# Technical Specification

Name: Adrian Rabbitt

Student Number: 14500447

Title: Jobder

Finish Date: 20/05/2018

# Jobder Functional Specification

## Table of Contents

1. Introduction........................................................................3

2. Motivation........................................................................4

3._Glossery ........................................................................5

4. Research........................................................................6

5.System Design ........................................................................8

6.Employer & Employee Fuctionalties..................................................11.

7. Implementation ........................................................................13

8. Problems Solved ........................................................................40

8. Results........................................................................45

9. Future Work........................................................................46

# 1). Introduction

## 1.1) ***Overview***

The Android application Jobder is a virtual Employment System for all fields and skills of work. The application will try and overcome the employment barriers that numerous people are faced with. The main target group I hope to help are students, people with no qualifications and poor lexical skills. The android application consists of two modes:

- the Employer and
- the Employee.

The Employee is responsible for filling in his/her details. The Employee's main responsibility is creating/editing their profile, viewing their conversation and uploading relevant achievements as snapshots. e.g. Safe Pass.

The Employer mode consist of a login / sign up, the Employer will be brought to an activity where he/she can pick a field in which they are in search of Employees e.g. painting, gardening etc. When the Employer picks his/her field in search of employment they will be brought to a map displaying the Employees registered to that field according to their location with a set marker. The Employer will click on the employee's reference emoji picture which will allow them to view the Employees Profile, previous ratings from other employers and the Employees documents. When the Employee has studied the Employee's CV they can message the Employee and try to agree on a fixed term of work.

The Employee mode consists also of a login / sign up, the Employee is brought to an activity consisting of the users Homepage. The Home page consists of four main options consisting of "My Profile" (Employees Profile, "Contacts" (Employees Contacts)," Conversations" (Employees Conversations) and "Files" (Consisting of the Employees achievements).

There is also web application monitoring the statistics of employment with easy-to-view charts and graphs. The web application is only for statistics to view for the users of Jobder to visualise trends in employment.

## 2). Motivation

The economic crash in 2008 left a significant number of people without employment. In the last couple of years, the economic climate has been improving with new jobs being created. Although the media is focusing on the upturn, unfortunately the situation has not improved for certain groups. For people with no qualifications, poor lexical skills and students, the main problem encountered is communication rather than available work. I have studied this after personal experience with employment difficulties and have been thinking of ways to overcome this problem. The creation of this application that I will design will try and overcome these employment barrier's.

The application Jobder has all the features to grow and explore the recruitment industry. This application will differ from other employment applications due to its wide target groups and its ease, simplicity and graphical features.

## 3). Glossary

- ***Java***
  Java is an object-oriented programming language which has an independent platform.

- ***Xml***
  XML stands for extensible Mark-up Language, like HTML. The language was designed to store and transport data which was designed to be self-descriptive.

- ***SQL***
  Structured Query Language, SQL is a standardized query language for requesting information from a database.

- ***Java Script.***
  Java Script is an object-oriented computer programming language commonly used to create interactive effects within web browsers and web applications.

- ***Node.js***
  Node.js is an open source server framework that has the ability to run on certain platforms.

- **HT*ML/CSS***
  Short for Hypertext Mark-up Language, the authoring language used to create documents on the World Wide Web. CSS stands for Cascading Style Sheets.

- ***Bootstrap***
  Bootstrap is an open source toolkit for developing with HTML, CSS, and JS

## 4). Research

During the implementation of the Application Jobder there was quite a lot of research and investigation preformed from the Functional Specification to the finished project.

Here are samples of subjects of research I found that had a significant impact on the Application.

- ***Android Studio.***
  https://developer.android.com/studio/
  Android Studio is the official Integrated Development Environment (IDE) for Android app development, based on IntelliJ. On top of IntelliJ's powerful code editor and developer tools, Android Studio offers even more features that enhance your productivity when building Android apps, such as a, flexible Gradle-based build system and a fast and feature-rich emulator

- ***Firebase.***
  *https://www.androidauthority.com/introduction-to-firebase-765262/*
  Firebase is a mobile platform from Google offering a number of different features that you can pick 'n mix from. Specifically, these features revolve around cloud services, allowing users to save and retrieve data to be accessed from any device or browser. This can be useful for such things as cloud messaging, hosting, crash reporting.

- ***Location Android.***
  https://developer.android.com/guide/topics/location/strategies
  When developing a location-aware application for Android, you can utilize GPS and Android's Network Location Provider to acquire the user location. Although GPS is most accurate, it only works outdoors, it quickly consumes battery power, and doesn't return the location as quickly as users want. Android's Network Location Provider determines user location using cell tower and Wi-Fi signals, providing location information in a way that works indoors and outdoors, responds faster, and uses less battery power.

- ***Cloud Messaging.***
  https://firebase.google.com/docs/cloud-messaging/
  Using FCM, you can notify a client app that new email or other data is available to sync. You can send notification messages to drive user re-engagement and retention. For use cases such as instant messaging, a message can transfer a payload of up to 4KB to a client app.

- ***Geocoding .***
  *https://developers.google.com/maps/documentation/geocoding/start*
  Geocoding is the process of converting addresses (like a street address) into geographic coordinates (like latitude and longitude), which you can use to place markers on a map or position the map.

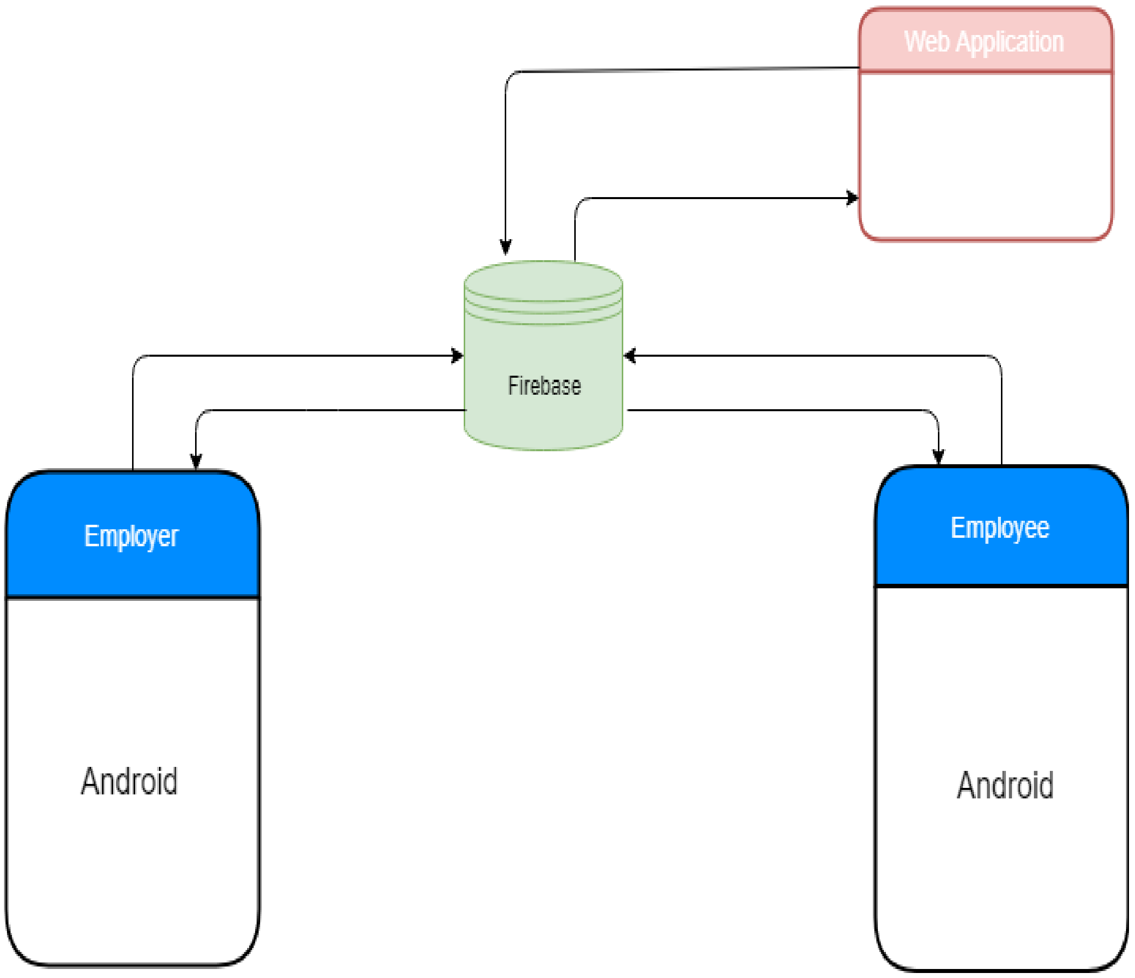
- ***Bootstrap.***
  https://getbootstrap.com/
  Bootstrap is an open source toolkit for developing with HTML, CSS, and JS. Quickly prototype your ideas or build your entire app with our Sass variables

and mixins, responsive grid system, extensive prebuilt components, and powerful plugins built on jQuery.

- ***Fusion Charts.***
  *https://www.fusioncharts.com/*
  Fusion charts are an open source library that allows the implementation of graphical charts.

## 5).System Architecture.

**Figure 1 -** Diagram of the System Architecture of the application Jobder.



***Firebase***

Firebase is the server and database storage for the application. Allowing Store and sync data with our NoSQL cloud database. Data is synced across all clients in real time and remains available when the app goes offline.

***Employer***

Employer is a version of the android application.

***Employee***

Employee is a version of the android application.

***Web Application***

Web application displays Employment statistics for the application.

**Figure 2 -** High Level Design of the application Jobder.

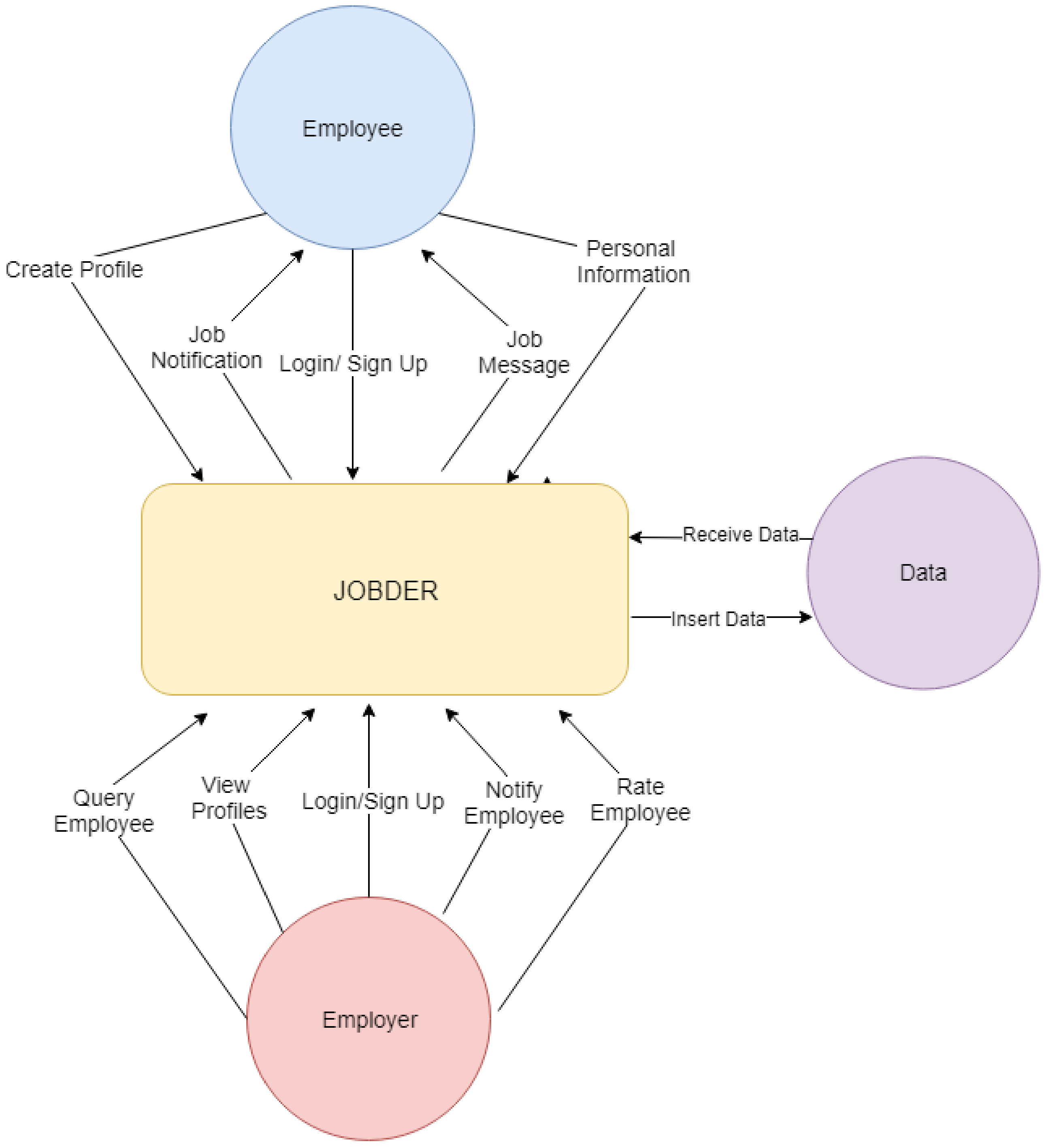
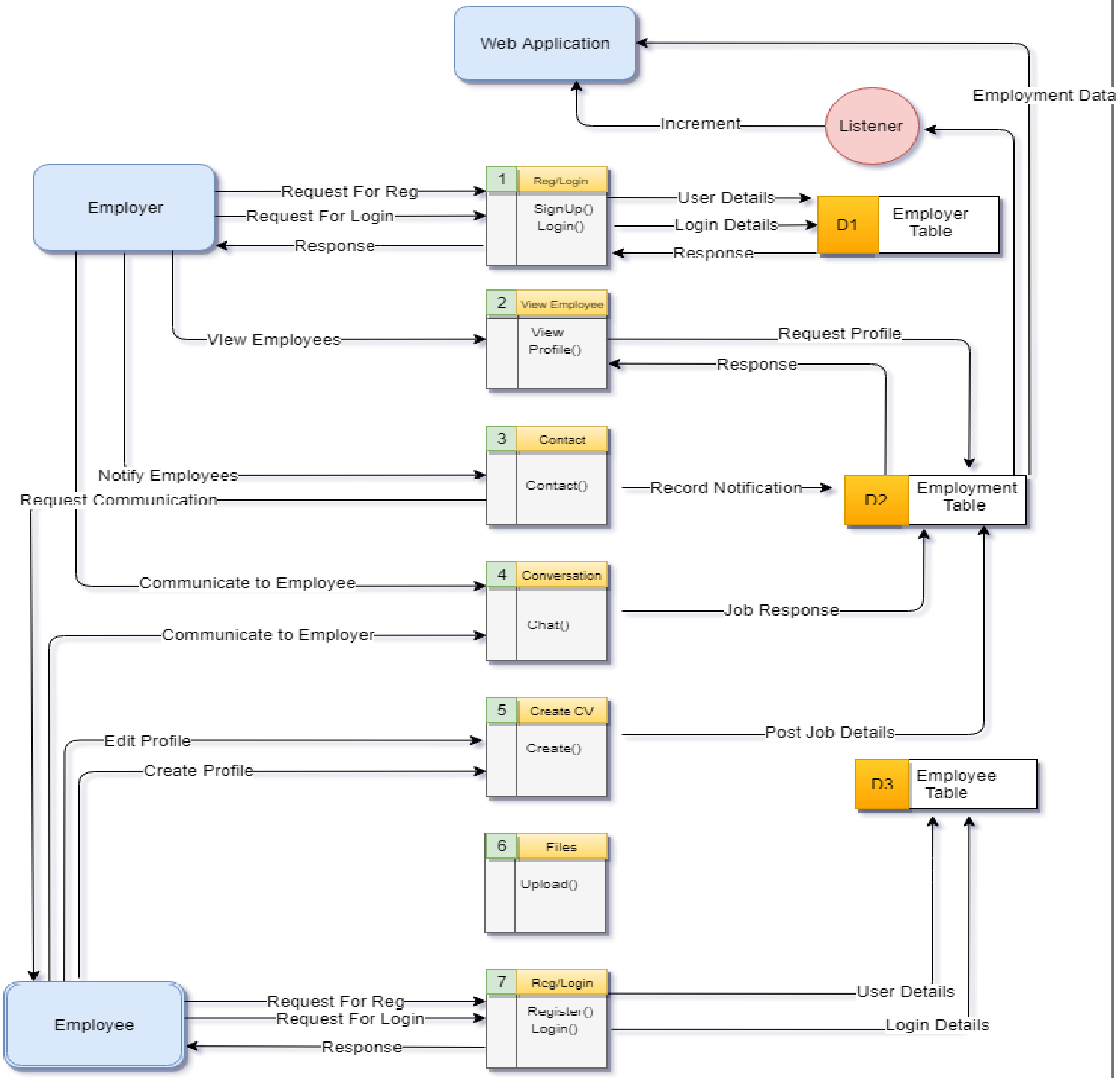


**Figure 3 –** DFD of the application Jobder



# 6). Employee & Employer Functionalities.

## 2.1 ***Product / System Functions (Briefly Explained)***

- ***Login/Sign up***
  When the users of Jobder have initialized their choice of modes they will be brought to the Login page where they will be allowed to sign up or login with their personal details. If the user wishes to sign up they will be brought to a separate activity for registration.

- ***Employees Home Page*****.**
  The Employees Home Screen consists of the options the Employee has within the application. Profile, Contacts, Conversation and Files. There are two buttons on the top of the page to locate setting's and current information about the application.

- ***CV/Profile***
  The Employee can edit/ create their profile, allowing them to change their profile picture, choose employment field, fill in personal information, fill in previous employment, and submit their data according to their current location.

- ***Conversation's***
  In the Employee mode of the application the user has also the opportunity to view his/her recent conversation's. The Employee does not have the opportunity to start a thread of conversation's.

- ***Contact's***
  In the Employee mode of the application the user has also the opportunity to view his/her current contacts available consisting of the Employers. By clicking on a contact, he/she can start a message thread.

- ***Files***
  The Employee also has the choice to upload achieved documents or references as an image. The Employee simply clicks on the file upload button to upload a file onto their account.

- ***Employee Settings***
  When the Employee wants to view or change their settings they just simply click on the setting icon on the top right-hand corner of the Employees home screen.

- ***Employers Home Page***
  The Employers Home Screen consists of all the Jobs available for the Employer to choose from in search of Employees.

- ***Instant Messaging.***
  Provides a communication interface between Employers and Employees, to send and recieve messages.

- ***Profile.***
  The Employee is displayed to the user using the Profile Activity which consists of the Employees Profile image and options to view profile content, such as ratings and Certificates (Employees achievements).

- ***Employer Settings.***
  When the Employer wants to view or change their settings they just simply click on the setting icon on the top right-hand corner of the Employers home screen.
  The Settings include the current location the Employer is viewing profiles in, Gender filtering showing Men/Women in the local area using switches. The range in which profiles can be viewed in KM and the age range. The settings use Shared Preferences and database fields to store the current state of each settings field.
  When the Employer wishes to change the location in which the profiles are searched he/she simply click on the viewing field where a popup dialog appears, by entering in the place name or the address of the destination, the application geocodes the input address into latitude and longitude values where the viewing range is set for the Employer. Geocoding is the process of transforming a street address or the name of a location into location coordinates, (Latitude, Longitude).

- ***Push Notifications***
  Once a message is received by the application the user is notified through a push notification. The push notifications were created through FCM where each account that the application was used in there is a saved unique token ID of that Device.

- ***Web Application.***
  There is a simple web application that allows users of the application Jobder to login using their same email and passwords to view different statistics of the application. The statistics display the number of different Employment fields in the application. The application was implemented using HTML, CSS, JavaScript and Node.js.
  The UI of the web application was implemented using Bootstrap which is an open source toolkit for developing HTML, CSS and JS. The graphs in the application using fusion charts which is also an open source platform for creating charts.

# 7). Implementation

Here I will explain the implementation of certain main features of the application.

**Login**

The Sign-Up page consists of four fields to fill in consisting of the users Name, Email, Password, a field to renter passwords for correctness and a drop-down spinner to clarify the user's mode. Each of the fields consist of a Text View and Edit Text respectively.

To assist in user adaption there is specific indications to highlight to the users what fields are mandatory and optional. This is portrayed to the user by a red asterisk which will be visible to the user if they have not completed the corresponding field.

Each EditText (field on sign up) has an attached Text Watcher which is a method that is called to notify the application when a char sequence is added changed or removed from the set text field. When the user adds text to the fields the text watcher notifies change and appends a Spannable String Builder which had the text field title and the red asterisks.

When the user has entered his/her details there are several individual tests that the char sequence must pass before the user is signed up.

1) All the fields must not consist of the value null. This is the first and simplest test that will ensure the functions are not dealing with null references that could cause the application to crash.

2) When the user is entering his/her email address the application must ensure this is a valid registered Gmail account.

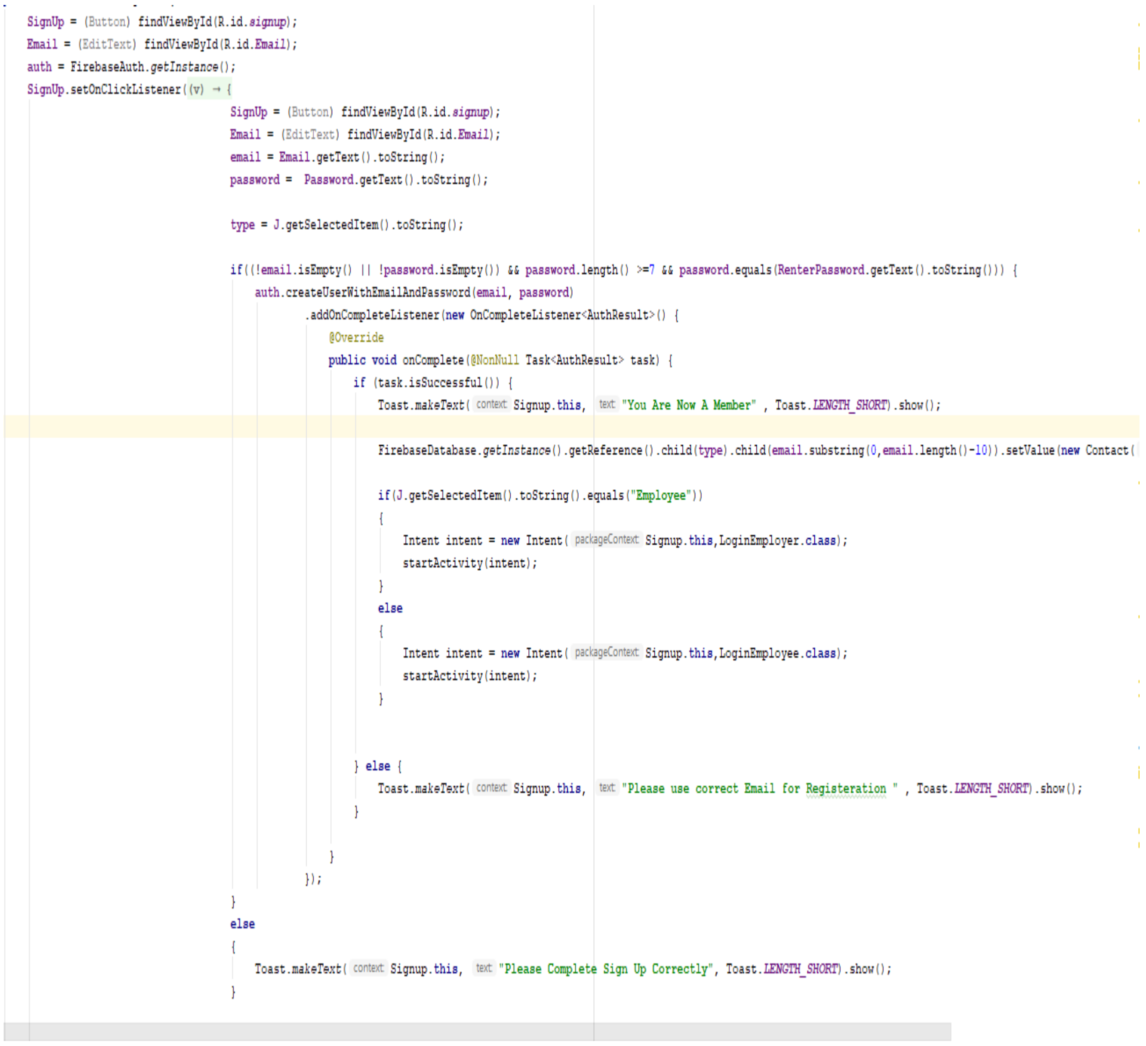
3) The users Password must be greater than or equal to 8 characters.

4) When re-entering the Password, the password must equal to the previous password.

5) Asterisk are displayed when fields are currently not completed.

If the user of the application has forgotten their password, they can click on forgot my password where a pop-up dialog will be displayed to enter the users email, where instructions will be sent by email on how to rest their password.

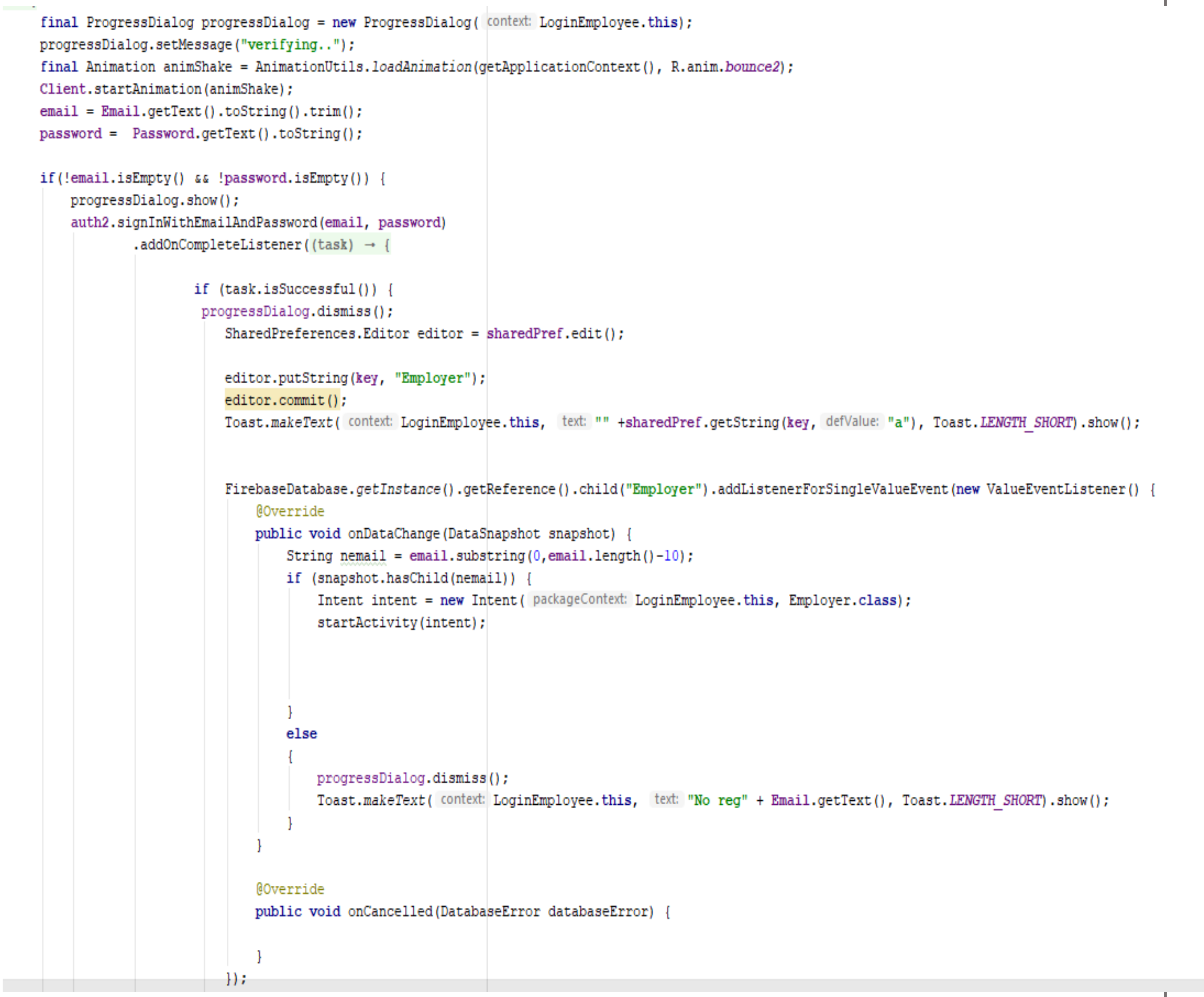
**Figure 4 -** Screen shot of the code to Sign Up Users:



```
SignUp = (Button) findViewById(R.id.signup);
Email = (EditText) findViewById(R.id.Email);
auth = FirebaseAuth.getInstance();
SignUp.setOnClickListener((v) → {
                                SignUp = (Button) findViewById(R.id.signup);
                                Email = (EditText) findViewById(R.id.Email);
                                email = Email.getText().toString();
                                password =  Password.getText().toString();

                                type = J.getSelectedItem().toString();

                                if((!email.isEmpty() || !password.isEmpty()) && password.length() >=7 && password.equals(RenterPassword.getText().toString())) {
                                    auth.createUserWithEmailAndPassword(email, password)
                                            .addOnCompleteListener(new OnCompleteListener<AuthResult>() {
                                                @Override
                                                public void onComplete(@NonNull Task<AuthResult> task) {
                                                    if (task.isSuccessful()) {
                                                        Toast.makeText( context: Signup.this,  text: "You Are Now A Member" , Toast.LENGTH_SHORT).show();

                                                        FirebaseDatabase.getInstance().getReference().child(type).child(email.substring(0,email.length()-10)).setValue(new Contact(

                                                        if(J.getSelectedItem().toString().equals("Employee"))
                                                        {
                                                            Intent intent = new Intent( packageContext: Signup.this,LoginEmployer.class);
                                                            startActivity(intent);
                                                        }
                                                        else
                                                        {
                                                            Intent intent = new Intent( packageContext: Signup.this,LoginEmployee.class);
                                                            startActivity(intent);
                                                        }


                                                    } else {
                                                        Toast.makeText( context: Signup.this,  text: "Please use correct Email for Registeration " , Toast.LENGTH_SHORT).show();
                                                    }

                                                }
                                            });
                                }
                                else
                                {
                                    Toast.makeText( context: Signup.this,  text: "Please Complete Sign Up Correctly", Toast.LENGTH_SHORT).show();
                                }
```

When the user logins their details are validated against Firebases NoSql database.

**Figure 5 -** Screen shot of the code for users to Sign in.



```
final ProgressDialog progressDialog = new ProgressDialog( context: LoginEmployee.this);
progressDialog.setMessage("verifying..");
final Animation animShake = AnimationUtils.loadAnimation(getApplicationContext(), R.anim.bounce2);
Client.startAnimation(animShake);
email = Email.getText().toString().trim();
password =  Password.getText().toString();

if(!email.isEmpty() && !password.isEmpty()) {
    progressDialog.show();
    auth2.signInWithEmailAndPassword(email, password)
            .addOnCompleteListener((task) → {

                    if (task.isSuccessful()) {
                     progressDialog.dismiss();
                        SharedPreferences.Editor editor = sharedPref.edit();

                        editor.putString(key, "Employer");
                        editor.commit();
                        Toast.makeText( context: LoginEmployee.this,  text: "" +sharedPref.getString(key, defValue: "a"), Toast.LENGTH_SHORT).show();


                        FirebaseDatabase.getInstance().getReference().child("Employer").addListenerForSingleValueEvent(new ValueEventListener() {
                            @Override
                            public void onDataChange(DataSnapshot snapshot) {
                                String nemail = email.substring(0,email.length()-10);
                                if (snapshot.hasChild(nemail)) {
                                    Intent intent = new Intent( packageContext: LoginEmployee.this, Employer.class);
                                    startActivity(intent);



                                }
                                else
                                {
                                    progressDialog.dismiss();
                                    Toast.makeText( context: LoginEmployee.this,  text: "No reg" + Email.getText(), Toast.LENGTH_SHORT).show();
                                }
                            }

                            @Override
                            public void onCancelled(DatabaseError databaseError) {

                            }
                        });
```

## Splash Screen/ Options for Users

When the application is launching a splash, screen is displayed consisting of the Application logo which is running on a separate thread separate from the main UI thread. When the application is launched for the first time by the user he/she is brought to an options page which will allow them to choose between modes of administration the Employee or the Employer. The page consists of two buttons Employee/ Employer and a logo screen.
When the options page is executed, the UI options are transformed onto the display screen via animations. The animation was created locally on the device which its duration is determined by milliseconds.  <translate> is used in xml to repeatedly fix the coordinates of an object
 for a specific duration along the X and Y axis.

## CV/ Profile.

When the Employee wishes to add a profile or edit their profile they will be brought to the CV Activity where they are required to complete all mandatory and optional fields. They also have the chance to upload an image as their Profile Picture.

The fields in the CV Activity are formatted in a Linear Layout, were the fields are represented as edit Texts highlighted with Text-Views. Optional choice fields such as Title, Gender and Job description are displayed using Spinners which provide a quick way to select one value from a set. In the default state the spinner shows its currently selected value.

When the user wishes to upload their Profile Picture they select on the add icon located hovering over the Profile picture screen. When the add icon is clicked the user has the chance to upload an image from the Camera-Roll on the local device. This is achieved through an intent.ActionPick that was set to search through all pictures in the camera roll, returning a URI from the path of local data. The received file path is passed into a startActivityForResult() method. Here this preforms a process, separate from the main UI thread to get the selected image from the camera roll.

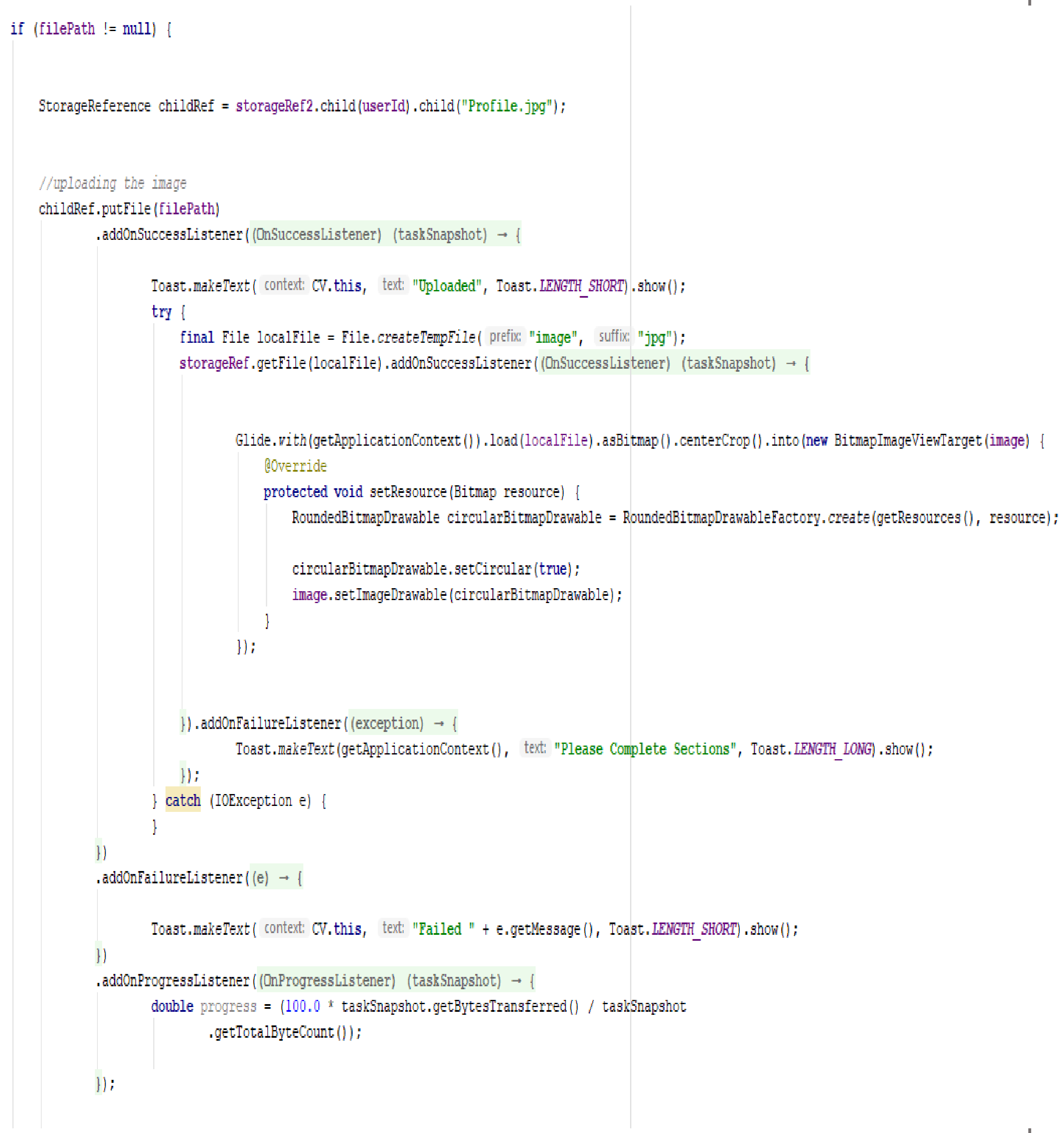
When the image is received it is uploaded to Firebase Storage according the users unique ID. Firebase cloud storage is a separate repository from Firebase database, it is a simple cost-effective object storage service built from google. It's is also very good for storage security regardless of what the quality of the network. When the file path of the image is located, it is stored in a placeholder, where it is then inserted into the storage programmatically. When the new image is uploaded into storage it also must be signalled to retrieve the image to ensure to the user that their image has changed successfully. This is achieved from creating a reference to the storage database where the file was uploaded and saving the file path to a placeholder where it can be downloaded using a specified service. The service I used to download the users image was Glide, which is an image Loader Library for android developed by bumptech. Its two main attributes are "load" and "into" where load establishes the file path from the server where the image is located, where into establishes the destination of the downloaded file in this case it is an Image View.

Once all the fields are filled in android the Employee can submit all their personal data. When the fields are completed and has passed the acceptance test all the edit texts fields are inserted into an object of its type. The object is then uploaded to the Firebase real time database under the users unique ID. Once the data is submitted it must be retrieved to show that the Employee that their data had been submitted successfully. This is achieved by referencing the Employees object, where it is downloaded, and the unique fields are pulled from the current object downloaded within the application e.g. Name, Address, Phone Number.

When the Employee submits their data, their current location represented by latitude and longitude values are stored into their current object within the database. The Location is retrieved from the Location Tracker class where the Location Manager retrieves the last known location from the GPS provider.

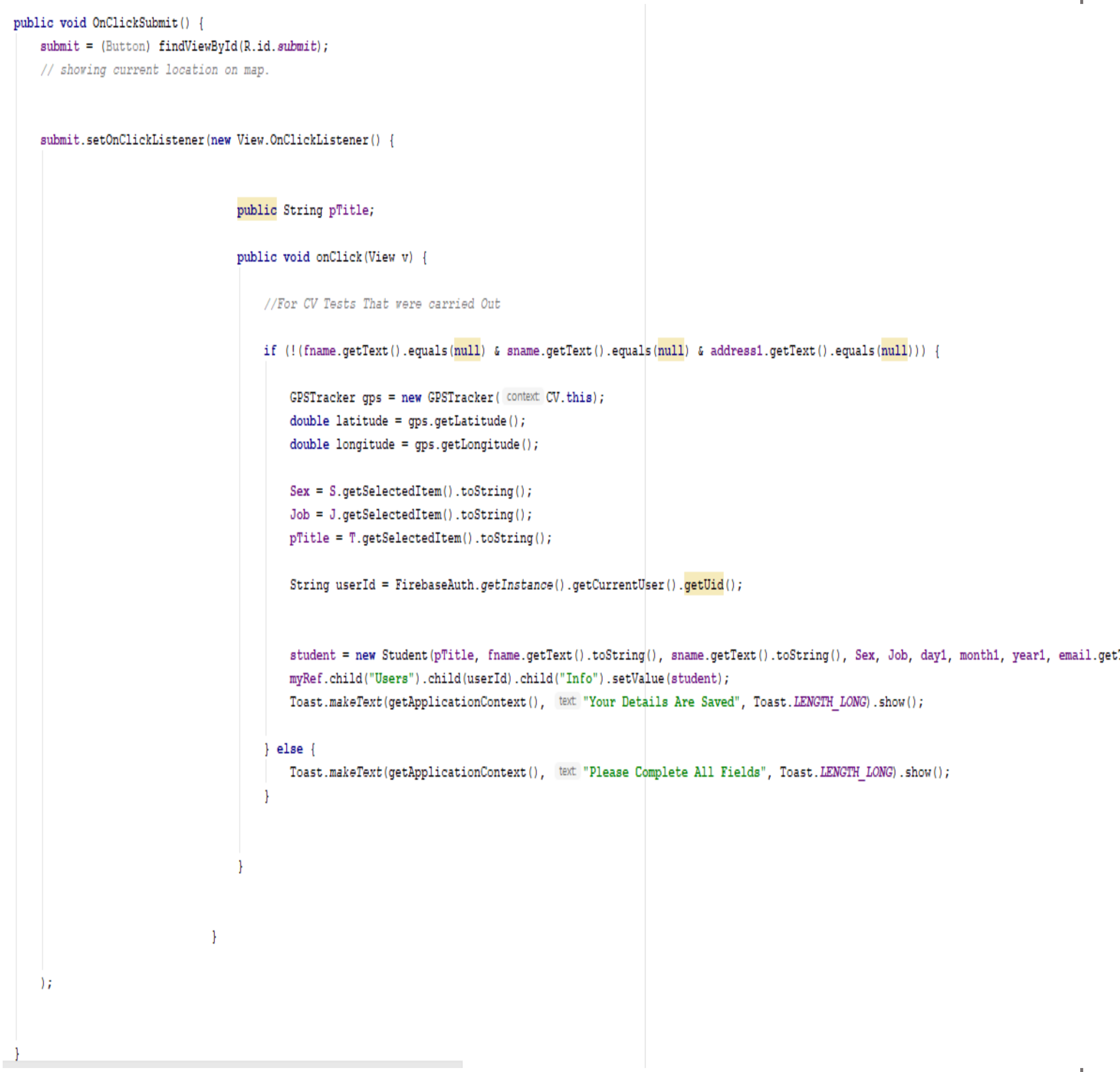
Latitude and Longitude values are then pulled from the location interface using getLatitude() and getLongitude().

**Figure 6** - screenshot of the code to upload and retrieve the Employee’s Profile.



```
if (filePath != null) {


    StorageReference childRef = storageRef2.child(userId).child("Profile.jpg");


    //uploading the image
    childRef.putFile(filePath)
            .addOnSuccessListener((OnSuccessListener) (taskSnapshot) → {

                Toast.makeText( context: CV.this,  text: "Uploaded", Toast.LENGTH_SHORT).show();
                try {
                    final File localFile = File.createTempFile( prefix: "image",  suffix: "jpg");
                    storageRef.getFile(localFile).addOnSuccessListener((OnSuccessListener) (taskSnapshot) → {


                        Glide.with(getApplicationContext()).load(localFile).asBitmap().centerCrop().into(new BitmapImageViewTarget(image) {
                            @Override
                            protected void setResource(Bitmap resource) {
                                RoundedBitmapDrawable circularBitmapDrawable = RoundedBitmapDrawableFactory.create(getResources(), resource);

                                circularBitmapDrawable.setCircular(true);
                                image.setImageDrawable(circularBitmapDrawable);
                            }
                        });


                    }).addOnFailureListener((exception) → {
                        Toast.makeText(getApplicationContext(),  text: "Please Complete Sections", Toast.LENGTH_LONG).show();
                    });
                } catch (IOException e) {
                }
            })
            .addOnFailureListener((e) → {

                Toast.makeText( context: CV.this,  text: "Failed " + e.getMessage(), Toast.LENGTH_SHORT).show();
            })
            .addOnProgressListener((OnProgressListener) (taskSnapshot) → {
                double progress = (100.0 * taskSnapshot.getBytesTransferred() / taskSnapshot
                        .getTotalByteCount());

            });
```

**Figure 7** - Code used to submit the Employees Profile according to their current location.



```
public void OnClickSubmit() {
    submit = (Button) findViewById(R.id.submit);
    // showing current location on map.


    submit.setOnClickListener(new View.OnClickListener() {


                              public String pTitle;

                              public void onClick(View v) {

                                  //For CV Tests That were carried Out

                                  if (!(fname.getText().equals(null) & sname.getText().equals(null) & address1.getText().equals(null))) {

                                      GPSTracker gps = new GPSTracker( context: CV.this);
                                      double latitude = gps.getLatitude();
                                      double longitude = gps.getLongitude();

                                      Sex = S.getSelectedItem().toString();
                                      Job = J.getSelectedItem().toString();
                                      pTitle = T.getSelectedItem().toString();

                                      String userId = FirebaseAuth.getInstance().getCurrentUser().getUid();


                                      student = new Student(pTitle, fname.getText().toString(), sname.getText().toString(), Sex, Job, day1, month1, year1, email.get
                                      myRef.child("Users").child(userId).child("Info").setValue(student);
                                      Toast.makeText(getApplicationContext(),  text: "Your Details Are Saved", Toast.LENGTH_LONG).show();

                                  } else {
                                      Toast.makeText(getApplicationContext(),  text: "Please Complete All Fields", Toast.LENGTH_LONG).show();
                                  }


                              }


                          }

    );


}
```

## Conversations.

Conversations are displayed in the format of a list view, where each conversation has the Seders name, date, content of message and a button for the Employee to delete the current conversation. Conversations are displayed according the unique match up of user Ids in the database. Each user has a list of current conversation Ids displayed under their individual id.

To notify change in the data there is an event listener set on that field to signal the application in change of data. The event Listener is an interface in the View class that contains one single call back method, supporting interaction with the applications UI. Each conversation does not match the default type of the List view which is a String therefore a customized Array Adapter has to be created to support the display of custom objects.

This only displays the current state of conversation in the current moment, so to establish use of the conversation's functionality to link to the correct message field. To provide the correct link between messages I created two Array List's that contain the user's names and user's ids corresponding to the correct index of conversations, so that when a conversation is clicked the Array Adapter will be able to reference the user by getting the clicked position in the array list. Also, when the delete button is clicked the node that corresponds to the user's link is removed from the database. This is done by retracting the node's in the database corresponding the position of the node and removing the value.

When the user clicks on their specific conversation object they are brought to an instant message Activity (ChatApplication.java) where they can instant message that user by connecting to the real time database.

**Figure 8 -** diagram showing the custom Adapter,

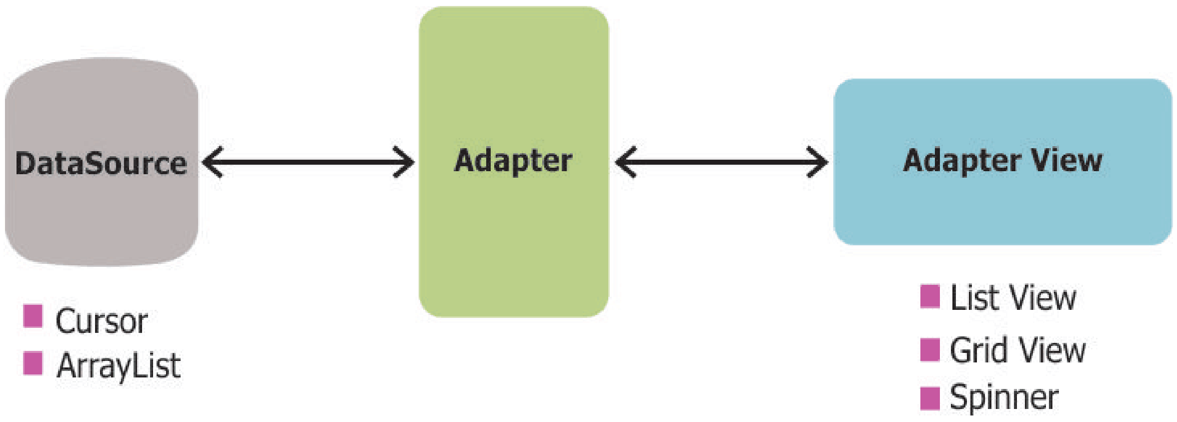
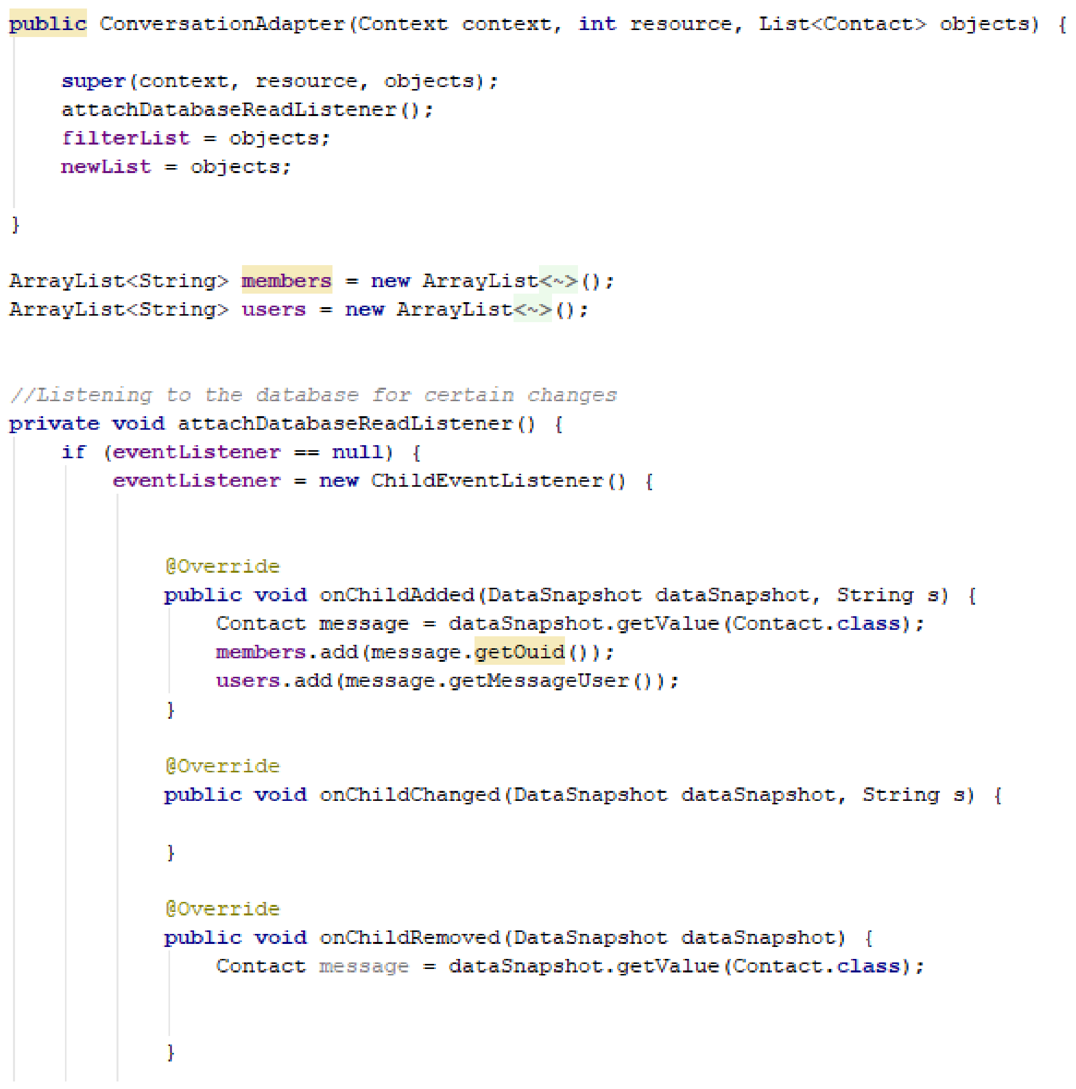
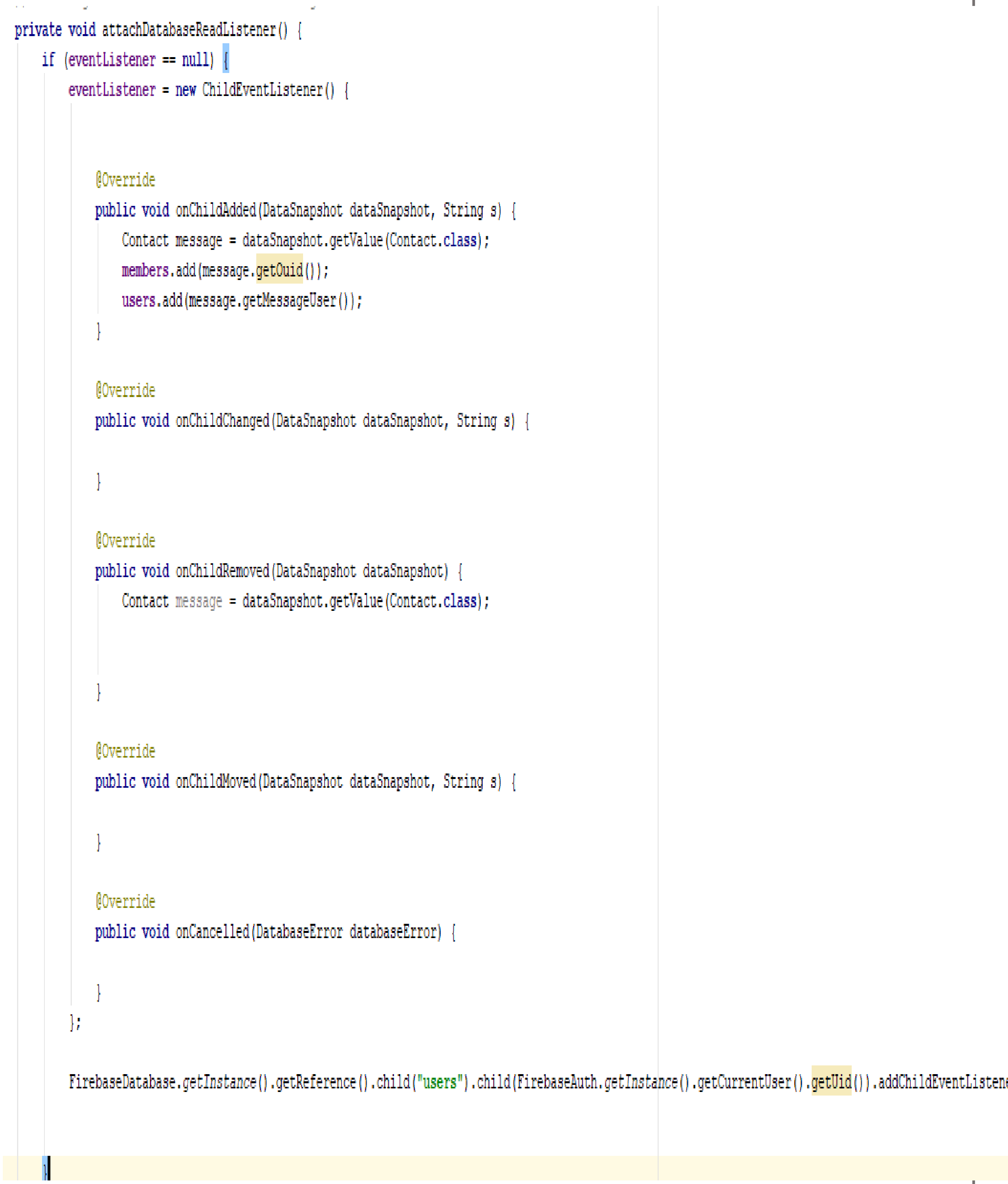


**Figure 9** - screen shot of the Conversation Custom Adapter class.



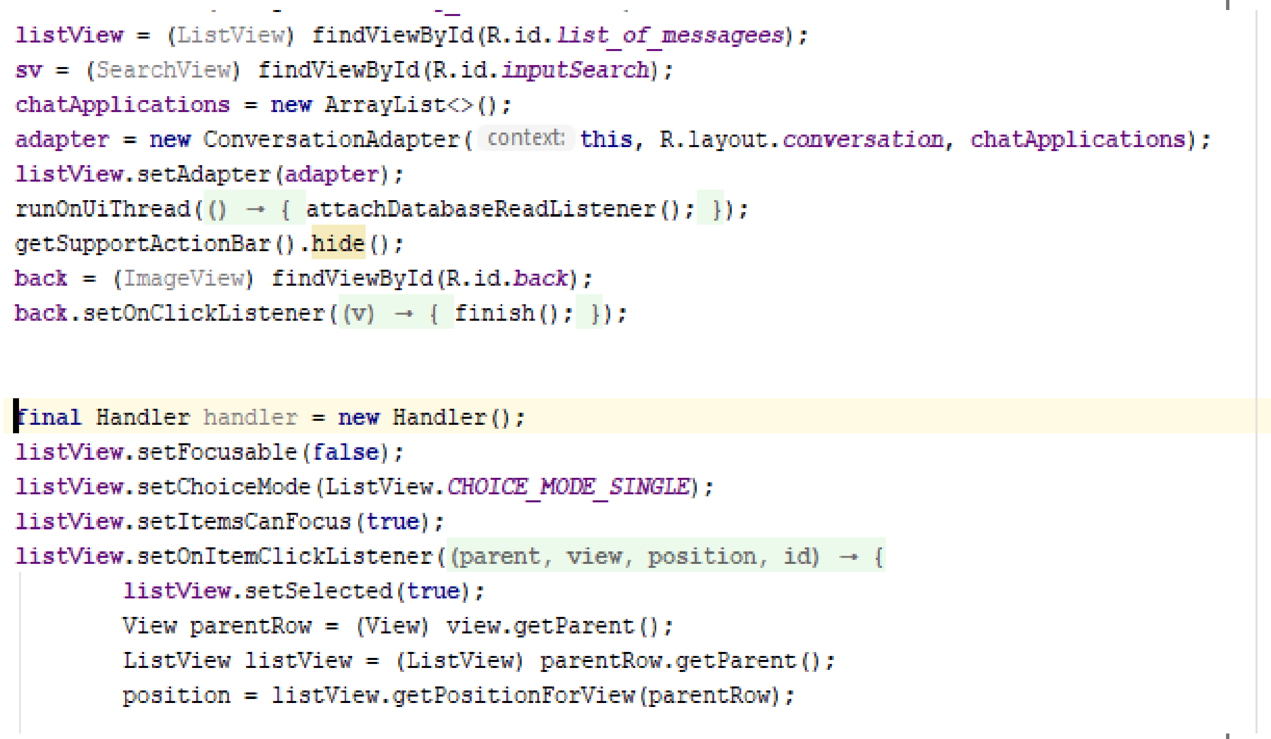
```java
public ConversationAdapter(Context context, int resource, List<Contact> objects) {

    super(context, resource, objects);
    attachDatabaseReadListener();
    filterList = objects;
    newList = objects;

}

ArrayList<String> members = new ArrayList<~>();
ArrayList<String> users = new ArrayList<~>();


//Listening to the database for certain changes
private void attachDatabaseReadListener() {
    if (eventListener == null) {
        eventListener = new ChildEventListener() {


            @Override
            public void onChildAdded(DataSnapshot dataSnapshot, String s) {
                Contact message = dataSnapshot.getValue(Contact.class);
                members.add(message.getOuid());
                users.add(message.getMessageUser());
            }

            @Override
            public void onChildChanged(DataSnapshot dataSnapshot, String s) {

            }

            @Override
            public void onChildRemoved(DataSnapshot dataSnapshot) {
                Contact message = dataSnapshot.getValue(Contact.class);


            }
```

**Figure 10** - Code implemented for the eventListner().



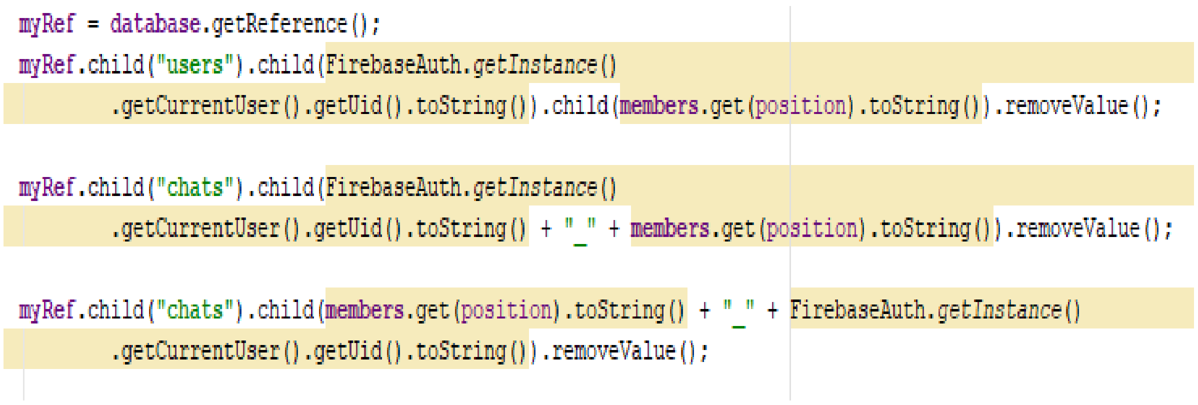
```java
private void attachDatabaseReadListener() {
    if (eventListener == null) {
        eventListener = new ChildEventListener() {


            @Override
            public void onChildAdded(DataSnapshot dataSnapshot, String s) {
                Contact message = dataSnapshot.getValue(Contact.class);
                members.add(message.getOuid());
                users.add(message.getMessageUser());
            }

            @Override
            public void onChildChanged(DataSnapshot dataSnapshot, String s) {

            }

            @Override
            public void onChildRemoved(DataSnapshot dataSnapshot) {
                Contact message = dataSnapshot.getValue(Contact.class);


            }

            @Override
            public void onChildMoved(DataSnapshot dataSnapshot, String s) {

            }

            @Override
            public void onCancelled(DatabaseError databaseError) {

            }
        };

        FirebaseDatabase.getInstance().getReference().child("users").child(FirebaseAuth.getInstance().getCurrentUser().getUid()).addChildEventListene


    }
```

**Figure 11** - Conversation Custom adapter.



```
listView = (ListView) findViewById(R.id.list_of_messagees);
sv = (SearchView) findViewById(R.id.inputSearch);
chatApplications = new ArrayList<>();
adapter = new ConversationAdapter( context: this, R.layout.conversation, chatApplications);
listView.setAdapter(adapter);
runOnUiThread(() → { attachDatabaseReadListener(); });
getSupportActionBar().hide();
back = (ImageView) findViewById(R.id.back);
back.setOnClickListener((v) → { finish(); });


final Handler handler = new Handler();
listView.setFocusable(false);
listView.setChoiceMode(ListView.CHOICE_MODE_SINGLE);
listView.setItemsCanFocus(true);
listView.setOnItemClickListener((parent, view, position, id) → {
        listView.setSelected(true);
        View parentRow = (View) view.getParent();
        ListView listView = (ListView) parentRow.getParent();
        position = listView.getPositionForView(parentRow);
```

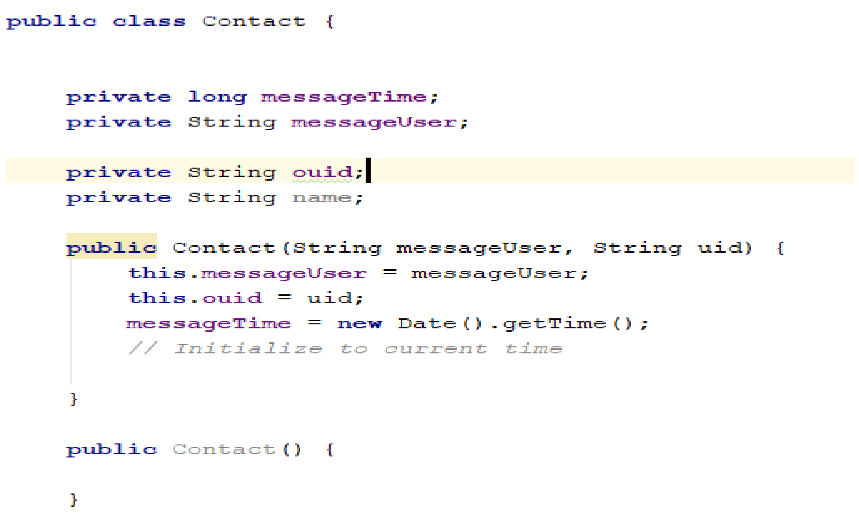
**Figure 12** - Deleting Conversations Screenshot:



```
myRef = database.getReference();
myRef.child("users").child(FirebaseAuth.getInstance()
        .getCurrentUser().getUid().toString()).child(members.get(position).toString()).removeValue();

myRef.child("chats").child(FirebaseAuth.getInstance()
        .getCurrentUser().getUid().toString() + "_" + members.get(position).toString()).removeValue();

myRef.child("chats").child(members.get(position).toString() + "_" + FirebaseAuth.getInstance()
        .getCurrentUser().getUid().toString()).removeValue();
```

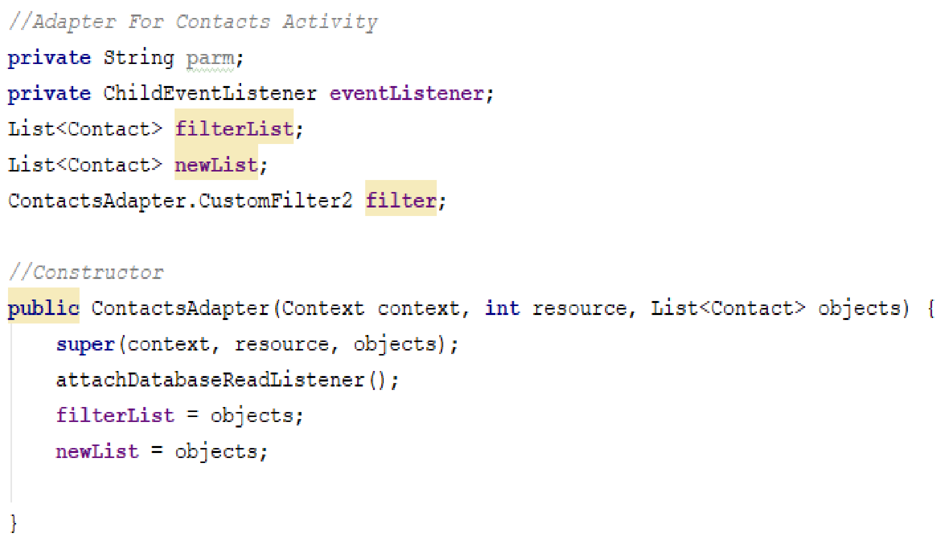
- ***Contact's***
  The Contacts functionality works in the same way as the conversations, the Employee does not have the ability to add a contact but only to revise a contact if the Employer messages the Employee. When an Employee messages an Employee they are shown up in the Conversations Activity and in the Contacts Activity. When the user wishes to message an Employer that is registered to that Employee they can click on the contact that will bring them to the instant messaging activity. This Process works in the same as the conversation functionality.

**Figure 13** - Screen Shot of Contact object:



```
public class Contact {


    private long messageTime;
    private String messageUser;

    private String ouid;
    private String name;

    public Contact(String messageUser, String uid) {
        this.messageUser = messageUser;
        this.ouid = uid;
        messageTime = new Date().getTime();
        // Initialize to current time

    }

    public Contact() {

    }
```

**Figure 14** - Screen Shot of Contact Custom Adapter.



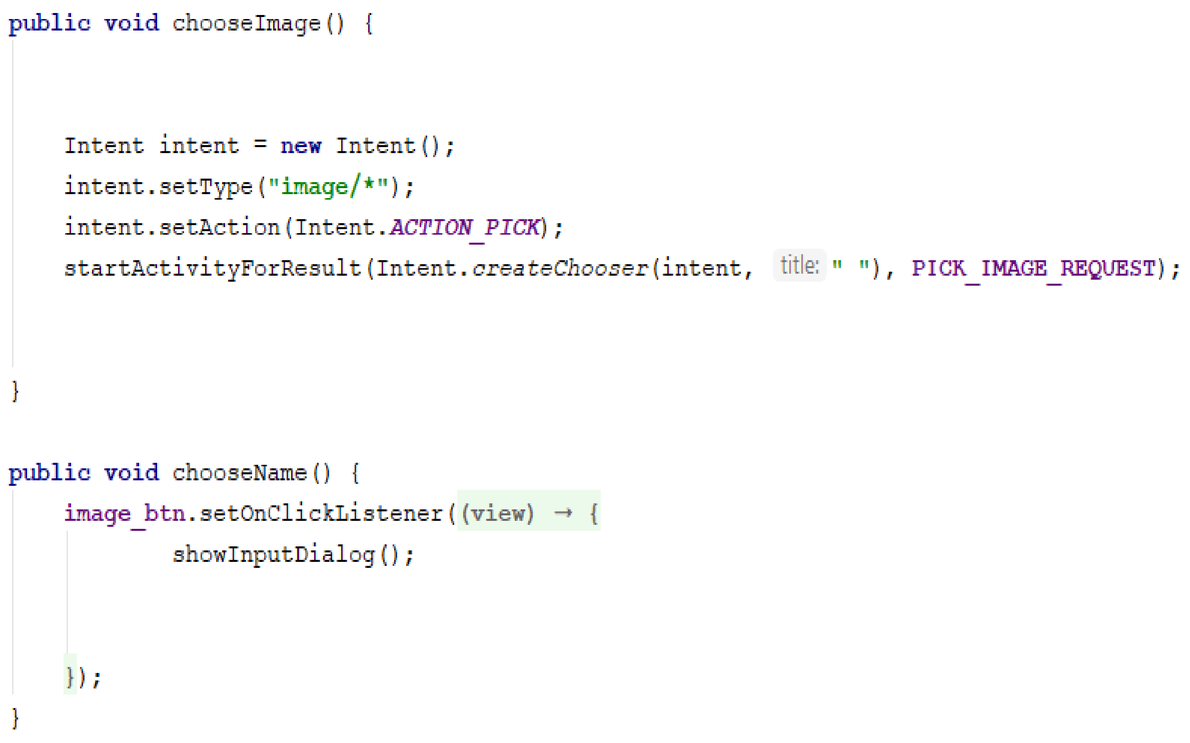
```
//Adapter For Contacts Activity
private String parm;
private ChildEventListener eventListener;
List<Contact> filterList;
List<Contact> newList;
ContactsAdapter.CustomFilter2 filter;

//Constructor
public ContactsAdapter(Context context, int resource, List<Contact> objects) {
    super(context, resource, objects);
    attachDatabaseReadListener();
    filterList = objects;
    newList = objects;

}
```

## ***Files***

This is done using the same principles as the image upload for the CV. Once uploaded the Employee can see their upload’s in a list view with the Title of what they have uploaded e.g. Safe Pass

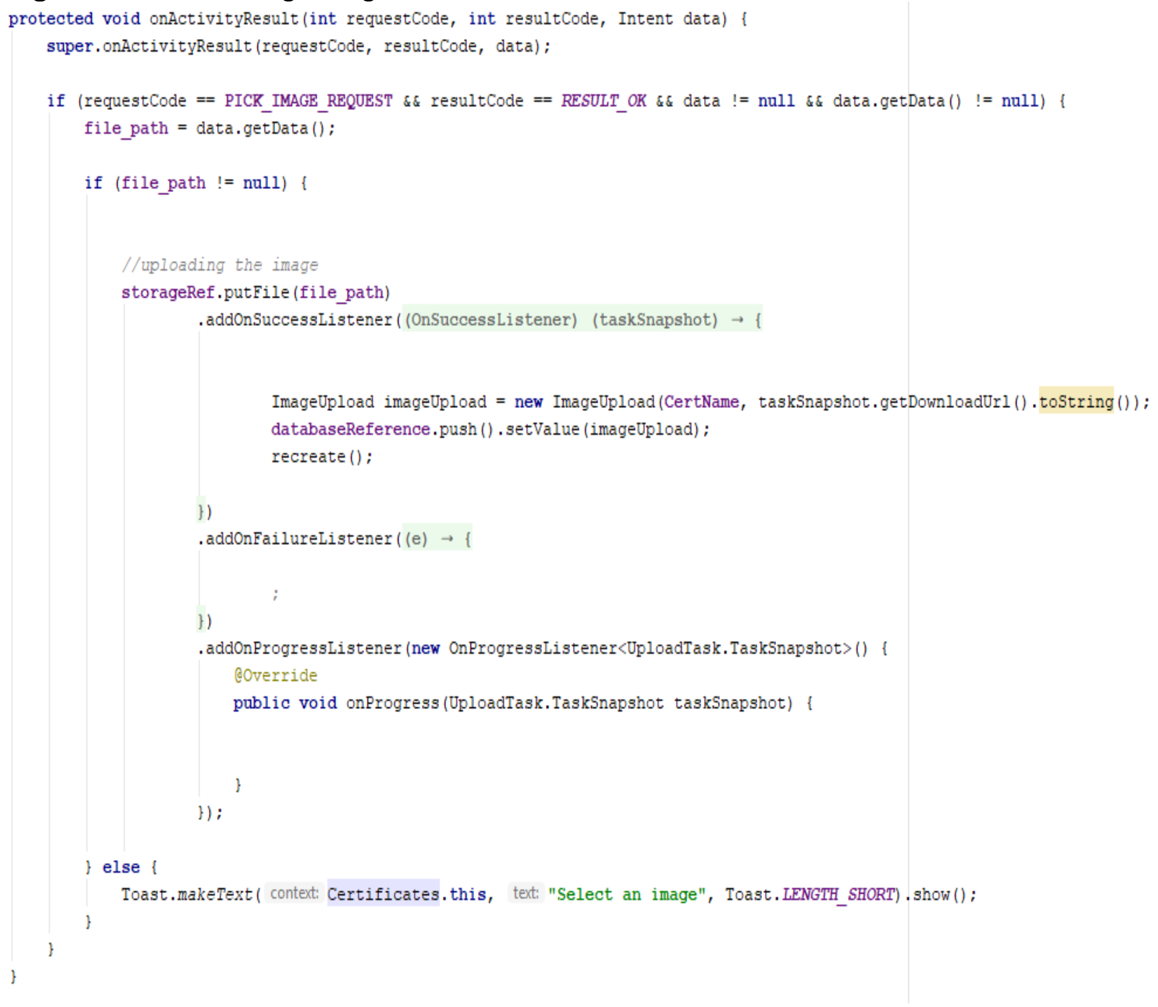
The Files are uploaded to Firebase’s storages service with the link also uploaded with an object also consisting of the files name. The List Adapter listens for an event change which one triggered will download all the files for that specific user into the list adapter.

**Figure 15** - Uploading Files:



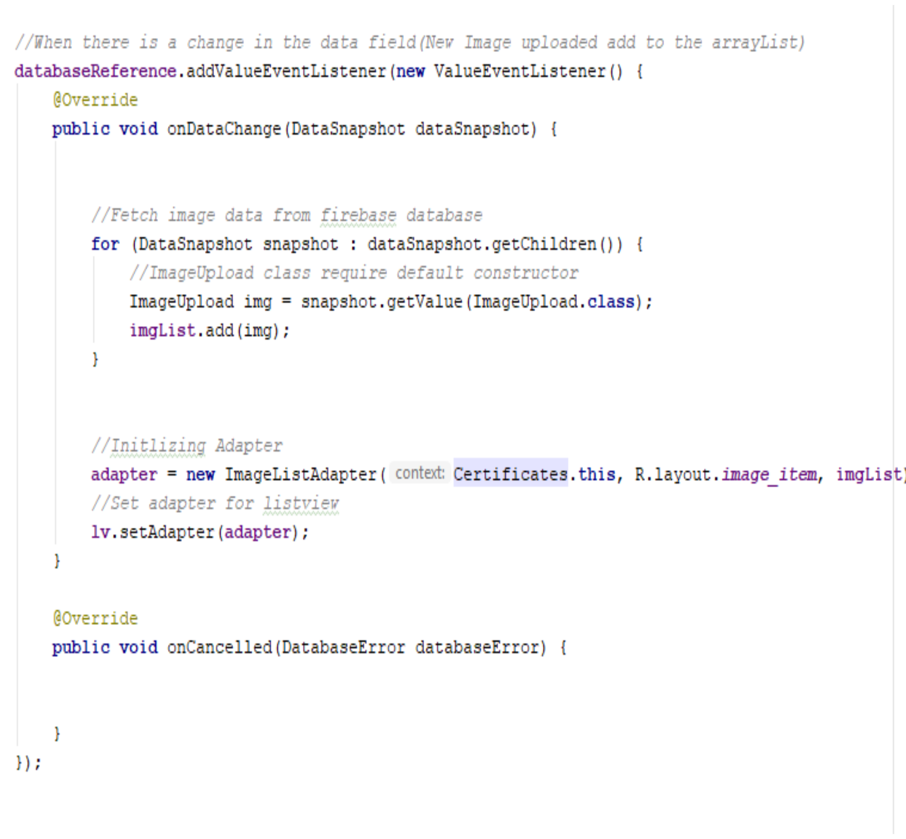
```
public void chooseImage() {


    Intent intent = new Intent();
    intent.setType("image/*");
    intent.setAction(Intent.ACTION_PICK);
    startActivityForResult(Intent.createChooser(intent, title: " "), PICK_IMAGE_REQUEST);


}

public void chooseName() {
    image_btn.setOnClickListener((view) → {
            showInputDialog();


    });
}
```

**Figure 16** – Sending Image to database.



```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    if (requestCode == PICK_IMAGE_REQUEST && resultCode == RESULT_OK && data != null && data.getData() != null) {
        file_path = data.getData();

        if (file_path != null) {


            //uploading the image
            storageRef.putFile(file_path)
                    .addOnSuccessListener((OnSuccessListener) (taskSnapshot) → {


                            ImageUpload imageUpload = new ImageUpload(CertName, taskSnapshot.getDownloadUrl().toString());
                            databaseReference.push().setValue(imageUpload);
                            recreate();

                    })
                    .addOnFailureListener((e) → {

                            ;
                    })
                    .addOnProgressListener(new OnProgressListener<UploadTask.TaskSnapshot>() {
                        @Override
                        public void onProgress(UploadTask.TaskSnapshot taskSnapshot) {


                        }
                    });

        } else {
            Toast.makeText( context: Certificates.this,  text: "Select an image", Toast.LENGTH_SHORT).show();
        }
    }
}
```

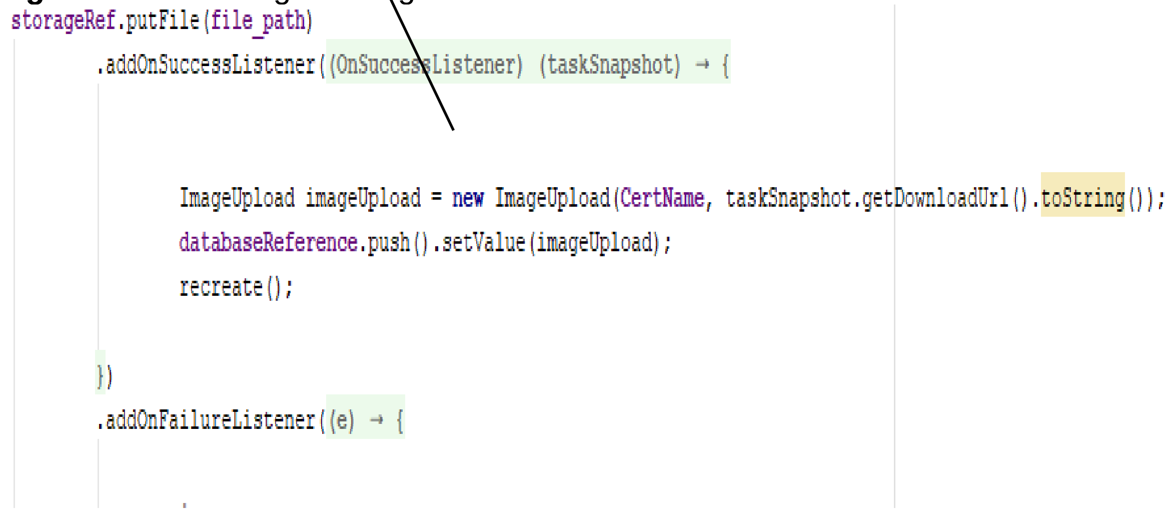
**Figure 17** - Listening for changes in firebase, and inserting into custom adapter:



```
//When there is a change in the data field(New Image uploaded add to the arrayList)
databaseReference.addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {


        //Fetch image data from firebase database
        for (DataSnapshot snapshot : dataSnapshot.getChildren()) {
            //ImageUpload class require default constructor
            ImageUpload img = snapshot.getValue(ImageUpload.class);
            imgList.add(img);
        }


        //Initlizing Adapter
        adapter = new ImageListAdapter( context: Certificates.this, R.layout.image_item, imgList);
        //Set adapter for listview
        lv.setAdapter(adapter);
    }

    @Override
    public void onCancelled(DatabaseError databaseError) {


    }
});
```

The files are stored in firebase storage where its link is stored in the database, so the listener is watching out for a change in the firebase database and when signalled will reference the file in storage.
Therefore, there are two insertions in firebase, Inserting into the database and storage.

**Figure 18 –** Pushing an Image to Firebase.



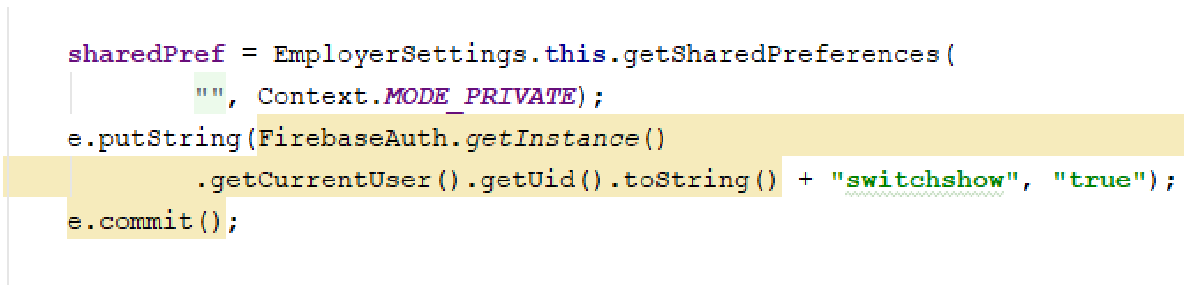
```
storageRef.putFile(file_path)
        .addOnSuccessListener((OnSuccessListener) (taskSnapshot) → {


                ImageUpload imageUpload = new ImageUpload(CertName, taskSnapshot.getDownloadUrl().toString());
                databaseReference.push().setValue(imageUpload);
                recreate();


        })
        .addOnFailureListener((e) → {
```

## ***Settings***

The Settings include the Maximum range that the Employees are in Scope, which can be easily adjusted by the user using the Progress bar. Notifications can also be triggered to be turned off or on and the Employee can choose to have their Profile visible on the map using a switch. Finally, the Employee has the chance to Logout of the application by clicking the Logout button.
To make the settings constant and not to be always relying on network activity I implemented their state by using Shared Preferences. Shared Preference’s object points to a file containing key-value pairs and provide methods to read and write to them.  These values will always be available to the Employee using when using the applications and the data will be received instantly.

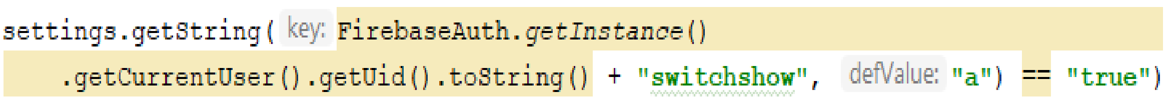
However, for the distance to be correct the Employees distance is saved to the database as using shared preference would not be able to retrieve the distance for the Employer to Judge.

**Figure 19** - Establishing shared Preferences and Inserting.



```
sharedPref = EmployerSettings.this.getSharedPreferences(
        "", Context.MODE_PRIVATE);
e.putString(FirebaseAuth.getInstance()
        .getCurrentUser().getUid().toString() + "switchshow", "true");
e.commit();
```

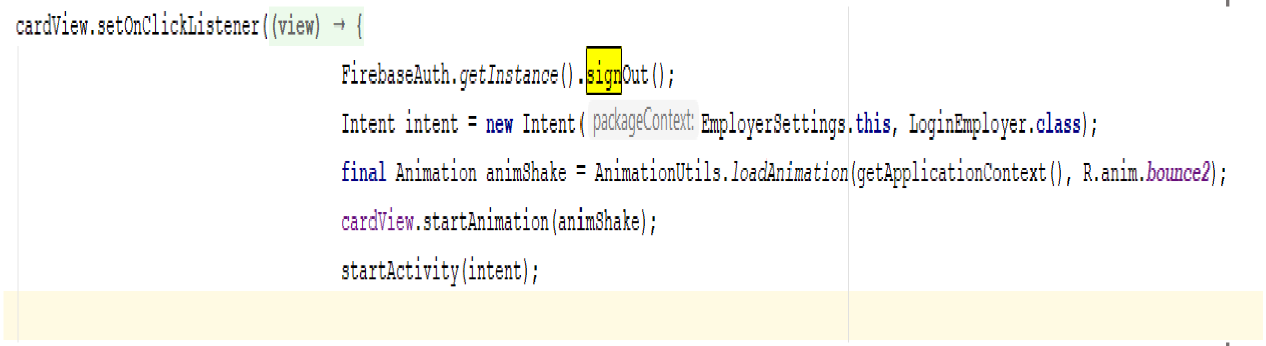
**Figure 20** - Retrieving from shared Preferences.



```
settings.getString( key: FirebaseAuth.getInstance()
    .getCurrentUser().getUid().toString() + "switchshow",  defValue: "a") == "true")
```

When the logout button is clicked, the current user is established, and their session is cancelled where they will be given the opportunity to sign in again.

**Figure 21 - Logout**



```
cardView.setOnClickListener((view) → {
                    FirebaseAuth.getInstance().signOut();
                    Intent intent = new Intent( packageContext: EmployerSettings.this, LoginEmployer.class);
                    final Animation animShake = AnimationUtils.loadAnimation(getApplicationContext(), R.anim.bounce2);
                    cardView.startAnimation(animShake);
                    startActivity(intent);
```

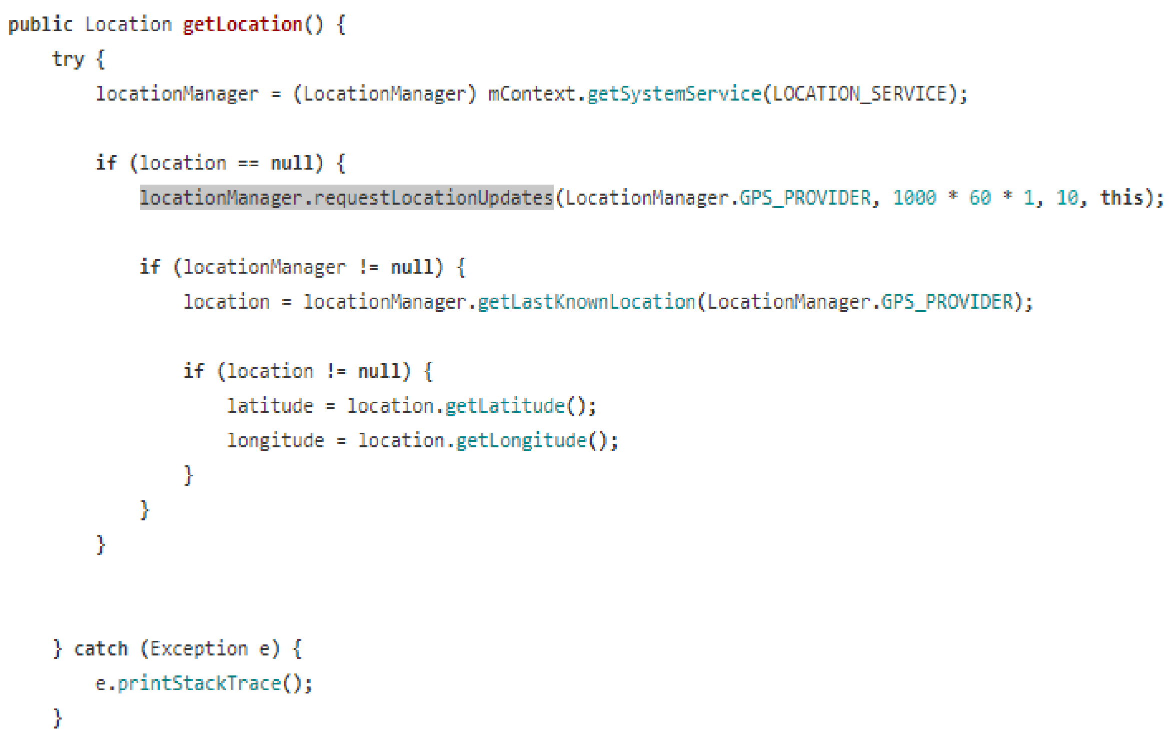
## ***GPS Tracker:***

The GPSTracker class is focused on receiving the current location of the device when called.

The location will constantly be changing as the mobile will be moving around therefore I had to make sure I was receiving location updates. To achieve the change in location I had to implement the Location Listener interface this interface is used for receiving notifications when the location has changed. Therefore, I had to extend service and implement the location interface.

The Location Manger class provides access to the systems location services, these services allow applications to obtain periodic updates of the devices geographical location. When the class is called the last known location to the device is received.

**Figure 22 – G**et Current Location

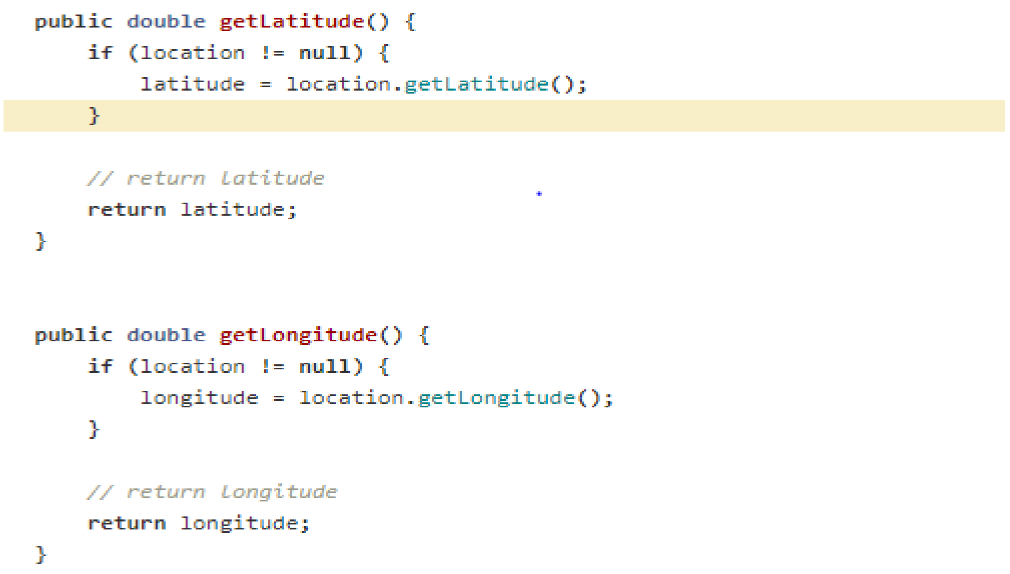


```
public Location getLocation() {
    try {
        locationManager = (LocationManager) mContext.getSystemService(LOCATION_SERVICE);

        if (location == null) {
            locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 1000 * 60 * 1, 10, this);

            if (locationManager != null) {
                location = locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);

                if (location != null) {
                    latitude = location.getLatitude();
                    longitude = location.getLongitude();
                }
            }
        }


    } catch (Exception e) {
        e.printStackTrace();
    }
```

The location is updated by registering for location updates using the Location Manager every 60 seconds and every 10 meters. Once the location is established the last previous know location to the device is received and is used as the current location where latitude and longitude values are pulled from it.  The last location is received by calling the function getLastKnowLocation(String provider), this returns a location indicating the data from the last known location fix obtained from the Location Manager.

Once the location is established separate customized methods were created to get the latitude and longitude values from the current location.

**Figure 23 –** Get Latitude and longitude values.

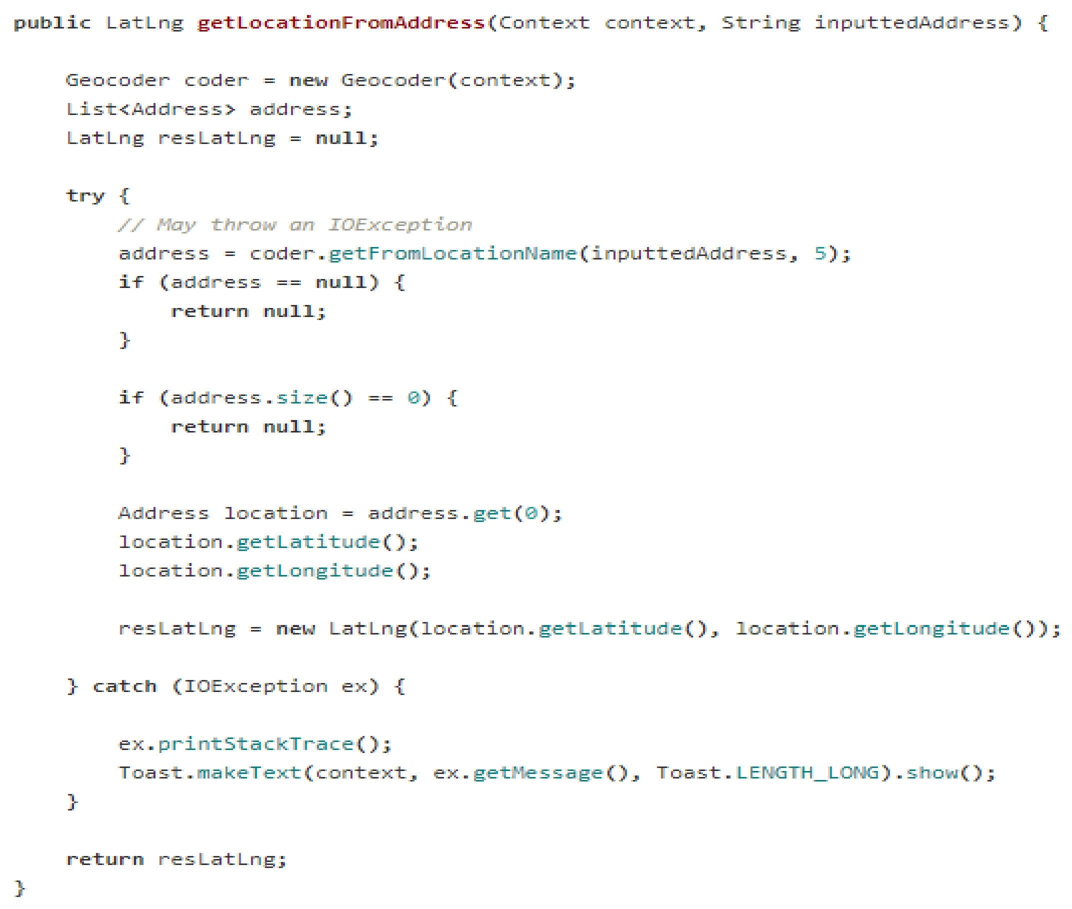


```
public double getLatitude() {
    if (location != null) {
        latitude = location.getLatitude();
    }

    // return latitude
    return latitude;
}


public double getLongitude() {
    if (location != null) {
        longitude = location.getLongitude();
    }

    // return longitude
    return longitude;
}
```

## ***Geocoding***

To translate an address into valid location values I implement the use of Geocoding.
Geocoding is the process of transforming a street address or other description of a location into a (latitude, longitude) coordinate. Here is the function I created to translate an address into valid latitude and longitude coordinates.

**Figure 24 –** Geocoding Address.



```
public LatLng getLocationFromAddress(Context context, String inputtedAddress) {

    Geocoder coder = new Geocoder(context);
    List<Address> address;
    LatLng resLatLng = null;

    try {
        // May throw an IOException
        address = coder.getFromLocationName(inputtedAddress, 5);
        if (address == null) {
            return null;
        }

        if (address.size() == 0) {
            return null;
        }

        Address location = address.get(0);
        location.getLatitude();
        location.getLongitude();

        resLatLng = new LatLng(location.getLatitude(), location.getLongitude());

    } catch (IOException ex) {

        ex.printStackTrace();
        Toast.makeText(context, ex.getMessage(), Toast.LENGTH_LONG).show();
    }

    return resLatLng;
}
```

- ***<u>Google Maps API displaying images according to their location.</u>***

  When the Employer has chosen their field of in search of Employment, there chosen field is passed into the google maps Activity. e.g(Plumbing). All the Employees profiles that match the job reference are filtered into an Array List using a Query. Once the jobs are established into the Array more filters are applied to meet with the Employers specific search accounting their settings.

  The Distances are filtered using the Employees constant latitude and longitude vales in comparison to the Employers Current Latitude and Longitude values. To retrieve the correct distance between the I used the Haversine formula. The haversine formula determines the great circle distance between two points on a sphere given their latitude and longitude values. I implemented a function that took in the Employees latitude and longitude parameters as parameters and returned the distance, if the returned distance was satisfied the application would show the Employees Profile on the map.

  Each Employee on the Map is represented by an Emoji represented by the gender of the Employee. The Emojis are bit images that are transformed onto the map as a Bit-Map displayed as a marker. Each marker holds the Employees name and the Employees unique ID individually. Therefore, that when the markers are clicked their unique ID is passed to the Profile Activity where the current profile of that Employee will be displayed.

  When the application is filtering Employees based on their distance it uses a haversine algorithm to calculate the distance between the Employers location and Employees Location. The haversine formula determines the great-circle distance between two points on a sphere given their longitudes and latitudes.

**Figure 25 –** Great circle distance.

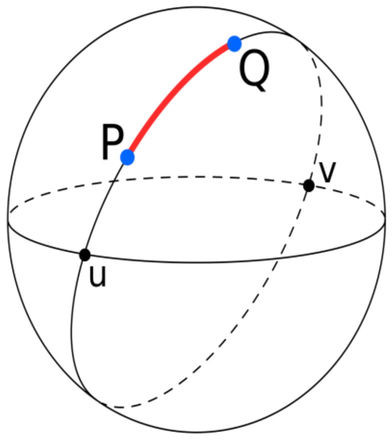
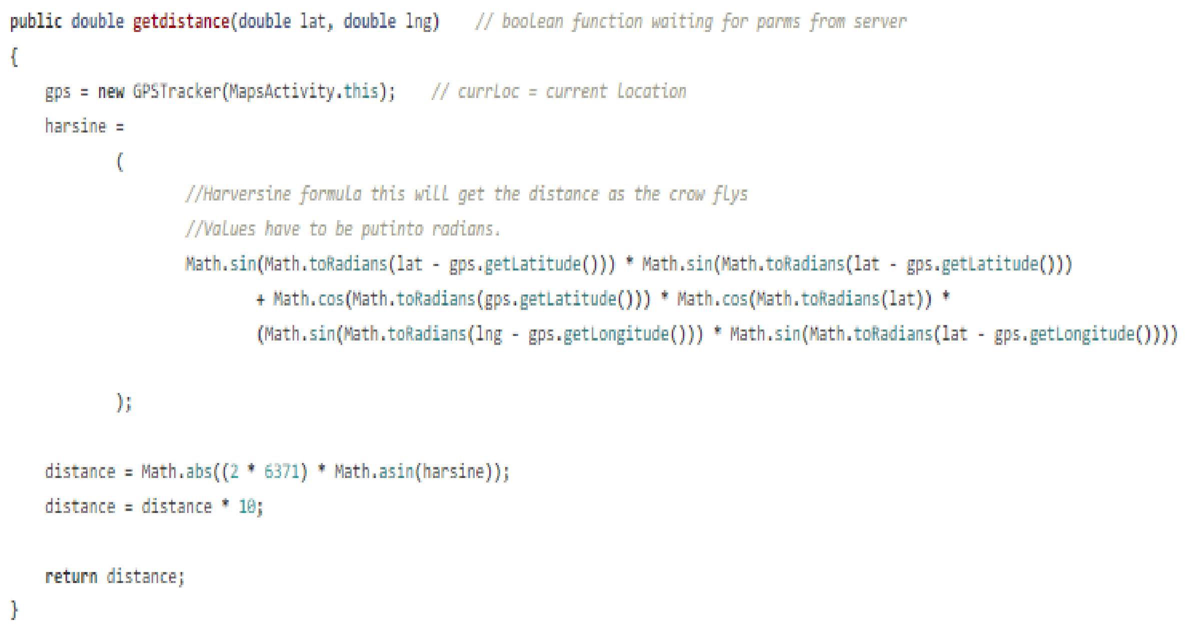
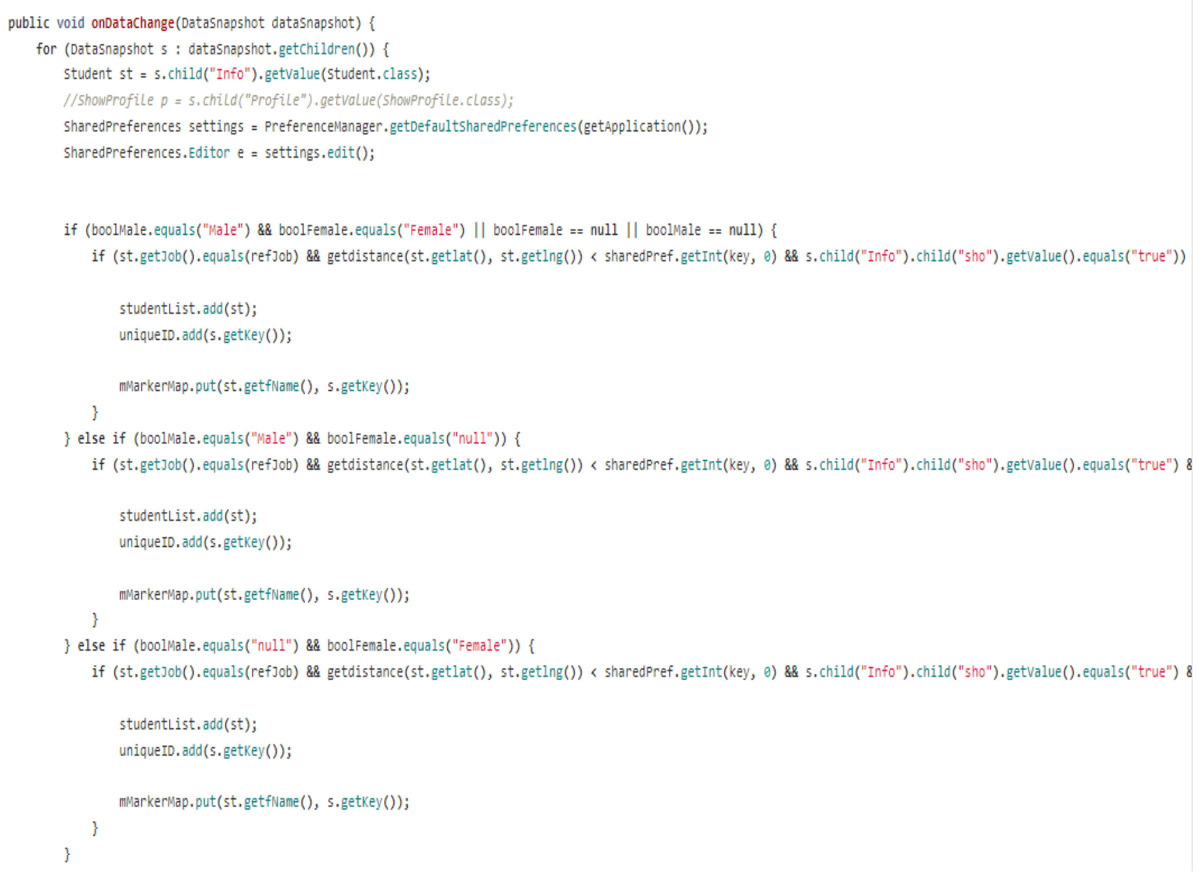


**Figure 26 –** Haversine Formula.



```
public double getdistance(double lat, double lng)    // boolean function waiting for parms from server
{
    gps = new GPSTracker(MapsActivity.this);    // currLoc = current location
    harsine =
            (
                    //Harversine formula this will get the distance as the crow flys
                    //Values have to be putinto radians.
                    Math.sin(Math.toRadians(lat - gps.getLatitude())) * Math.sin(Math.toRadians(lat - gps.getLatitude()))
                            + Math.cos(Math.toRadians(gps.getLatitude())) * Math.cos(Math.toRadians(lat)) *
                            (Math.sin(Math.toRadians(lng - gps.getLongitude())) * Math.sin(Math.toRadians(lat - gps.getLongitude())))

            );

    distance = Math.abs((2 * 6371) * Math.asin(harsine));
    distance = distance * 10;

    return distance;
}
```

**Figure 27** - Filtering through Employees and displaying them on the map interface.



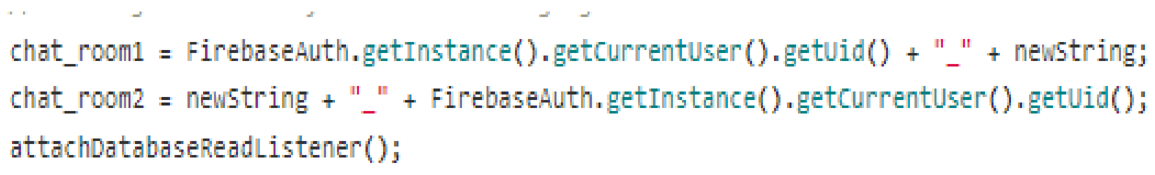
```
public void onDataChange(DataSnapshot dataSnapshot) {
    for (DataSnapshot s : dataSnapshot.getChildren()) {
        Student st = s.child("Info").getValue(Student.class);
        //ShowProfile p = s.child("Profile").getValue(ShowProfile.class);
        SharedPreferences settings = PreferenceManager.getDefaultSharedPreferences(getApplication());
        SharedPreferences.Editor e = settings.edit();


        if (boolMale.equals("Male") && boolFemale.equals("Female") || boolFemale == null || boolMale == null) {
            if (st.getJob().equals(refJob) && getdistance(st.getlat(), st.getlng()) < sharedPref.getInt(key, 0) && s.child("Info").child("sho").getValue().equals("true"))

                studentList.add(st);
                uniqueID.add(s.getKey());

                mMarkerMap.put(st.getfName(), s.getKey());
            }
        } else if (boolMale.equals("Male") && boolFemale.equals("null")) {
            if (st.getJob().equals(refJob) && getdistance(st.getlat(), st.getlng()) < sharedPref.getInt(key, 0) && s.child("Info").child("sho").getValue().equals("true") &

                studentList.add(st);
                uniqueID.add(s.getKey());

                mMarkerMap.put(st.getfName(), s.getKey());
            }
        } else if (boolMale.equals("null") && boolFemale.equals("Female")) {
            if (st.getJob().equals(refJob) && getdistance(st.getlat(), st.getlng()) < sharedPref.getInt(key, 0) && s.child("Info").child("sho").getValue().equals("true") &

                studentList.add(st);
                uniqueID.add(s.getKey());

                mMarkerMap.put(st.getfName(), s.getKey());
            }
        }
```

- *Instant Messaging.*

  In the application instant messaging is how the Employer and the Employee communicate. This was established using firebase real time database. The messaging activity uses adapters and to hold the different object messages of the sender and the receiver. The receiver is the current user of the messaging activity at a certain point. This differentiates the layout of the messages; sender's objects are displayed on the right while the receiver's objects are displayed on the left.

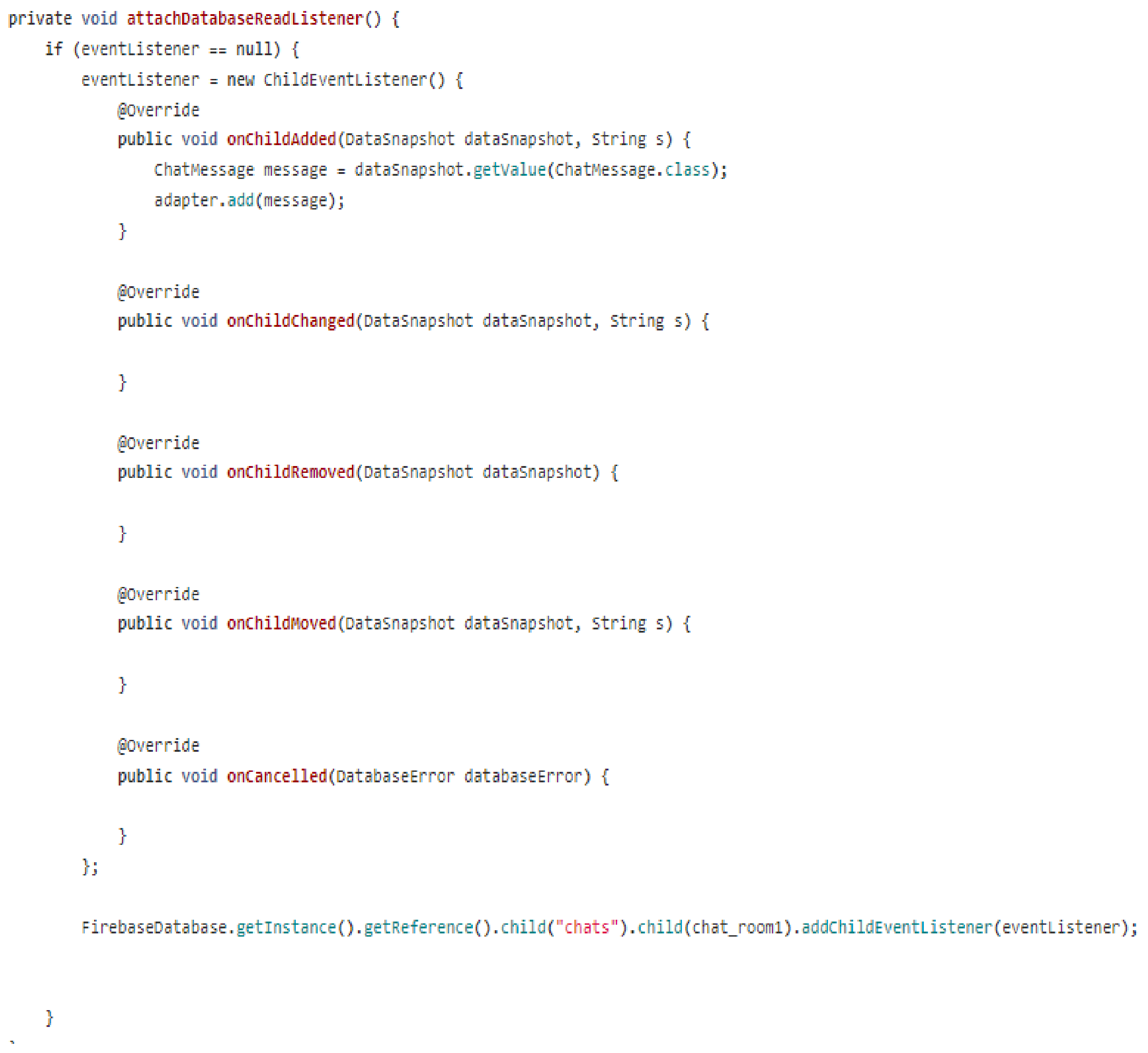
  Each Conversation has a unique field in the database. When the Employer sends the message to the Employee a chat room is created named by the sender's ID and the receivers ID concatenated. This is labelled as chat room 1, when the Employee replies with a message another chat room is created with the same data as chat room1 which is labelled as chat room 2 named by the receiver's ID and the senders ID concatenated. (note opposite naming than chat room1). This allows both the Employer and the Employee to access the same conversation. Every time there is a message update in the firebase the listener signals the application and the new object is added to the adapter.

**Figure 28** - The two Chat Rooms



```
chat_room1 = FirebaseAuth.getInstance().getCurrentUser().getUid() + "_" + newString;
chat_room2 = newString + "_" + FirebaseAuth.getInstance().getCurrentUser().getUid();
attachDatabaseReadListener();
```

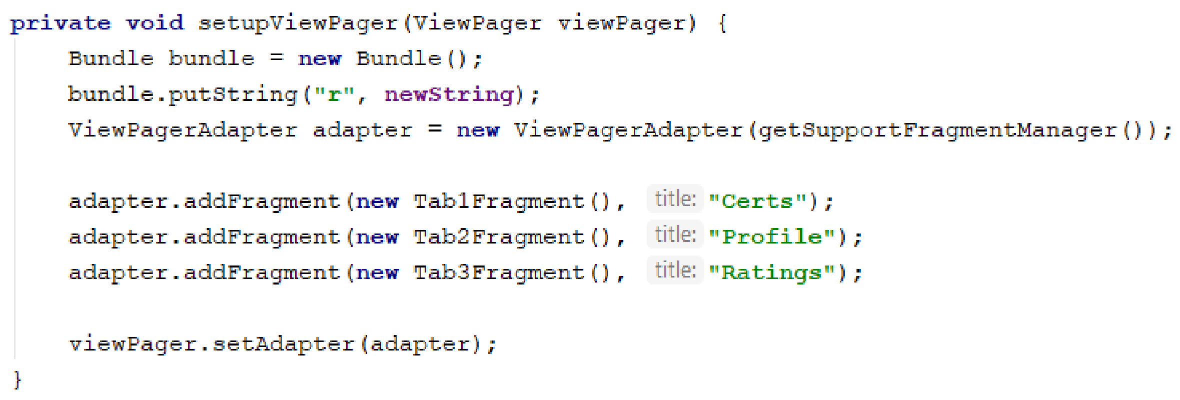
**Figure 29** - Listening for updates on messages:



```
private void attachDatabaseReadListener() {
    if (eventListener == null) {
        eventListener = new ChildEventListener() {
            @Override
            public void onChildAdded(DataSnapshot dataSnapshot, String s) {
                ChatMessage message = dataSnapshot.getValue(ChatMessage.class);
                adapter.add(message);
            }

            @Override
            public void onChildChanged(DataSnapshot dataSnapshot, String s) {

            }

            @Override
            public void onChildRemoved(DataSnapshot dataSnapshot) {

            }

            @Override
            public void onChildMoved(DataSnapshot dataSnapshot, String s) {

            }

            @Override
            public void onCancelled(DatabaseError databaseError) {

            }
        };

        FirebaseDatabase.getInstance().getReference().child("chats").child(chat_room1).addChildEventListener(eventListener);


    }
```

- ***Profile.***
  The Employee is displayed to the user using the Profile Activity which consists of the Employees Profile image and options to view profile content, ratings and Certificates (Employees achievements). These three options are displayed using a View Pager, which allows the user to flip left and right through pages of data. Each database is implemented using a Pager Adapter to generate the current pages the view shows. These fragments (data pages) are used to display and advertise the data that than Employee has uploaded and ratings from other Employers.

**Figure 30 –** View Pager



```
private void setupViewPager(ViewPager viewPager) {
    Bundle bundle = new Bundle();
    bundle.putString("r", newString);
    ViewPagerAdapter adapter = new ViewPagerAdapter(getSupportFragmentManager());

    adapter.addFragment(new Tab1Fragment(),  title: "Certs");
    adapter.addFragment(new Tab2Fragment(),  title: "Profile");
    adapter.addFragment(new Tab3Fragment(),  title: "Ratings");

    viewPager.setAdapter(adapter);
}
```

1. ***Fragment 1(Certificates)***
   Fragment 1 consists of the all the Employees Certificates that are displayed using a list view of objects representing an image and the image name. These values are received from firebase storage and database repository.

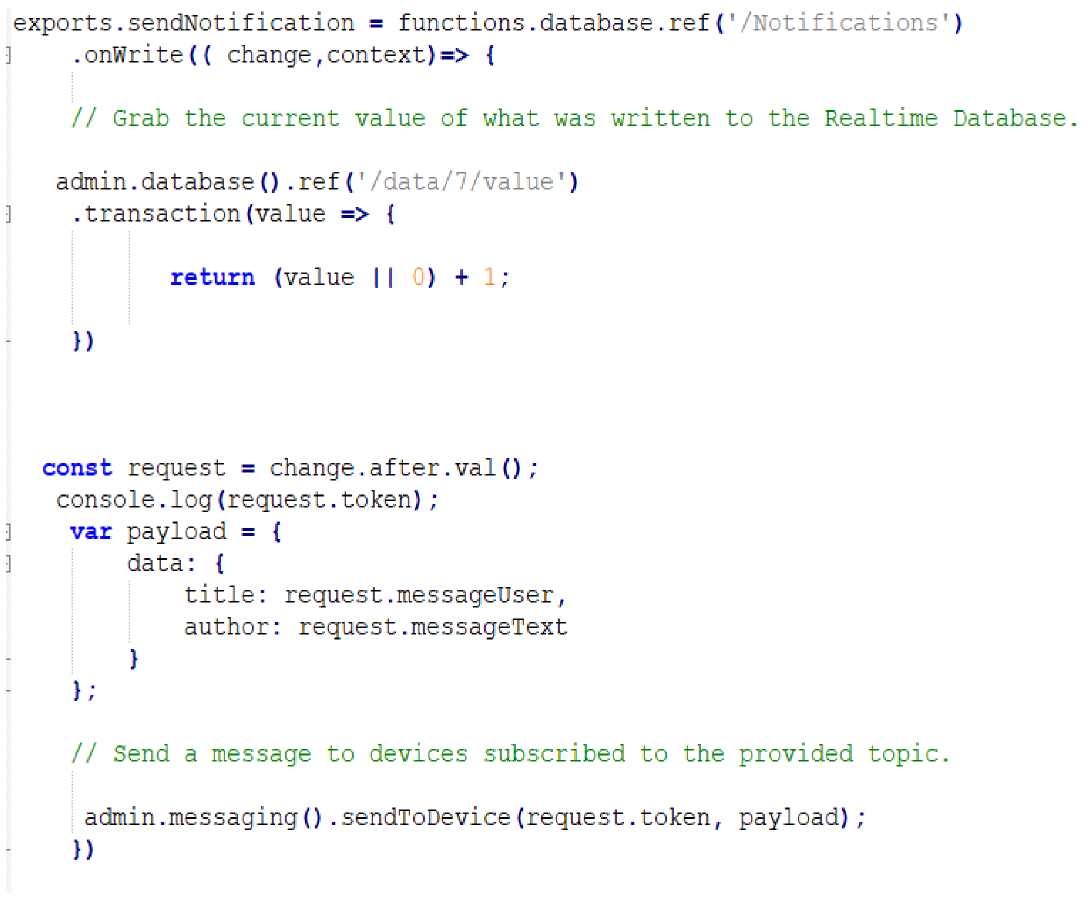
2. ***Fragment 2(CV)***
   Fragment 2 displays the Employees CV, this data is received from the database and formatted correctly to display what the Employee has to offer to the Employer.

3. ***Fragment 3(Ratings)***
   Fragment 3 displays all the ratings of the Employee of previous work completed. These are displayed as a List View of objects consisting of a rating bar, name of the Employer submitted, and the comment created.
   The Employer has also the chance to provide the Employee with a rating by simply clicking on the rating bar where a pop-up dialog will be displayed to enter the comment for the Employee.

- ***Push Notifications***

  Once a message is received by the application the user is notified through a push notification. The push notifications were created through FCM where each account that the application was used in there is a saved unique token ID of that Device.
  Each user of the application has a hashed value of their token according to their unique ID. The push notifications were implemented through node.js functions which sit on the firebase Server waiting for a change in the notifications field. Every time a message is sent an instance of that message is sent to the Notifications field in firebase. Once the function has seen a change in the database table a push notification is sent to the receiver containing partial content of the message. Once the application receives the notification it handles it and displays it to the user's device, regardless if the application is still running in the back ground or not.

**Figure 31** - Push notification function sitting on firebase server.



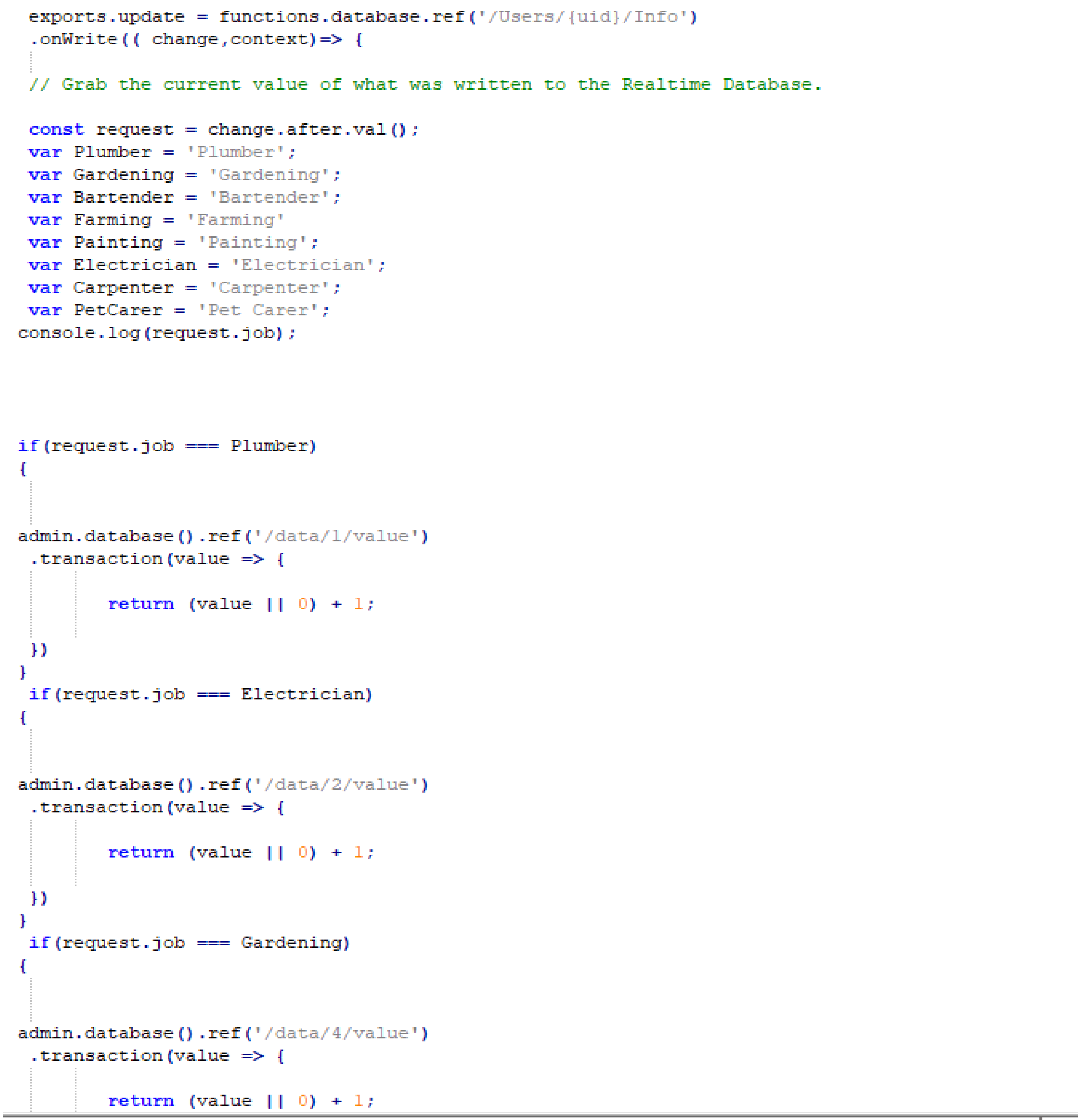
```
exports.sendNotification = functions.database.ref('/Notifications')
    .onWrite(( change,context)=> {

    // Grab the current value of what was written to the Realtime Database.

   admin.database().ref('/data/7/value')
    .transaction(value => {

           return (value || 0) + 1;

    })



  const request = change.after.val();
   console.log(request.token);
    var payload = {
        data: {
            title: request.messageUser,
            author: request.messageText
        }
    };

    // Send a message to devices subscribed to the provided topic.

     admin.messaging().sendToDevice(request.token, payload);
    })
```

- ***Web Application***

The application was implemented using HTML, CSS, JavaScript and Node.js. The UI of the web application was implemented using Bootstrap which is an open source toolkit for developing HTML, CSS and JS. The graphs in the application using fusion charts which is also an open source platform for creating charts.

When a user is added to the database there is a node.js function sitting on the database that updates the web application fields in the database, this ensures that the data is real time data and updating continuously.

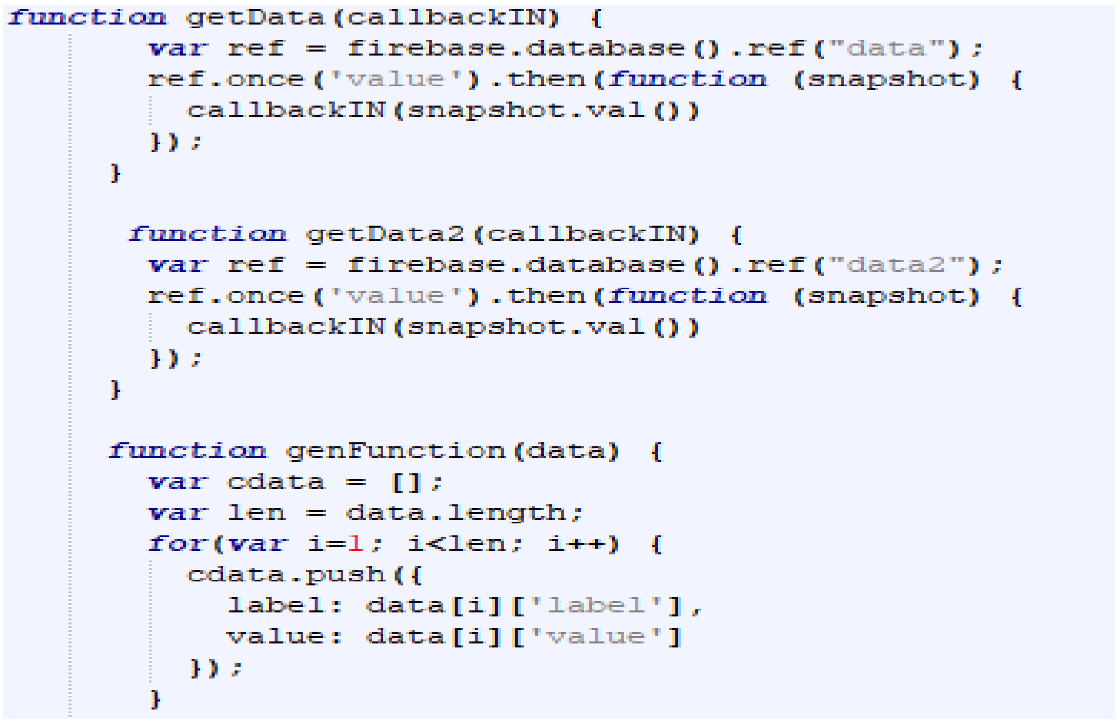
**Figure 32** – Update Employment



```
exports.update = functions.database.ref('/Users/{uid}/Info')
.onWrite(( change,context)=> {

// Grab the current value of what was written to the Realtime Database.

const request = change.after.val();
var Plumber = 'Plumber';
var Gardening = 'Gardening';
var Bartender = 'Bartender';
var Farming = 'Farming'
var Painting = 'Painting';
var Electrician = 'Electrician';
var Carpenter = 'Carpenter';
var PetCarer = 'Pet Carer';
console.log(request.job);



if(request.job === Plumber)
{

admin.database().ref('/data/1/value')
.transaction(value => {

        return (value || 0) + 1;

})
}
if(request.job === Electrician)
{

admin.database().ref('/data/2/value')
.transaction(value => {

        return (value || 0) + 1;

})
}
if(request.job === Gardening)
{

admin.database().ref('/data/4/value')
.transaction(value => {

        return (value || 0) + 1;
```

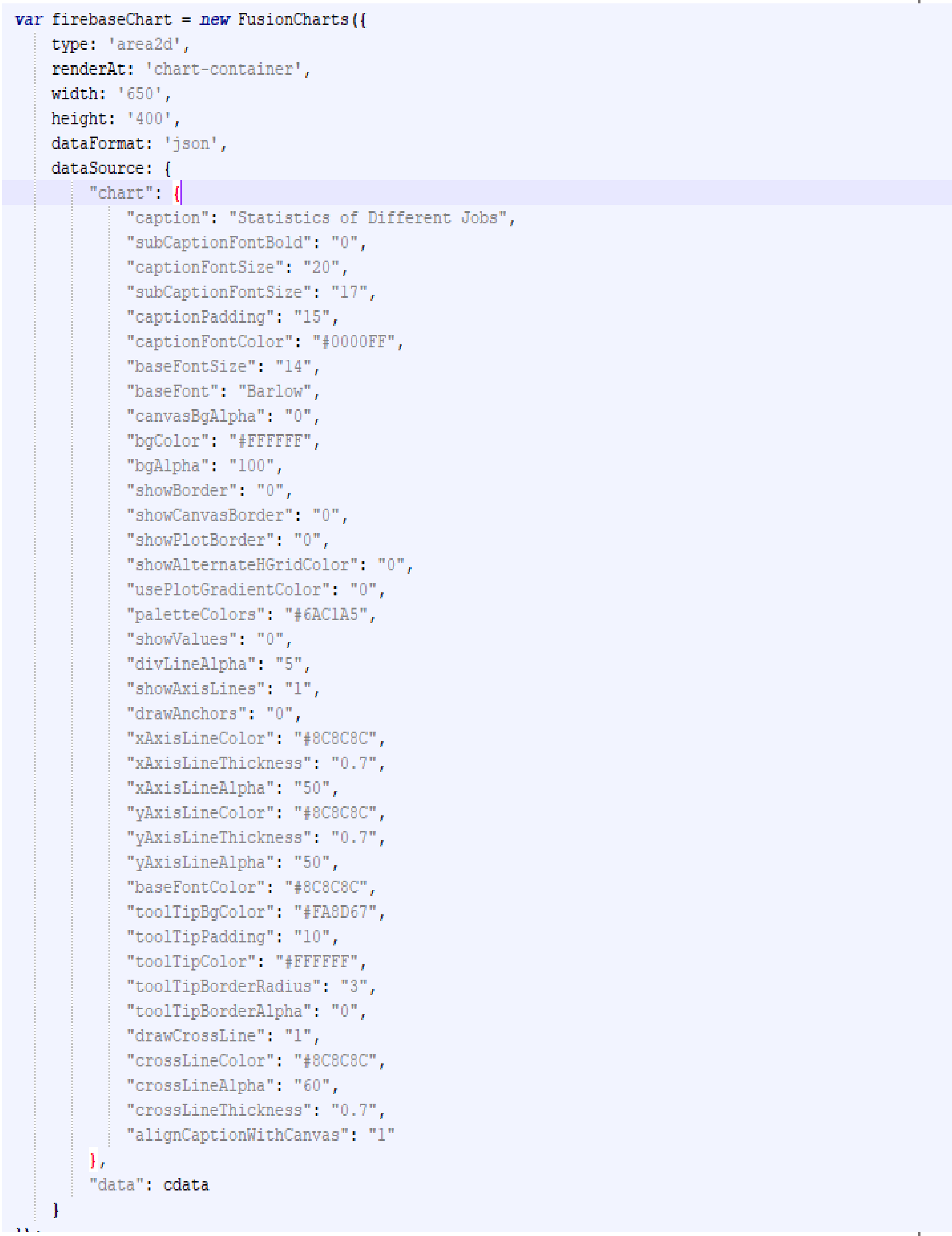
.

**Figure 33 -** Connecting fusion charts with firebase



```
function getData(callbackIN) {
       var ref = firebase.database().ref("data");
       ref.once('value').then(function (snapshot) {
         callbackIN(snapshot.val())
       });
     }

      function getData2(callbackIN) {
       var ref = firebase.database().ref("data2");
       ref.once('value').then(function (snapshot) {
         callbackIN(snapshot.val())
       });
     }

     function genFunction(data) {
       var cdata = [];
       var len = data.length;
       for(var i=1; i<len; i++) {
         cdata.push({
           label: data[i]['label'],
           value: data[i]['value']
         });
       }
```

**Figure 34** - Displaying fusion charts with firebase:



```
var firebaseChart = new FusionCharts({
    type: 'area2d',
    renderAt: 'chart-container',
    width: '650',
    height: '400',
    dataFormat: 'json',
    dataSource: {
        "chart": {
            "caption": "Statistics of Different Jobs",
            "subCaptionFontBold": "0",
            "captionFontSize": "20",
            "subCaptionFontSize": "17",
            "captionPadding": "15",
            "captionFontColor": "#0000FF",
            "baseFontSize": "14",
            "baseFont": "Barlow",
            "canvasBgAlpha": "0",
            "bgColor": "#FFFFFF",
            "bgAlpha": "100",
            "showBorder": "0",
            "showCanvasBorder": "0",
            "showPlotBorder": "0",
            "showAlternateHGridColor": "0",
            "usePlotGradientColor": "0",
            "paletteColors": "#6AC1A5",
            "showValues": "0",
            "divLineAlpha": "5",
            "showAxisLines": "1",
            "drawAnchors": "0",
            "xAxisLineColor": "#8C8C8C",
            "xAxisLineThickness": "0.7",
            "xAxisLineAlpha": "50",
            "yAxisLineColor": "#8C8C8C",
            "yAxisLineThickness": "0.7",
            "yAxisLineAlpha": "50",
            "baseFontColor": "#8C8C8C",
            "toolTipBgColor": "#FA8D67",
            "toolTipPadding": "10",
            "toolTipColor": "#FFFFFF",
            "toolTipBorderRadius": "3",
            "toolTipBorderAlpha": "0",
            "drawCrossLine": "1",
            "crossLineColor": "#8C8C8C",
            "crossLineAlpha": "60",
            "crossLineThickness": "0.7",
            "alignCaptionWithCanvas": "1"
        },
        "data": cdata
    }
})
```

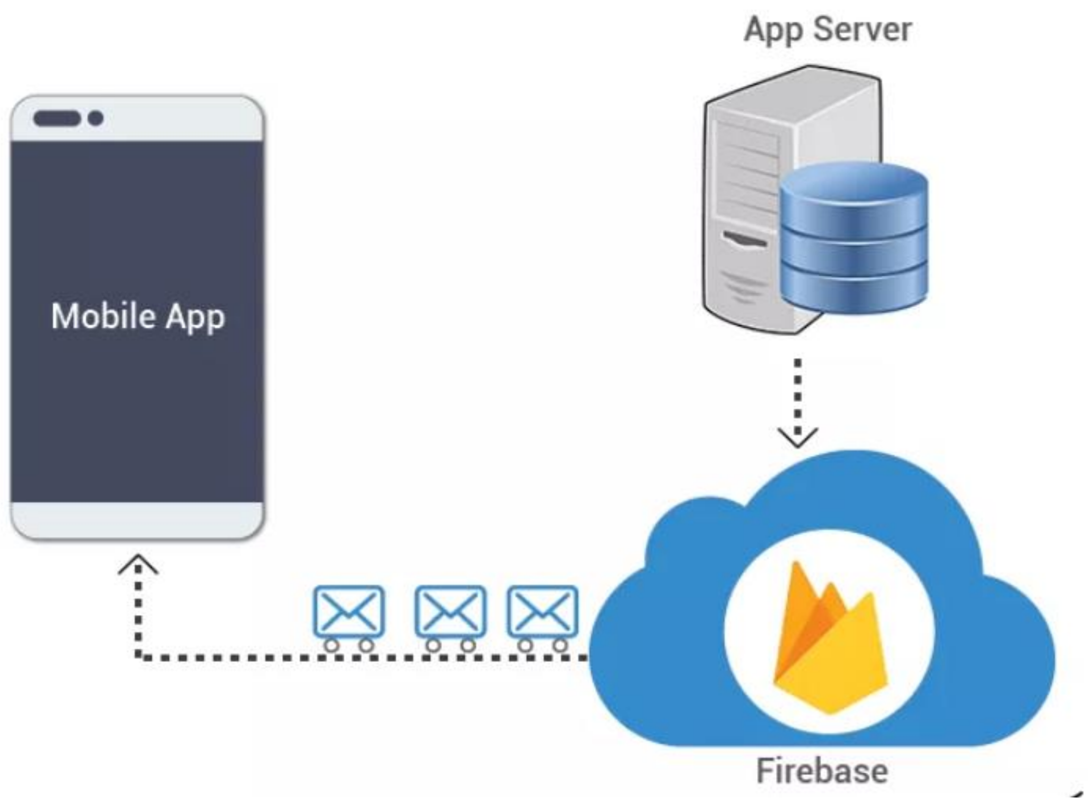
## 8). Problems Solved

### 1) Push Notification’s

When I started implementing push notifications, I was just using an event listener and manually sending the notifications manually through the application. This design was achieving its goal However it would not work when the application was not running in the background. Although creating a thread to listen in the background could be used it would be creating a lot of unnecessary work for the application.

To resolve this, I used functions in firebase, which was hosted live on the firebase server listening for changes in the database, when there was a change it would push the notification to the device corresponding to its token.

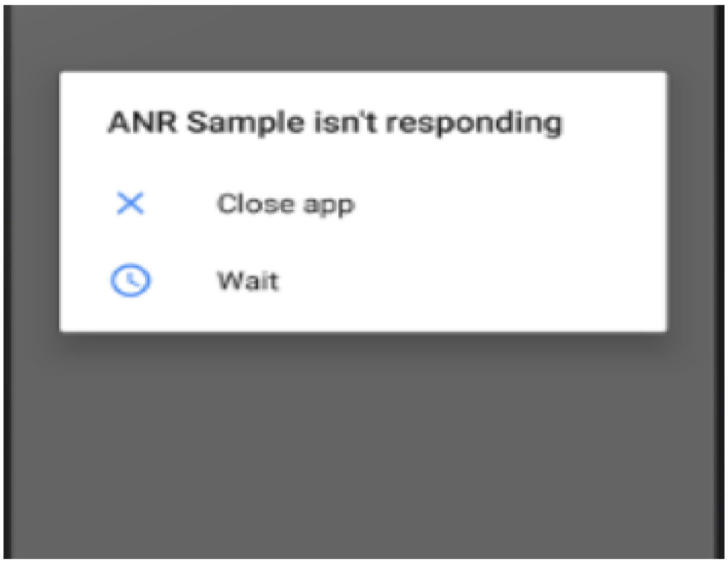
**Figure 35 –** Push Notification System.



## 2) **Application Not Responding (ANR).**

This was an encountered error that I was receiving throughout the development process of the application. This happens in the android application when the main UI thread of the app is blocked or there is too much processes trying to run. It was not until mid-way through the development process that I started to see this. The reason I received this error was when my personal phone which was a Huawei p8 Lite which had a Octa-core 1.2 GHz Cortex-A5 and 3 GB of ram which had more than enough power to run the application which broke. I then ran the application on an old Samsung galaxy although it was a powerful phone due to its age it was not able to handle all the process running on the main UI thread which caused the application to display an application not responding (ANR).

**Figure 36 -** Example of an ANR**.**


ANR Sample isn't responding

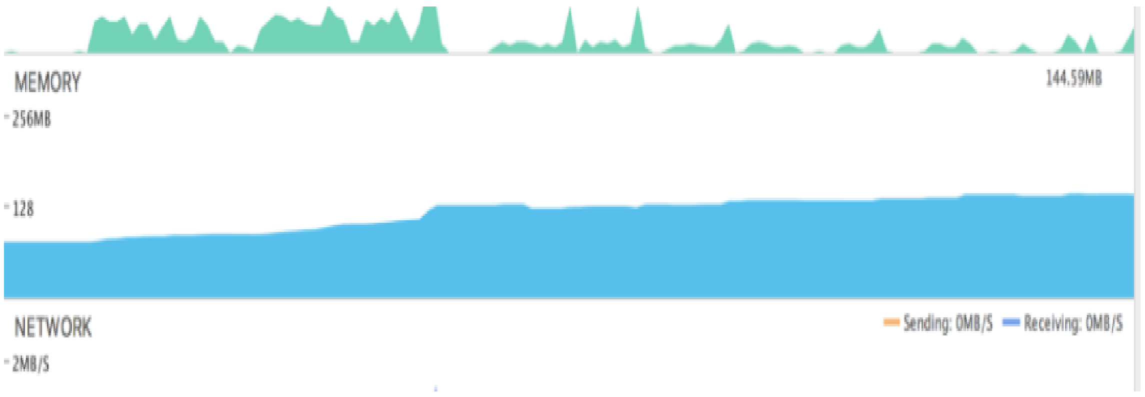
### **Resolution.**

The main areas that I was receiving ANR was when I was downloading data (especially images) and uploading files from the actual devices. This was because the application was trying to overcome several processes at a time. The most effective way to overcome this problem was a Worker Thread. The best way to create a worker thread is with AsyncTask class. This extends AsyncTask and implements the doInBackGround() method to perform the work. However by just using ONActivityResult() stopped the thread until the data was available which eliminated the ANR.

**3) Out Of Memory Error (OOM)**

The main reason for receiving OOM error is due to Memory Leaks. This happens when a program to release discarded memory causing impaired performance or Failure. The main cause of Memory Leaks is the Context of the object.  Every Application has a registered activity which can be established by getApplicationContext() here relevant information of that activity is stored.
Here is an example of the memory measure of the application when there is an OOM.

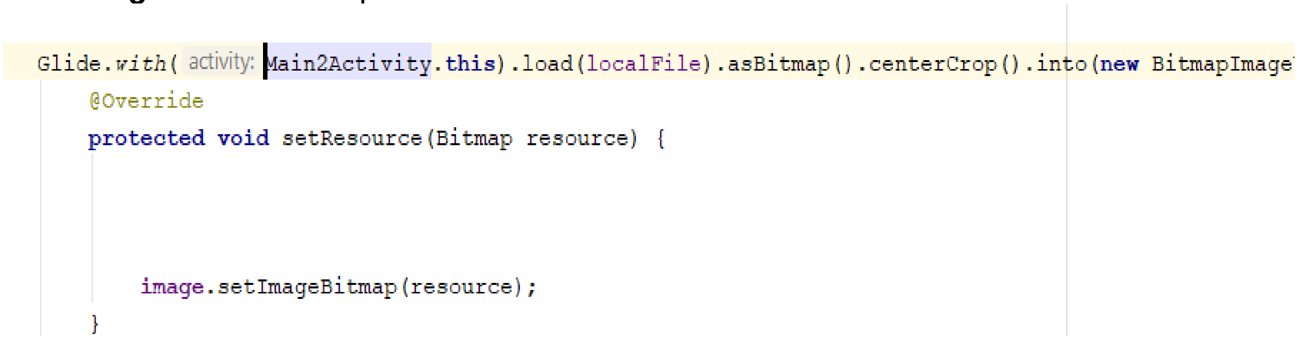
**Figure 37 –** Android Memory



**Resolution.**

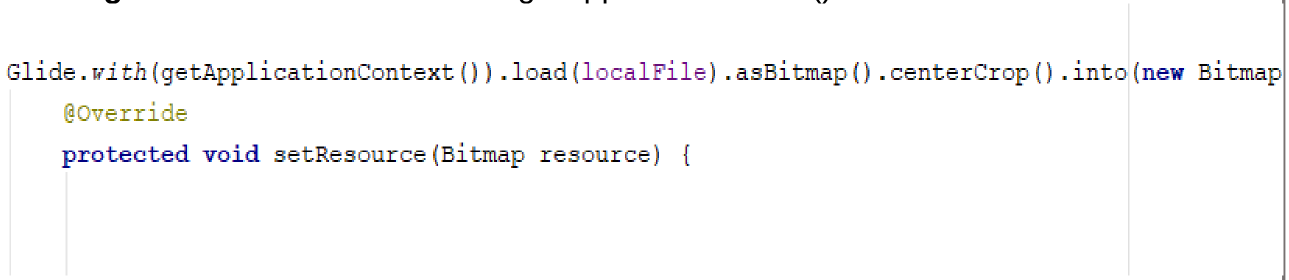
To avoid memory leaks I avoided passing context objects into activities and replaced the actual Context name with getApplicationContect().

**Figure 38 –** Example of Glide.



```
Glide.with( activity: Main2Activity.this).load(localFile).asBitmap().centerCrop().into(new BitmapImage
    @Override
    protected void setResource(Bitmap resource) {



        image.setImageBitmap(resource);
    }
```

**Figure 39 –** Fixed version with getApplictionContext().

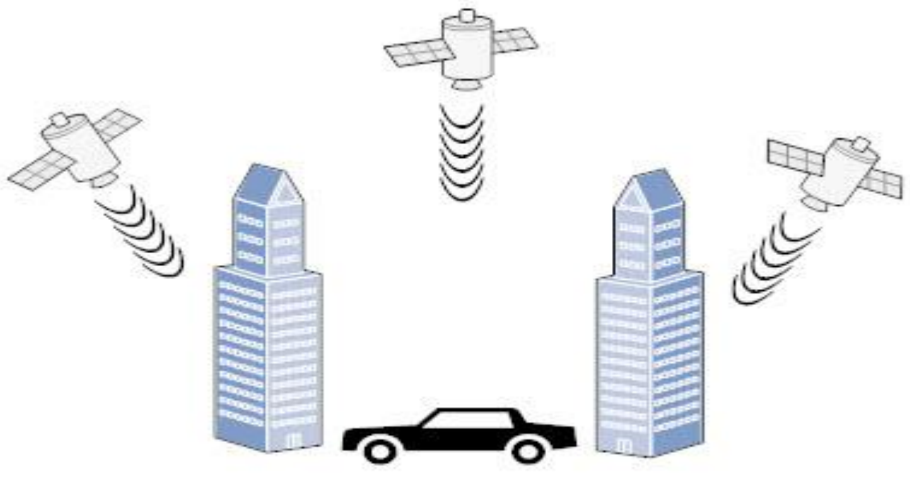


```
Glide.with(getApplicationContext()).load(localFile).asBitmap().centerCrop().into(new Bitmap
    @Override
    protected void setResource(Bitmap resource) {
```

**4) Location Returning Null.**

During the development stages of the application when I would go to frequently test it, I was receiving the result of 0.0,0.0 for latitude and longitude values. Although I had tested the GPS provider and it was working perfectly I could not establish why I was receiving a result of 0.0. After studying the application and external factors associated with the application I found out what the Problem was. When I was testing the application inside buildings I was receiving a value of 0.0 since the GPS signals were finding it difficult to penetrate through building material.

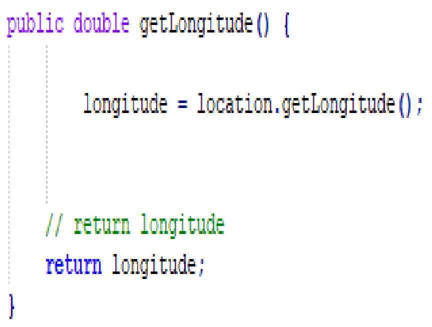
**Figure 40-** Signals penetrating through buildings.



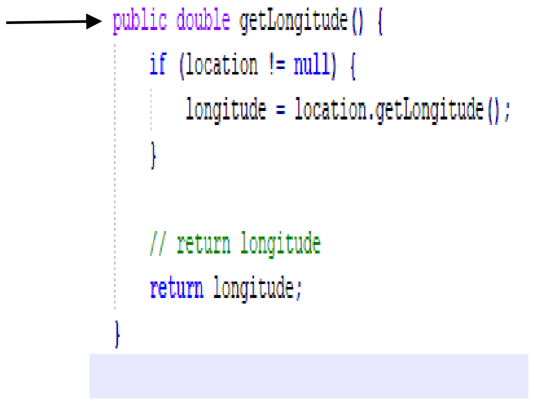
**Resolution.**

To fix the problem I had to fix my GPS Tracker class, I stopped the getLongitude() and getLatitude () methods from pulling values when the location was null.

**Figure 41**



```
public double getLongitude() {

        longitude = location.getLongitude();


    // return longitude
    return longitude;
}
```



```
public double getLongitude() {
    if (location != null) {
        longitude = location.getLongitude();
    }

    // return longitude
    return longitude;
}
```

## 9). Results

Overall I am delighted with the outcome of the Projects and felt that I have met m goals stated in the functional specification.  The application at standing point consists of two modes the Employer mode and the Employee mode. The Employer is able to create/ edit profile/ view conversations, view contacts and upload files that ae revelant. The Employee cannot contact the Employer first it is the Employer that makes first contact. The Employer can select area of employment and view available Employees on google maps API according to their location. There He/She can view the Employee profile, view ratings create raatings and message the Employee. The web application is soley designed just for statistics purposes where only members of the application are allowed to view.

However as I have felt that I met all my goals there are possible considerations that I would implemented if I had the chance to create the application again. I feel that I would have desiged the applications in parrell on the webapplication where the web application would consisnt of all the features of the android application.

When I was undergoing my user testing, I started to realse that there may be some barriers to the application that the users might find irritating. For example when I was testing a particular user stated that the location range was only 100km which was a short distance considering her field of work. Another user was asking me about being able to create working groups in the application where they could have their own chatrooms for a specified job.

However I felt like these were options that I would only consider during the implementation of the application as I would have never condidered during the planning phase. To avoid these misconceptions in the future I would recommend detailed requirements gathering before any of the implementation is started and allow the possibility of a user on site to influence the quaility of the application.

## 10). Future Work.

Now I feel that I am very passionate to further pursue in the development of the application as I feel it has a very good purpose and would fit very well in this current culture. Employment levels are further increasing as well as the skills and trade of Employees.
Although there are a lot of possible considerations of improvement to the application, I found that after I had completed my user testing a lot of people from different ages had a lot of positive comments about the application. This goes to show the innovative purpose of the application and how users could find it very easy to build their trust around it.

Another consideration that I was considering was to adapt the application for educational purposes, where students could go on to the application and search for grinds and lessons in a particular subject according to their location. I feel that the application can be adapted to fit a lot of users needs where location is considered a factor.