SUJAY HAZRA

1NH17CS127

TOPIC: Jam Secure - Jam Room Booking App with chat bot

CHAPTER 1

INTRODUCTION

The project here is referred to an Android application called JamSecure developed using Android Studios, with the help of firebase services (Realtime Database & Authentication) for the backend and the IBM Watson Assistant Service for developing the chat bot. The goal of this app is to provide a medium for people in the music industry to facilitate the booking of jam rooms with ease, The Jam rooms are nothing but Studio Rooms owned by different individuals all across the city that rent out a so-called 'Jam Room' on an hourly basis and provide sound systems and various equipments for music groups to practice/play music. The app supports multi user interface, depending on the category of user ("Users", "Owners"), the app launches respective dashboards and functionalities, which allows users to book jam rooms, and owners to view the list of users that have booked their jam rooms and their slots.

1.1 ANDROID

Android is a mobile operating system which is a Linux based operating system it is designed primarily for touch screens mobile devices such as smartphones and tablet computers. The operating system has been developed a lot in the last 15 years which had actually started from black and white phones to recent smartphones or mini computers. android is one of the most widely used mobile OS these days by the people. The android, it is a software that was found in Palo Alto of California in 2003.

Android is a powerful operating system. Also, it supports a large number of applications in Smartphones. These applications are more comfortable. Also advanced for users. The hardware that supports android software, it is based on the ARM architecture platform. The android is an open-source operating system which actually means that it's free and anyone can use it. The android has got millions of apps available in the system, that can help you manage your life one or another way and it is available to low cost in the market. Hence, for that reason android is very popular.

Android development is fully supported by java programming language. Even other packages that are API and JSE. These are not supported. The first version 1.0 which is of the android development kit (SDK) was released in 2008 and jelly bean is the latest updated version is a jelly bean.

The versions of android are as follows:

· Android 1.0: Android Alpha

· Android 1.1: Android Beta

· Android 1.5: Android Cupcake

· Android 1.6: Android Donut

· Android 2.0 - 2.1: Android Éclair

· Android 2.2 – 2.2.3: Android Froyo

· Android 2.3.3 – 2.3.7: Android Gingerbread

· Android 3.0 – 3.2.6: Android Honeycomb

· Android 4.0 – 4.0.4: Android Ice Cream Sandwich

· Android 4.1 – 4.3.1: Android Jelly bean

· Android 4.4 – 4.4.4: Android KitKat

· Android 5.0 – 5.1.1: Lollipop

· Android 6.0 – 6.0.1: Android Marshmallow

· Android 7.0 – 7.1.2: Android Nougat

· Android 8.0 – 8.1.0: Android Oreo

· Android 9.0: Android Pie

· Android 10: Android Q

· Android 11: Android R

1.2 ANDROID ARCHITECTURE

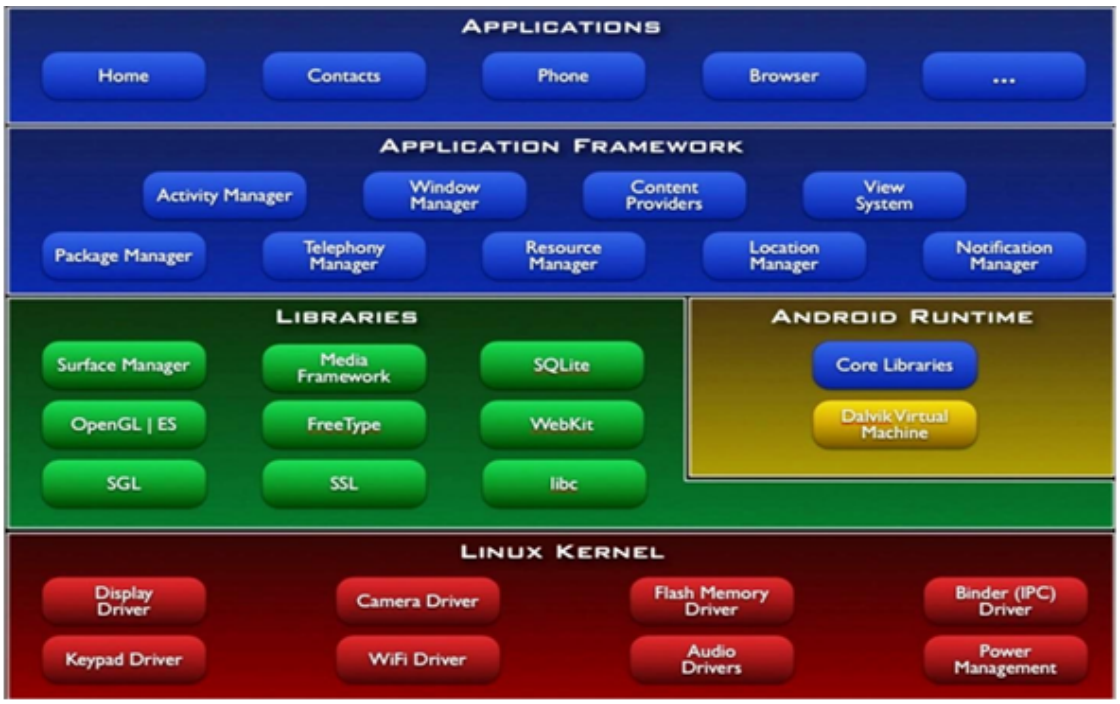
Android architecture contains different number of components to support any android device needs. Android is a software that contains an open-source Linux Kernel which is having collection of number of C/C++ libraries which are exposed through an application framework services.

Among all the components Linux Kernel provides main functionality of operating system functions to smartphones and Dalvik Virtual Machine (DVM) provide platform for running an android application.

The main components of android architecture are following: -

- Applications
- Application Framework
- Android Runtime
- Platform Libraries
- Linux Kernel

Android operating system, it is a stack of software commponents which is roughly divided into five sections and four main layers as shown below in the archiitecture diagram.



1.2.2 Android Architecture

Linux kernel

Linux is at the bottom of the layers - Linux 3.6 with approx.. 115 patches. This provides a level of abstractionn between the device hardware and it contains all the essential hardware drivers like camera, keypad, display etc. Also, the kernel hanndles all the things that Linux is really good at such as nettworking and a vast array of device drivers, which take the pain out of interfacing to peripheral hardware.

Libraries

On top of Linax kernel therre is a set of libraries including open-source Web browsre engine WebKit, well known library libc, SQLite datebase which is a useful repository for storage and sharing of application data, libraeries to play and record audio and video, SSL libraries responsible for Internet security etc.

Android Libraries

This category encoompasses those Java-based libraries that are speciific to Android development. Exaamples of libraries in this category include the application framework liibraries in addition to those that faacilitate user interface building, graphics drawiing and database access. A summary of some key core Android libraries available to the Android developer is as folllows −

- **android.app** − Provides acceess to the application model and is the cornerstone of all Android applications.
- **android.content** − Facilitates content acceess, publishing and messaging between applications and application components.
- **android.database** − Used to acceess data published by content providers and includes SQLite database management classes.
- **android.opengl** − A Java inteerface to the OpenGL ES 3D graphics rendering API.
- **android.os** − Proviides applications with access to standard operating system services including messages, system services and inter-process communication.
- **android.text** − Used to rendrer and manipulate text on a device display.
- **android.view** − The fundamental builiding blocks of application user interfaces.
- **android.widget** − A rich collection of pere-built user interface components such as buttons, labels, list views, layout managers, radio buttons etc.
- **android.webkit** − A set of classses intended to allow web-browsing capabilities to be built into applications.

Having covrered the Java-bassed core libraaries in the Android runtime, it is now time to turn our attention to the C/C++ based libraries contained in this layer of the Android software stack.

Android Runtime

This is the thiird section of the architecture and available on the second layer from the bottom. This section priovides a key component called **Dalvik Virtual Machine** which is a kind of Java Virtual Machine speicially designed and optimized for Android.

The Dalvik VM makss use of Linux core features like memory management and multi-threading, which is intriinsic in the Java language. The Dalvik VM enables every Android application to run in its own process, with its own instanece of the Dalvik virtual machine.

The Android runntime also proviides a set of core libraries which enlable Android application developers to write Androiid applications using standard Java programming language.

Application Framework

The Application Frameweork layer provides many higher-level services to applications in the form of Java classes. Appliication developers are allowed to make use of these services in their applications.

The Android framework includes the folllowing key services −

- **Activity Manager** − Contrlols all aspects of the application lifecycle and activity stack.
- **Content Providers** − Allosws applications to publish and share data with other applications.
- **Resource Manager** − Provides acceess to non-code embedded resources such as strings, color settings and user interface layouts.
- **Notifications Manager** − Allows appliications to display alerts and notifications to the user.
- **View System** − An extensible set of vieiws used to create application user interfaces.

Applications

You will find all the Androiid application at the top layer. You will write your application to be installled on this layer only. Exammples of such applications are Contacts Books, Browser, Games etc.

1.3 ABOUT MINI PROJECT

In the field of live professional music events, It is necessary for bands to come together and practice(Jam) in a place with musical equipment and sound systems. There are studio Jam Rooms owned by different individuals all across the cities that rent out a so-called 'Jam Room' on an hourly basis and provide sound systems and various equipment. As a part of few professional bands, I have booked and rented out many of these jam rooms throughout the years. A common problem faced during this process is that there is no platform for booking, everything is done on personal calls, etc. This way bookings are usually mismanaged, to name a few- The Jam room is sometimes scheduled to multiple bands at the same time, which becomes hard to compromise because of upcoming gigs or music events, Hard to locate the exact location of these Jam rooms, hard to keep track of all the bands jamming in the studio and manage their respective slots.

1.4 OBJECTIVE OF MINI PROJECT

As I have faced this problem in real time, I have come up with the Solution of developing an Android App with database connectivity that works as a platform for musical artists and studio owners to interact with each other and schedule slots, the application will have two kinds of users:-

1) Jam Room Owners owning the Jam Room - provide a GUI to check out all the slots booked so far, list with the name and basic info of the person who booked the slot and the time/date of the slot booked, cancel bookings, etc.

2) Musicians/Bands wanting a Jam Room - provide a GUI to search for jam rooms with slots available for that time/day, see and compare Jam rooms with their locations and pricings, contact info of the Jam room owner, booking facilities online.

1.5 ADVANTAGES OF MINI PROJECT

The advantages of the mini project for the person developing:-

A thorough understanding of the Android Studios IDE, further understanding of Java concepts, designing UI's, managing activities.

Strengthen the database management knowledge, understand various services offered by the firebase cloud services, learn to manage data in the form of JSON tree, in a key-value pair relationship, with parent and child nodes.

Understand the IBM cloud services and deeper knowledge on using the IBM Watson Assistant Service to learn how to develop a chat bot.

Develop Entrepreneurial skills, as this is also a startup idea.

The advantages of the mini project for the app users:-

A medium for people in the music industry to facilitate the booking of jam rooms with ease.

CHAPTER 2

REQUIREMENT SPECIFICATION

2.1 MOBILE REQUIREMENT

- ➔ Android Version 6.0(Marshmallow) and above
- ➔ Min 4GB Ram
- ➔ 4GB Storage Space Availability
- ➔ Internet support (3G/4G)

2.2 COMPUTER REQUIREMENT

Hardware Requirement

- ➔ Microsoft® Windows® 7/8/10 (64-bit),
- ➔ 4 GB minimum RAM, 8 GB RAM recommended,
- ➔ 2 GB of available disk space minimum, 4 GB Recommended,
- ➔ (500 MB for IDE + 1.5 GB for Android SDK and emulator system image),
- ➔ 1080 x 800 minimum screen resolution.

Software Requirement

- ➔ Android Studios,
- ➔ Firebase Library,
- ➔ IBM Cloud Account(Watson Assistant) .

CHAPTER 3

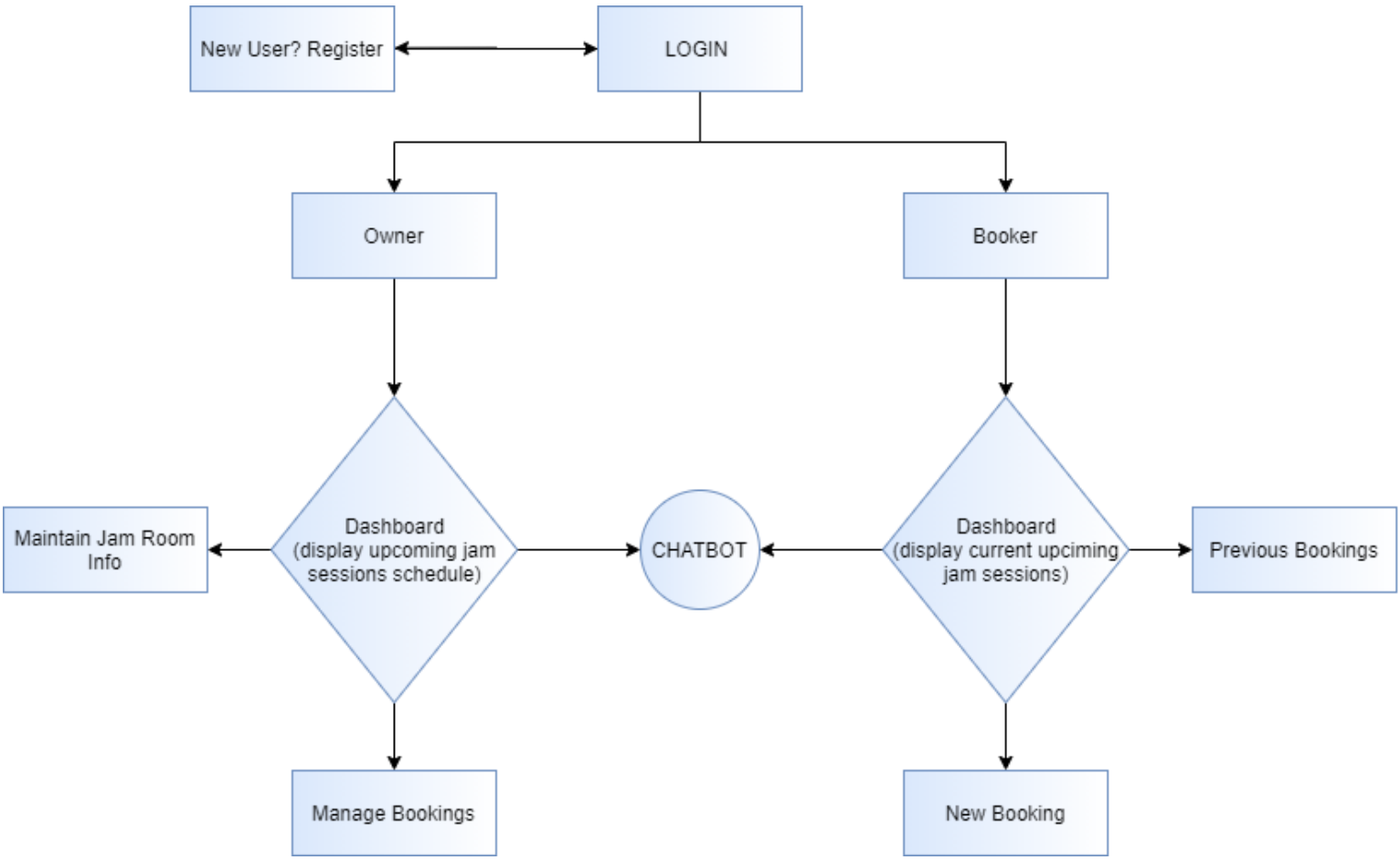
ANALYSIS AND DESIGN

3.1 ALGORITHM

- **Step 1:** A Login Screen Appears, The user is asked for his credentials (username and password). A “New User? Register now” prompt is present and when it is accessed, The user is navigated to an activity which asks the user to select a category to which the person belongs to, the user must now select if he is using the app to book jam rooms or he is a owner and looking for other people to book his jam rooms.
- **Step 2:** After selecting the category the person is navigated to the respective register activity where he/she fills in the details and credentials. Once that’s done, he will be taken to the respective dashboard.
- **Step 3:** Upon successful log-in, the user depending on his type (Owner/Booker) will be navigated to their respective dash boards.
- **Step 4:** The Booker dashboard consists of an activity that displays the current jam rooms booked by the user, and a floating button with 3 menu options - new booking, past booking & help bot.
- **Step 5:** New booking floating button will navigate the booker to a new activity which asks for mandatory details required for booking such as date, time, duration and location, after filling the details the user searches for the feasible jam room that matches the required criteria and proceeds to book.
- **Step 6:** The owner dashboard consists of an activity which displays the schedule for today’s bookings, and a chatbot button is present to guide the user.

- **Step 7:** Past bookings will pop up a dialog that displays the past bookings of that particular user.

3.2 FLOW CHART



CHAPTER 4

IMPLEMENTATION

4.1 Android concepts used

Apps and APK Files

An android app is an android appliication. An app is packaged in an APK file (Android application package). The APK file contains the compiled Java code and other reesources like texts and images for the Android application.

Widgets

Android widgets are GUI componeents which can be displayed outside of an activity. For instance, a weather widget showing todays weather is shown on manny Android home screens. Widgets are implemented and paackaged as part of an Android application.
Sometimes Views in Android are alsso referred to as "widgets". For inistance, many of the GUI components (View subclasses) are located in a Java package called android.widget. But GUI components are nott the same as a widget which can live on the home screen of an Android device. So, be careful whenn you read about Android to make a distinction between GUI components which can be used inside ViewGroups, Fragments and Activitiies (and also inside Widgets), and Widgets which can live on the homee screen of the Android device.

Android UI Controls:

Android provides a number of standard UII controls that enable a rich user experience. Desiigners and developers should thoroughly understand all of these controls for the followiing reasons:

· They are faster to impleiment. It can take up to ten times longer to develop a custom contrrol than to implement a user initerface with standard Android controls.

· They ensure good perforrmance. Custom controls rarely function as expected in their first implementation. By implemeinting standard controls, you can eliminate the need to test, revise and improve custom controls. Moreiover, while designers will spend a great deal of time thinking about how a control should look, theey may not always consider the many ways in which a custom control will behave in the usesr's hands. Items on a mobile device often need to grow and shrink in size as they are pincheed, or scroll if they are part of a list. Ass a result, creating a "clean" custom control from scratch can take a significant amount of design and development time. Google, however, has already thought about theese interactions and developed standard controls to properly address them.

· Android users expect standard controls. Through thieir interactions with other Android apps, users become accustomed to Android's standard controls. Deviiating from the standard Android user experiience can confuse and frustrate users, making them less likely to want to use your app and incorporate it into their dailiy activities.

Activity

Android applications are composedd of "activities" which are unique, focused actions a user can take. Because it can be difficult or time-consumiing to scroll, zoom in, or click links on a small screen, it is recommended that an app display only one activity per screen. This practice presents the user with only the most relevant information and alllows them to launch a new screen for additional information, or click the "back" button to view the previous activity. While a screen can exppose multiple tasks, it should help the user complete just one activity at a tiime. In Gmail for example, a user can only read the body of an e-miail (right) once he has clicked the relevant message (left). Tihis layout reduces the amount of information displayed on each screen and allows the user to easily naviigate between the Inbox and the message text.

User Interactions

When a useer first downloads your application, he will make snap judgments on the usability and intuitiveness of the appliication within the first few minutes of use. It is, therefore, crucial to balance the creativity of your app wiith the standard user interactions Androiid users have come to expect. These include:

· Hard buttons: inciluding Back, Menu, Home and Search buttons. Soft buttons that dupilicate these features will only confuse or frustrate Android users. Moreover, back ibutton behavior can be tricky andi needs to be defined up-front for every screen, as it is not always as simple as returning to the previous activities. Most mobiile phones, for example, offer both an “incoming call” activity and an “active call” activity. Once a user has answeredi and completed the call, the user would not expeiict to return to the “incoming call” activity upon pressing the “back” button, but rather to the activity that occurred before the incoming caill. If the app offers only one activity, the back buttoni should return the user to the device’s home page.

· Long press eilements: Items of a list can be long pressed to open a context menu that provides secondary information. “ToDo” liist apps, for example, oiften use a touch interaction to mark a task as completedi and a long press interaction to display a menu with “edit” or “delete” functionality.

Layouts

Android UI screens are frequently resized, both on the fly via piinch and zoom as well as at startup when Android adjusts the siize of the UI to fit the screen size of the mobile device on which it’s running. In order to make the most of the screen size and handle this reisizing gracefully, Anidroid provides a number of screen layout options. First, Androiid developers must specify whether eaich screen should follow a linear layout which manages controls in a horizontal or vertical fashiion or a relative layout which manages controls iin relation to one another. Liinear layouts are the most common, as in the example below. At left, the controls only stretch to accommodate the text and are positioned in a horizontal linie. In the miiddle

image, the same rules apply but in a vertical layout. At right, the vertical layout is maintaiined but the miiddle button stretches to accommodate the screen rather than the text. A reilative layout defines the position of controls by theiir relationship to other components on the same screen.

Androiid also offers speciific layout properties to control the way in which screen elements are diisplayed across Android devices and during use:

· Weight: The weiight property allows the developer to determine how free space is divided on the screen.

· Gravity: Gravity iis the term used for control alignment (right, bottom, top, or left) on an Android device.

. · Density independence: Your appliication achieves “densiity independence” when it preserves the physical size (from the user’s point of view) of user iinterface elements displayed on screens with different densities. Wiithout density independence, a UI element (such as a button) will appear larger on a low-denisity screen and smaller on a high-denisity screen.

Intents

Android applications typicallly borrow from other applications already on the device. Using intents you can siimplify both the programming requirements for your app and offer simpler, less cluttered screens. If your app needs to perform a functiion beyond its core abilities such as opening a photo, looking up a contact, or playing a video, the team should investigate whether a tooll that can perform that function already exiists in the OS or in a popular third-party app. If so, you can leverage that functionaility using intents. For example, if your app accesses usser contacts, you can use iintent objects to launch the device’s existing Contacts application. This will elimiinate programming duplication and speed up the user’s interaction with the deviice since the user will not need to re-learn how to add a contact to your partiicular app.

Android offers specific UI controls, activiiities, interactions, layout and resize options, as well as special constructs likie fragments and intents. While on the surface these appear to be thiings that the design team needs to work with, we contend that the entire team must be immersed in Android to cooirdinate design, workflow, and execution into a single, intuitive application — one that grabs users' attention and draws them into the real value of your product.

Services

Android servicies are background processes that can be executed on an Android deviice, even if no applicaiition is visible. Services do not need a user intierface. A service could for inistance check a remotee server for updates, or backup data eveery hour etc.

4.2 Functionality of the project

The functionality of the project is explained by explaining each of the activities in the project:-

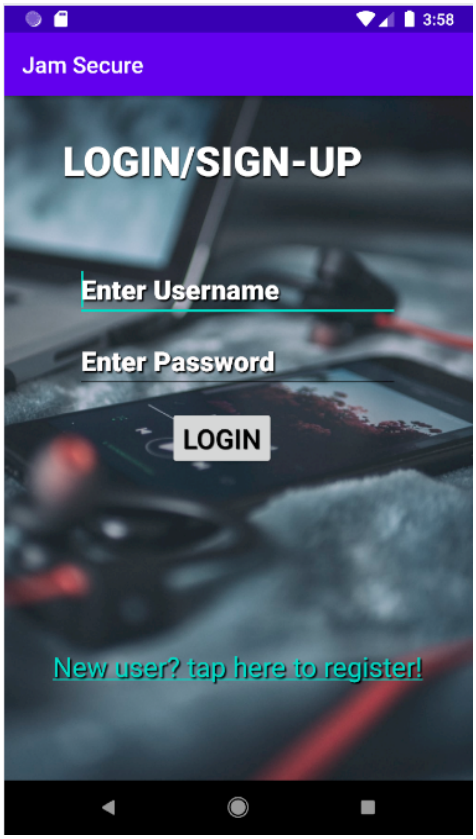
Login authentication

login is basically for the user to login with their existing account. Users are asked to provide their registered email id and password. Upon successful authentication the user depending on their category is navigated to their respective dashboard using Intents. Firebase Authentication library is used here.

```
mAuth.signInWithEmailAndPassword(email, password).addOnCompleteListener((task) → {
        if(task.isSuccessful()){
            progressBar.setVisibility(View.GONE);
            Toast.makeText( context: MainActivity.this, text: "Login Successful",Toast.LENGTH_LONG).show();
            mUser = mAuth.getCurrentUser();
            mDatabase = FirebaseDatabase.getInstance();
            mDatabase.getReference( path: "Owners").addListenerForSingleValueEvent(new ValueEventListener() {
                @Override
                public void onDataChange(@NonNull DataSnapshot snapshot) {
                    Map<String,Object> owners = (Map)snapshot.getValue();
                    if(owners.containsKey(mUser.getUid())){
                        Intent i = new Intent( packageContext: MainActivity.this,RoomOwner.class);
                        Log.d( tag: "nun", msg: "1");
                        i.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP); //so that you don't come back to login page again
                        startActivity(i);
                        //Owner code
                    }
                    else{
                        Intent i = new Intent( packageContext: MainActivity.this,Bookjamroom.class);
                        i.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                        //so that you don't come back to login page again
                        Log.d( tag: "nun", msg: "0");
                        startActivity(i);
                    }
                }

                @Override
                public void onCancelled(@NonNull DatabaseError error) {

                }
            });
        }
        else{
            progressBar.setVisibility(View.GONE);
            Toast.makeText(getApplicationContext(),task.getException().getMessage(), Toast.LENGTH_SHORT).show();
        }
    });
}
```

Figure 4.2.1 Login Code Snippet

Figure 4.2.2 Login Activity Screenshot:



Registration

If the user is new to the app, the user has a “New user? tap here to register!” hyperlinked text that will take them to the activity where they are asked to select from a radio button of two choices which specifies the category of “User” or “Owner” . Depending on this selection the user is navigated to the respective registration activities, where the user provides details such as name, email, password and other additional details depending on the user type.

```
if(pos==0){
    Toast toast = Toast.makeText( context: this,  text: "Please Select Location, not this one", Toast.LENGTH_LONG);
    toast.show();
    return;
}
progressBar.setVisibility(View.VISIBLE);
mAuth.createUserWithEmailAndPassword(em,password).addOnCompleteListener((task) → {

        if(task.isSuccessful()){
            User user = new User(name,em,phone,location);
            FirebaseDatabase.getInstance().getReference( path: "Users")
                    .child(FirebaseAuth.getInstance().getCurrentUser().getUid())
                    .setValue(user).addOnCompleteListener((task) → {
                    progressBar.setVisibility(View.GONE);
                   if(task.isSuccessful()){
                       Toast.makeText( context: Register.this, text: "Registration Successful" ,Toast.LENGTH_LONG).show();
                       progressBar.setVisibility(View.GONE);
                       Intent i = new Intent( packageContext: Register.this,Bookjamroom.class);
                       i.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP); //so that you don't come back to login page again
                       startActivity(i);
                    }
                   else{
                       Toast.makeText( context: Register.this, text: "Unable To Register",Toast.LENGTH_LONG).show();
                       progressBar.setVisibility(View.GONE);
                   }
            });
        }
        else{
            if(task.getException() instanceof FirebaseAuthUserCollisionException){
                Toast.makeText(getApplicationContext(), text: "You are already registered", Toast.LENGTH_SHORT).show();
                progressBar.setVisibility(View.GONE);
            }
            else{
            Toast.makeText( context: Register.this, task.getException().getMessage(),Toast.LENGTH_LONG).show();
                progressBar.setVisibility(View.GONE);
        }
        }
});
```

Figure 4.2.3 Registration Code Snippet

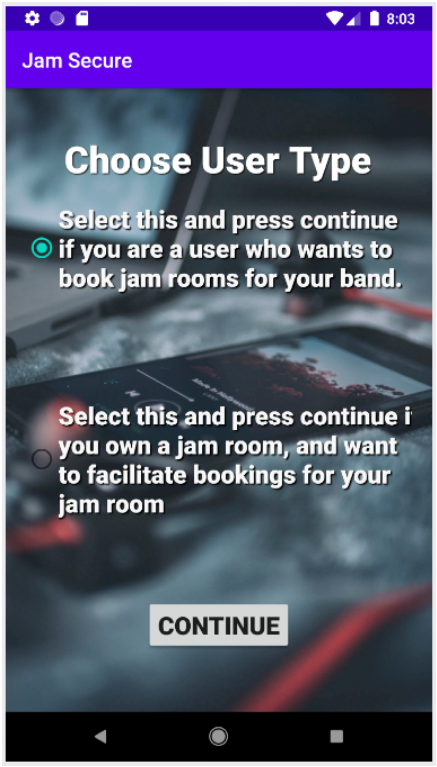


Figure 4.2.4 Choose user type activity

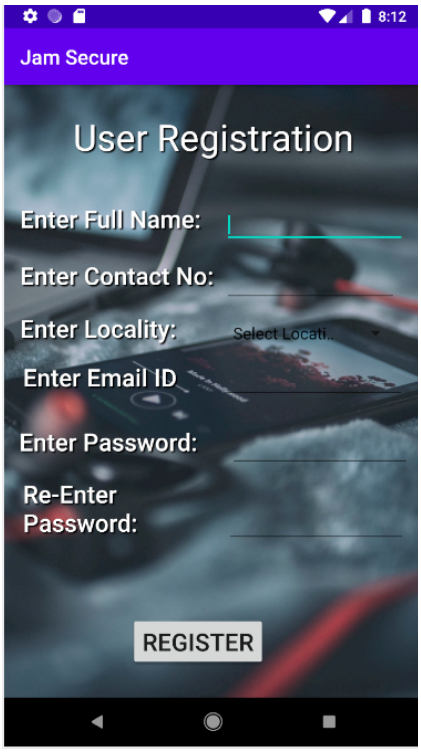


Figure 4.2.5 User Registration activity

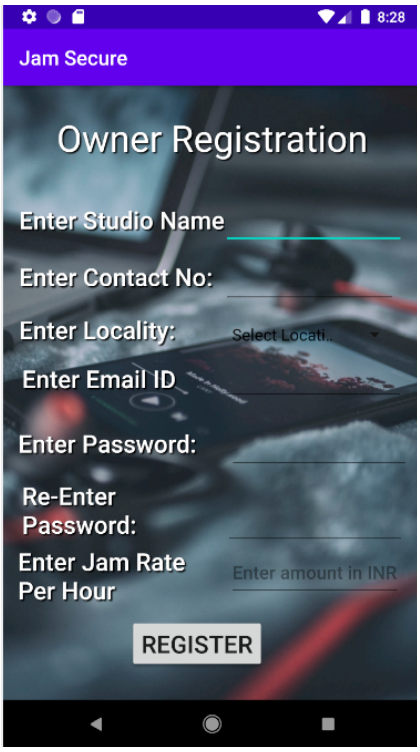


Figure 4.2.6 Owner Registration activity

Dashboards:

Once successful login or registration is done, the user is taken to a dashboard respective to the user type. The dashboard has nothing but various floating buttons to navigate to the activities that provide respective features depending on the users.

Figure 4.2.7 Dashboard Code navigation Snippet:

```
package com.example.jamsecure;

import ...

public class RoomOwner extends AppCompatActivity {
    @Override
    public void onBackPressed() {
        super.onBackPressed();
        finish();
    }
    FloatingActionButton b1;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_room_owner);
        b1= findViewById(R.id.bfa);

        b1.setOnClickListener((v) → {
                String url = "https://web-chat.global.assistant.watson.cloud.ib
                Intent i1 = new Intent(Intent.ACTION_VIEW, Uri.parse(url));
                RoomOwner.this.startActivity(i1);
        });

    }
}
```

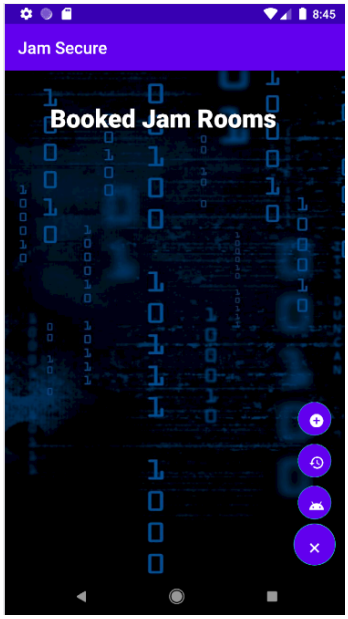


Figure 4.2.8 User Dashboard Activity

Figure 4.2.9 Owner Dashboard Activity

Booking Process:

The Users can book their jam rooms with the first floating button in their dashboard, the user is then navigated to the activity where they need to mention the preferred location, after selecting the location a list of jam rooms are displayed in the next activity depending on the location selected, this is done using intents for passing data between activities and using the data in firebase queries to fetch data from the database and display it using a firebase recycler adapter on a listview. The user can select one jam room from a list of jam rooms, and then the user will be taken to an activity where they specify date, duration and timings for booking, a datepicker and timepicker is set to facilitate this process with ease, after tapping the “Book” button, if the specified slot is available it is booked, if not a toast message is prompted to show that a slot exists.

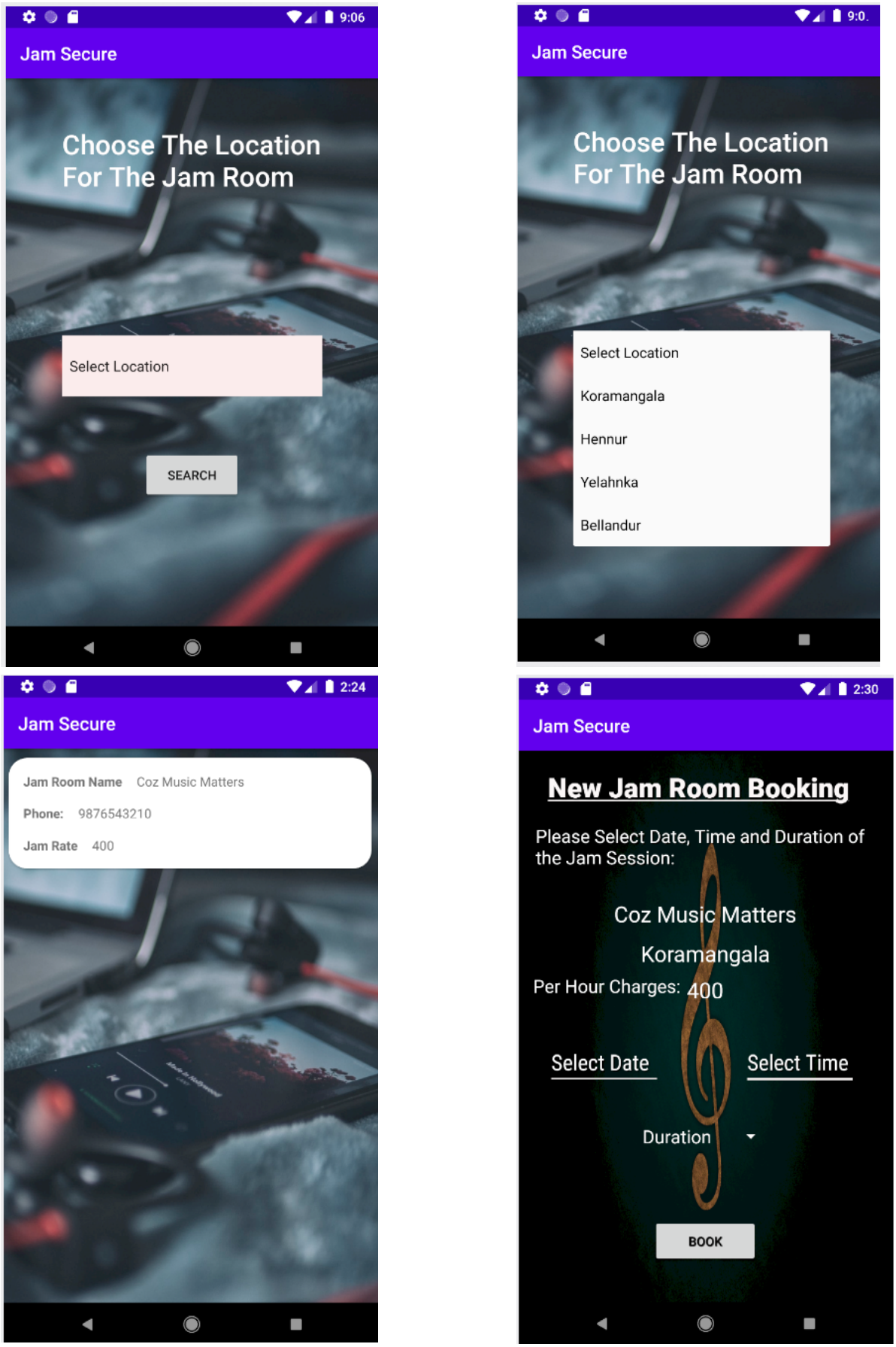
```
public class User_Select_Jam_Studio extends AppCompatActivity {

    private RecyclerView recyclerView;
    HelperAdapter adapter; // Create Object of the Adapter class
    DatabaseReference fmbase; // Create object of the
    // Firebase Realtime Database

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_user__select__jam__studio);
        Intent intent = getIntent();
        String location = intent.getStringExtra( name: "LOC");
        Log.d( tag: "THELOC",location);
        // Create a instance of the database and get
        // its reference
        fmbase = FirebaseDatabase.getInstance().getReference();
        Query mbase = fmbase.child("Owners").orderByChild("location").equalTo(location);


        recyclerView = findViewById(R.id.recycler1);

        // To display the Recycler view linearly
        recyclerView.setLayoutManager(
                new LinearLayoutManager( context: this));

        // It is a class provide by the FirebaseUI to make a
        // query in the database to fetch appropriate data
        FirebaseRecyclerOptions<FetchData> options = new FirebaseRecyclerOptions.Builder<FetchData>()
                .setQuery(mbase, FetchData.class)
                .build();
        // Connecting object of required Adapter class to
        // the Adapter class itself
        adapter = new HelperAdapter(options);
        // Connecting Adapter class with the Recycler view*/
        recyclerView.setAdapter(adapter);
    }
```

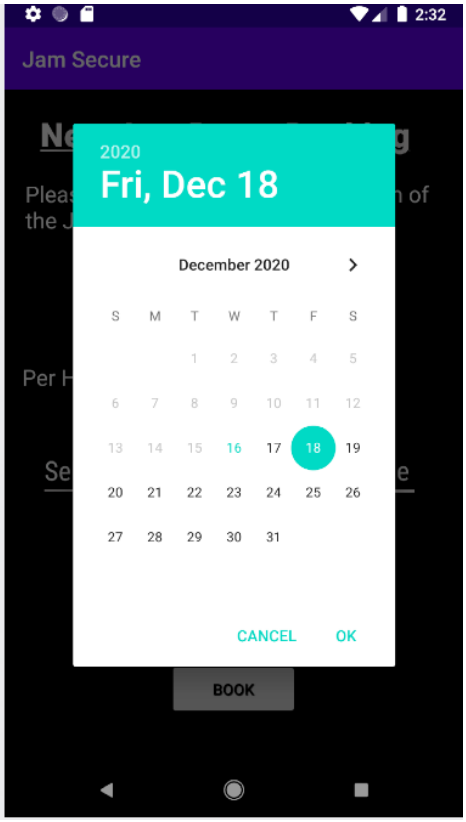
Figure 4.2.10 Code for selecting jam studio

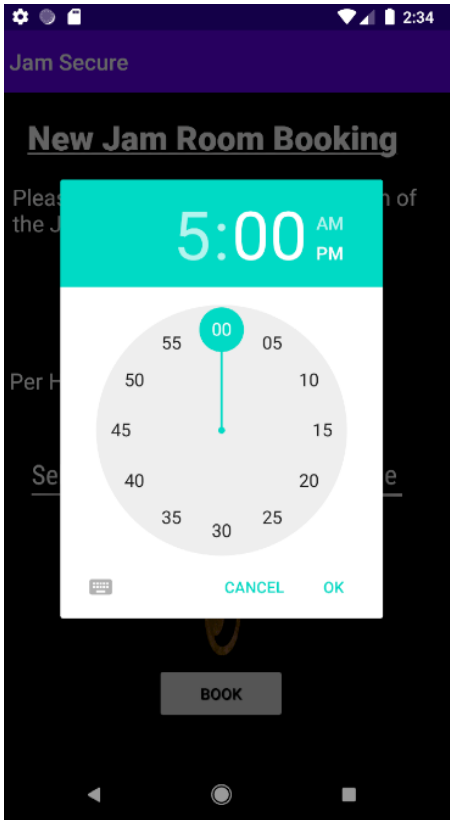
```
public class HelperAdapter extends FirebaseRecyclerAdapter<FetchData, HelperAdapter.jamroomsViewholder> {
    Context context;
    public HelperAdapter(
            @NonNull FirebaseRecyclerOptions<FetchData> options)
    {
        super(options);
    }
    // Function to bind the view in Card view(here
    // "person.xml") iwth data in
    // model class(here "person.class")
    @Override
    protected void onBindViewHolder(@NonNull final jamroomsViewholder holder, int position, @NonNull FetchData model)
    {
        final String em, nam, rate, loc;
        // Add firstname from model class (here
        // "person.class")to appropriate view in Card
        // view (here "person.xml")
        holder.JamName.setText(model.getSname());
        Log.d( tag: "NAM",model.getSname());

        // Add lastname from model class (here
        // "person.class")to appropriate view in Card
        // view (here "person.xml")
        holder.Phone.setText(model.getPhone());

        // Add age from model class (here
        // "person.class")to appropriate view in Card
        // view (here "person.xml")
        holder.JamRate.setText(model.getJam_rate());
        em=model.getSemail();
        nam=model.getSname();
        rate=model.getJam_rate();
        loc=model.getLocation();
      holder.itemView.setOnClickListener((view) -> {
                context = view.getContext();
              Intent intent = new Intent(context, NewJam.class);
              intent.putExtra( name: "name",nam);
              intent.putExtra( name: "email",em);
              intent.putExtra( name: "rate",rate);
              intent.putExtra( name: "loc",loc);
              Log.d( tag: "Murker",em);
              context.startActivity(intent);
      });

    }

    // Function to tell the class about the Card view (here
    // "person.xml")in
    // which the data will be shown
    @NonNull
    @Override
    public jamroomsViewholder
    onCreateViewHolder(@NonNull ViewGroup parent, int viewType)
    {
        View view
                = LayoutInflater.from(parent.getContext())
                .inflate(R.layout.jamrooms, parent,  attachToRoot: false);
        return new HelperAdapter.jamroomsViewholder(view);
    }

    // Sub Class to create references of the views in Crad
    // view (here "person.xml")
    class jamroomsViewholder extends RecyclerView.ViewHolder {
        TextView JamName, Phone, JamRate;
        public jamroomsViewholder(@NonNull View itemView)
        {
            super(itemView);

            JamName = itemView.findViewById(R.id.firstname);
            Phone= itemView.findViewById(R.id.lastname);
            JamRate = itemView.findViewById(R.id.age);
        }
    }
}
```

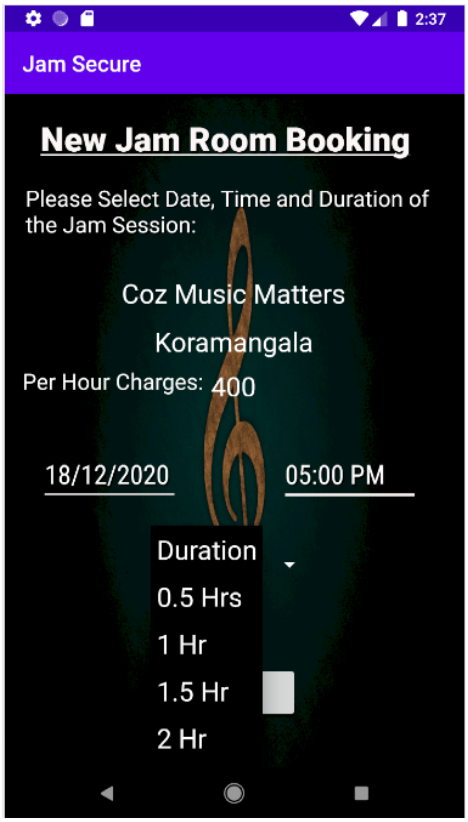
Figure 4.2.11 Adapter code

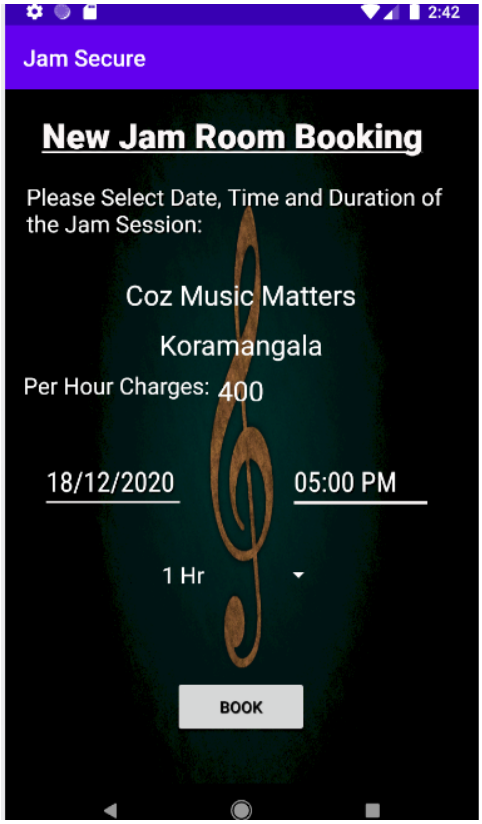
Figure 4.2.12 Cluster Image of the booking process:




2:32
Jam Secure
2020
Fri, Dec 18
December 2020
S M T W T F S
1 2 3 4 5
6 7 8 9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
CANCEL
OK
BOOK


2:34
Jam Secure
New Jam Room Booking
5:00
AM
PM
00
05
10
15
20
25
30
35
40
45
50
55
CANCEL
OK
BOOK


2:37
Jam Secure
New Jam Room Booking
Please Select Date, Time and Duration of the Jam Session:
Coz Music Matters
Koramangala
Per Hour Charges: 400
18/12/2020
05:00 PM
Duration
0.5 Hrs
1 Hr
1.5 Hr
2 Hr


2:42
Jam Secure
New Jam Room Booking
Please Select Date, Time and Duration of the Jam Session:
Coz Music Matters
Koramangala
Per Hour Charges: 400
18/12/2020
05:00 PM
1 Hr
BOOK

Chat Bot

Each type of user has a button that will lead them to a chatbot in their dashboards, the chat bot will listen to any type of meaningful text from the user and give answers accordingly, the chat bot is set up here to brief the user about the working of the app, this way the user is not stuck anywhere and has a clear idea of what to do in the app.

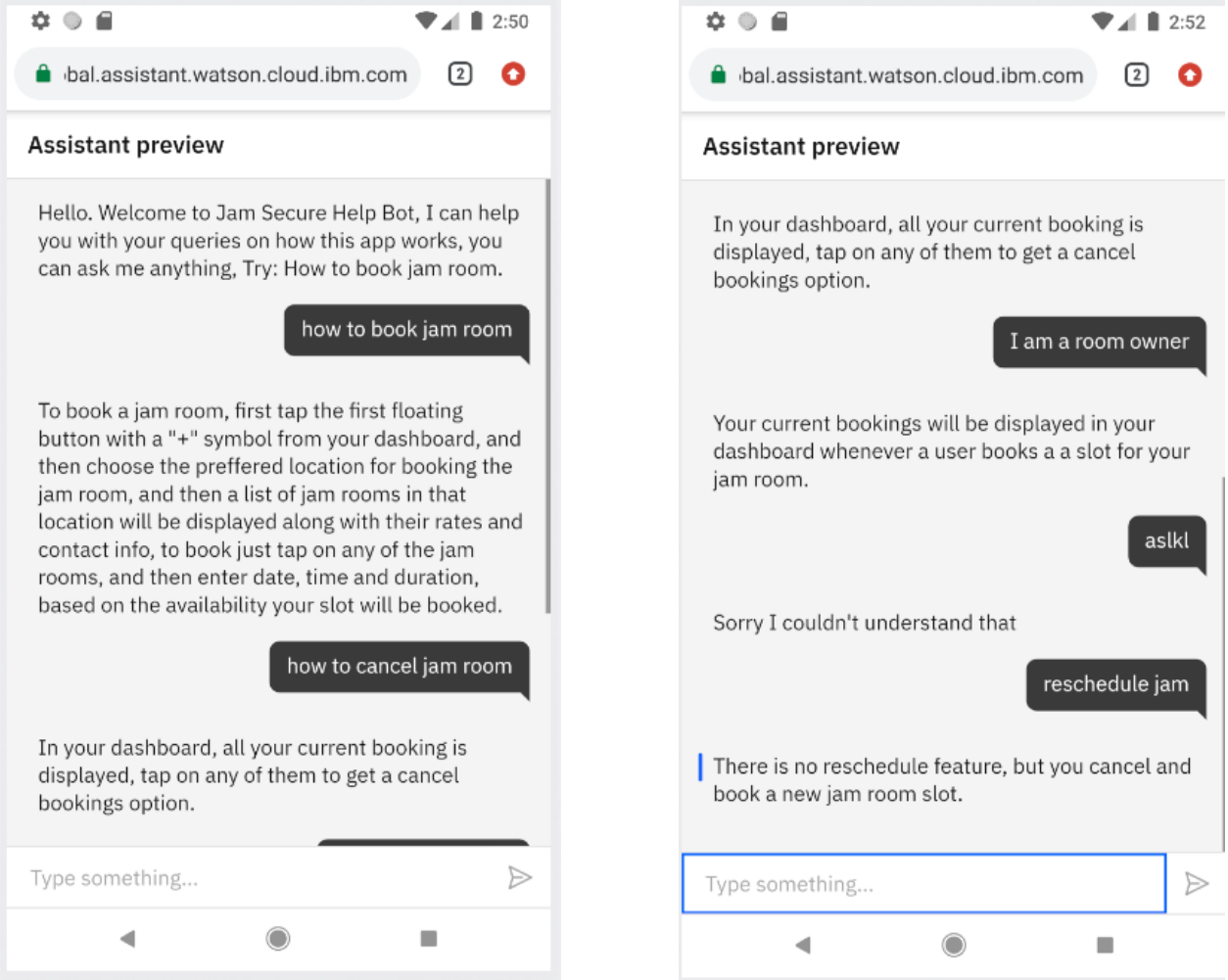


Figure 4.2.12 Chat Bot

4.3 Firebase Database Connectivity

Store and sync data wiith our NoSQL cloud database. Data is syincced across all clients in realtime, and remains available when your appp goes offline. The Firebase Realtiime Database is a cloudhosted database. Data is storred as JSON and synchronized in realtiime to every connected cliient. When you build cross-platform apps with our iOS, Android, and JavaScript SDKs, all of your cliients share one Realtiime Database instance and automatically receiive updates with the newest daaata. The Firebase Realtime Database lets you builld rich, collaborative appilications by allowing secure access to the database diirectly from client-side code. Data is persisted locally, and even while offline, realtiime events continue to fiire, giving the end user a responsive experience. When the deviice regains connection, the Realtime Database synchronizes the local data channges with the remote updates that occurrred while the client was offline, merging any confliicts automatically. The Realtime Database proviides a flexible, expression-based rules langguage, called Firebase Realtime Database Securiity Rules, to define how your data should be structiured and when data can be read from or written to. When integrated with Firebase Authentication, developeirs can define who has access to what data, and how they can access itt. In our project Firebase connectivity is maiinly to store real time user information and retrieve as quickly as possible. The informatiion regarding registration is users, the path travellled by host are all stored and retrieved in real tiime which makes the project dynamic.

4.4 IBM Cloud Watson Assistant Service

Watson Assistant is IBM's AI product that lets you build, train, and deploy conversational interactions into any application, device or channel.
Most chatbots try to mimic human interactions, which can frustrate users when a misunderstanding arises. Watson Assistant is more. It knows when to search for an answer from a knowledge base, when to ask for clarity and when to direct users to a human. Watson Assistant can be deployed in any cloud or on-premises environment – meaning smarter AI is finally available wherever you need it.
Training the assistant to understand the customer in natural language is easy thanks to AI technology. Industry-leading natural language understanding (NLU) provides training recommendations to build, run, and improve the assistant.

CHAPTER 6

CONCLUSION

Jam Secure is a proficient and anomalies free app that facilitates the booking of jam rooms with ease, and helps jam room owners keep track of their jam sessions. The development of this project has made me grow my understanding of Java programming language, android studios IDE, firebase realtime database and authentication and IBM AI powered chatbot, to a greater extent. And this knowledge can be used to make many more apps of great functionalities.