPrescription().equals("")){
112         binding.addPrescription.setText(home.getPrescription());
113     }
114     if (home.getStatus().equals("accepted")){
115         binding.acceptBtn.setText("Accepted");
116         binding.acceptBtn.setEnabled(false);
117         binding.rejectBtn.setVisibility(View.GONE);
118     }else {
119         binding.acceptBtn.setEnabled(true);
120         binding.rejectBtn.setVisibility(View.VISIBLE);
121     }
122     if (home.getStatus().equals("rejected")){
123         binding.rejectBtn.setText("Rejected");
124         binding.rejectBtn.setEnabled(false);
125         binding.acceptBtn.setVisibility(View.GONE);
126     }else {
127         binding.rejectBtn.setEnabled(true);
128         binding.acceptBtn.setVisibility(View.VISIBLE);
129     }
130
131     if (home.getUserId().equals(mAuth.getUid())){
132         binding.rejectBtn.setVisibility(View.GONE);
133         if (home.getStatus().equals("pending")){
134             binding.rejectBtn.setVisibility(View.GONE);
135             binding.acceptBtn.setVisibility(View.VISIBLE);
136             binding.acceptBtn.setEnabled(false);
137             binding.acceptBtn.setText("Pending");
138         }
139
140         if (binding.meetingLink.getText().toString().equals("Add Link")){
141             binding.meetingLink.setText("No meeting available");
142             binding.meetingLink.setEnabled(false);
143         }
144         if (binding.addPrescription.getText().toString().equals("Add Prescription")){
145             binding.addPrescription.setText("No Prescription Available");
146             binding.addPrescription.setEnabled(false);
147         }
148     }
149 }
150 }
```

Figure 6.20

The figure above shows the codes for the AppointmentAdapter. The adapter displays the appointments into a recyclerview using a viewholder. It allows doctors to accept or reject the appointments. In the appointment card, details such as patient name, doctor name is fetched from the database by accessing the respective Class.

5.0 Conclusion

To conclude the project and the document, the team has managed to design and implement a mobile application that fulfils the objective of building a system that offers functionalities such as ability to schedule/cancel appointments, the ability to request prescriptions and easy access to medical records. Hopefully with these features, the application would appeal to many people who want to use an application with convenient features like this and vastly improve the field of mobile applications by adhering to these principles of taking note of what people would want in an application. By using the waterfall methodology, we have managed to finish the project in time while making it functional. Some improvements that could be made to the application would be a native video function because currently the video function links to an external site which some users may find cumbersome to use. Besides that, due to time constraints, department/specialisation can be added by the admin instead of hard coding them.

Mobile Programming & Screen Design 2 Documentation - Medicare

ORIGINALITY REPORT



PRIMARY SOURCES

1	Submitted to KDU College Sdn Bhd Student Paper	2%
2	www.imaginovation.net Internet Source	1%
3	developer.android.com Internet Source	1%
4	kinsta.com Internet Source	1%
5	Kevin Languedoc. "Build iOS Database Apps with Swift and SQLite", Springer Science and Business Media LLC, 2016 Publication	1%
6	docsdrive.com Internet Source	1%
7	Submitted to Roehampton University Student Paper	<1%
8	Submitted to Canadian University of Dubai Student Paper	<1%

9	en.wikibooks.org Internet Source	<1 %
10	Submitted to University of Salford Student Paper	<1 %
11	neurologyindia.com Internet Source	<1 %
12	Submitted to City University of Hong Kong Student Paper	<1 %
13	www.nwdc.navy.mil Internet Source	<1 %

Exclude quotes Off
Exclude bibliography Off

Exclude matches Off