CHAPTER 5
IMPLEMENTATION AND TESTING

5.1 Implementation

After having a project from Firebase or Google console, the name of the package that has been created to create an application will be requested. After that, it is directed to the page to download the JSON file.

```
1. {
2.   "project_info": {
3.     "project_number": "850287614524",
4.     "firebase_url": "https://skripsi-219010.firebaseio.com",
5.     "project_id": "skripsi-219010",
6.     "storage_bucket": "skripsi-219010.appspot.com"
7.   },
8.   "client": [
9.     {
10.       "client_info": {
11.      "mobilesdk_app_id":
   "1:850287614524:android:c0b3d15d3409c694",
12.      "android_client_info": {
13.      "package_name": "taindri.example.com.skripsiku"
14.             }
```

Lines 1-7 are projects that are created in firebase and later automatically exist and can be viewed or edited when opening the google console with the same email. Contains the need for a project with package name information made in Android Studio.

```
15.      dependencies {
16.      classpath 'com.android.tools.build:gradle:3.1.3'
17.      classpath 'com.google.gms:google-services:4.0.1'
18.      }
```

To be able to run a JSON file that has been entered into the android studio package, it takes a despendency that functions so that when displaying data can

run properly according to the android studio sdk itself. Line 15 - 16 is the dependency that must exist in the file gradle. Because if it doesn't exist and doesn't match the Android Studio version of SDK, the application won't run.

```
19.   implementation 'com.android.support:appcompat-v7:27.0.1'
20.   testImplementation 'junit:junit:4.12'
21.   implementation 'com.google.android.gms:play-services:11.0.4'
22.   implementation 'com.google.gdata:core:1.47.1'
23.   implementation 'com.android.support:recyclerview-v7:27.0.1'
24.   implementation 'com.yarolegovich:lovely-dialog:1.1.0'
25.   }
```

In the second section of gradle when completing the program in a class, there is an automatic library that automatically automatically increases in the grad directly. There are also things that must be added manually according to the needs used to run the application. Line number 19-21 is the dependency following the SDK used in the project, if the android studio is still an older version then it must follow the old version of the SDK too. Line number 22 is the gdata library used by google fire contacts. Line number 23 is the library for the list of contact lists displayed by the application. Line number 24 is a library to display a descending dialog.

```
26.   private static final int AUTHORIZATION_CODE = 1993;
27.   private static final int ACCOUNT_CODE = 1601;
28.   private final String SCOPE = https://www.google.com/m8/feeds
```

The above code is a dialogue from Google, which is permission to get information that is still within the scope of Google Fire itself. Line number 28 is access to google fire contacts. Every google fire has different access.

```
29.     Glide.with(getApplicationContext()).load(personPhotoUrl)
30.             .thumbnail(0.5f)
31.             .crossFade()
32.             .diskCacheStrategy(DiskCacheStrategy.ALL)
```

```
33.             .into(imgProfilePic);
```

on Lines 29 - 33 is the display function when signing in a google account. When you have logged in, then exit the application and at the time of login again a photo of the Google account that has just logged in will appear.

```
34.     if (i==0){
35.         Intent intent = new
  Intent(mContext,ViewKontak.class);
36.         intent.putExtra("firstname",
  kontakList.get(position).getFirstName());
37.
  intent.putExtra("lastname",kontakList.get(position).getLastName
  ());
38.         intent.putStringArrayListExtra("listPhone",
  (ArrayList<String>) kontakList.get(position).getListPhone());
39.         intent.putStringArrayListExtra("listEmail",
  (ArrayList<String>) kontakList.get(position).getListEmail());
40.         intent.putExtra("contactid",
  kontakList.get(position).getContactId());
41.         mContext.startActivity(intent);
```

In code above is the code function of the dialog display for contact view.

```
42.     else if (i==1){
43.         Intent intent = new Intent(mContext,
  EditKontak.class);
44.         intent.putExtra("firstname",
  kontakList.get(position).getFirstName());
45.
  intent.putExtra("lastname",kontakList.get(position).getLastName
  ());
46.         intent.putStringArrayListExtra("listPhone",
  (ArrayList<String>) kontakList.get(position).getListPhone());
47.         intent.putStringArrayListExtra("listEmail",
  (ArrayList<String>) kontakList.get(position).getListEmail());
48.         intent.putExtra("contactid",
  kontakList.get(position).getContactId());
49.         ((Activity) mContext).startActivityForResult(intent,
  30);
```

In the code above is the code function from the dialog display for contact updates.

```
50. <uses-permission android:name="android.permission.INTERNET" />
51. <uses-permission
  android:name="android.permission.READ_CONTACTS" />
52. <uses-permission
  android:name="android.permission.WRITE_CONTACTS" />
53. <uses-permission
  android:name="android.permission.ACCESS_NETWORK_STATE" />
54. <uses-permission
  android:name="android.permission.USE_CREDENTIALS" />
55. <uses-permission
  android:name="android.permission.GET_ACCOUNTS" />
56. <uses-permission android:name="android.permission.CALL_PHONE"
  />
```

At Line 34 - 40 is the permission that is used so that the application can run smoothly. The application used requires permission through the device, namely reading contacts, writing or editing contacts, internet access, device credentials, and phone calls on the device.

```
57.     <activity android:name=".Kontak" />
58.     <activity android:name=".ViewKontak" />
59.     <activity android:name=".EditKontak" />
60.     <activity android:name=".TambahKontak">
```

At Line 41 - 44 is an activity class that will be used for intent or more clearly which will be called by a button or whatever.

```
61.  private URL ConvertToUrl(String urlStr) {
62.     try {
63.     URL url = new URL(urlStr);
64.     URI uri = new URI(url.getProtocol(),
  url.getUserInfo(),
65.                 url.getHost(), url.getPort(),
  url.getPath(),
66.                 url.getQuery(), url.getRef());
67.         url = uri.toURL();
68.         return url;
```

```
69.     } catch (Exception e) {
70.         e.printStackTrace();
71.     }
72.     return null;
```

At Lines 45 - 56 is a code that addresses the uri redirect. Used by OAuth Server to return results. The URI redirect sent by Third Party Apps to the OAuth Server must be the same as the Redirect URI registered when Third Party Apps gets Client Id. If the entered parameter is valid, then the OAuth Server will redirect to a login page like the picture above. The Resource Owner will enter the username and password that will be sent back to the OAuth Server. The OAuth Server checks whether the entered username and password are valid. If it's valid then the OAuth Server will send back an approval page whether the Resource Owner allows the data to be accessed by Third Party Apps. The data is in accordance with the data requested by Third Party Apps on the parameter scopes. If the Resource Owner agrees, the OAuth Server will send a data to the client via the HTTP Redirect. The data is in the form of code and state parameters. The code will be used by the client to be exchanged for tokens, and the state received is the same state as the state sent by the client.

```
73.     call.setOnClickListener(new View.OnClickListener() {
74.     public void onClick(View arg0) {
75.     Intent callIntent = new Intent(Intent.ACTION_CALL);
76.
  callIntent.setData(Uri.parse("tel:"+phoneNumberStr));
77.             if
  (ActivityCompat.checkSelfPermission(ViewKontak.this,
78.      Manifest.permission.CALL_PHONE) !=
  PackageManager.PERMISSION_GRANTED) {
79.                 return;
80.             }
81.             startActivity(callIntent);
82.         }
```

On line 57 - 66 is the code for making calls. Placed in the view contact class. On line 60 to call the number displayed on the contact. The 61st line is a permissions to be able to call contacts on the telephone device.

5.2 Testing

Testing on this project by conducting a demo via android studio or by building the APK and directly installing it on the cellphone. Can be seen below the testing results that have been done:

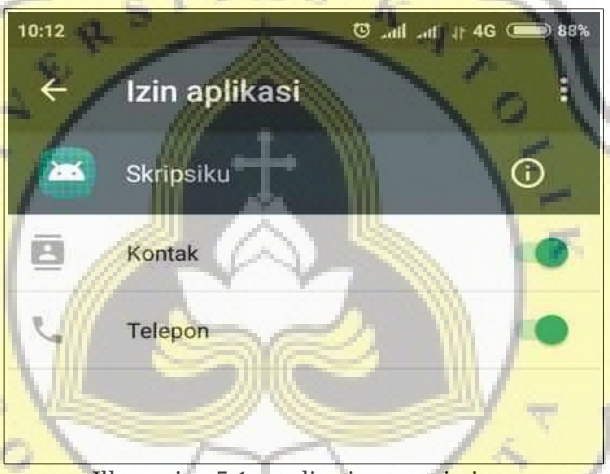

Illustration 5.1: application permissions

In the picture above shows a permit for the newly installed application. The required permissions are only contacts and telephone. Contacts to read data stored on the cellphone while the phone so that we can call and answer the phone from the application.

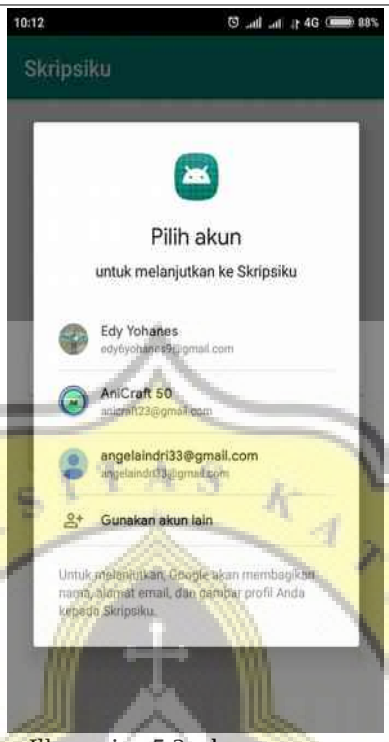

Illustration 5.2: choose account

In the picture above the user is expected to choose an account to log in.

Illustration 5.3: list contact

In the picture above is a contact list display. This contact is taken from synchronized contact data via google.

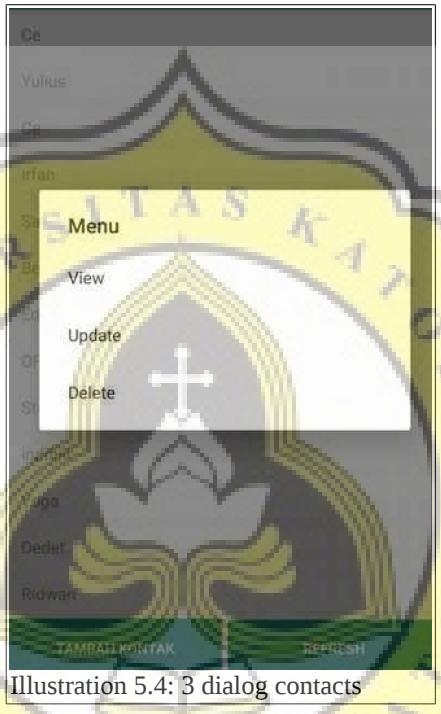

Illustration 5.4: 3 dialog contacts

In the picture above is a dialog. Users can choose the view dialog, edit, or delete. The dialog appears when clicking one of the contacts in the list.

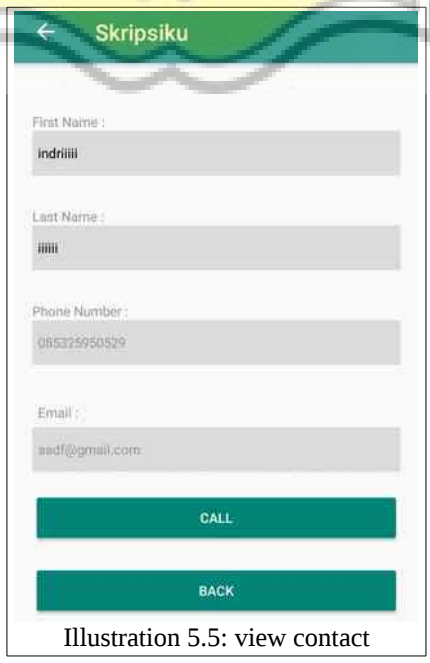

Illustration 5.5: view contact

In the picture above shows the view view. The Call button can be clicked and immediately switches to the call on the phone.

Illustration 5.6: calling

In the picture above is the process of calling after clicking the call button which can be found in the contact view.

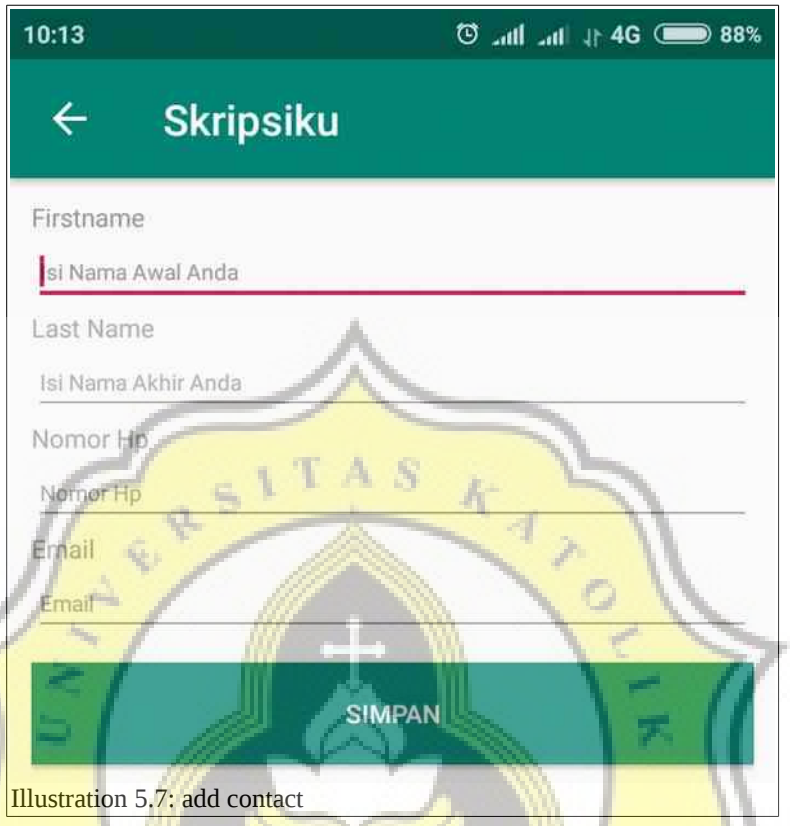

Illustration 5.7: add contact

In the picture above is a form to fill contact data. Can be found when clicking add contact. All must be filled in to be able to sync with a google account.