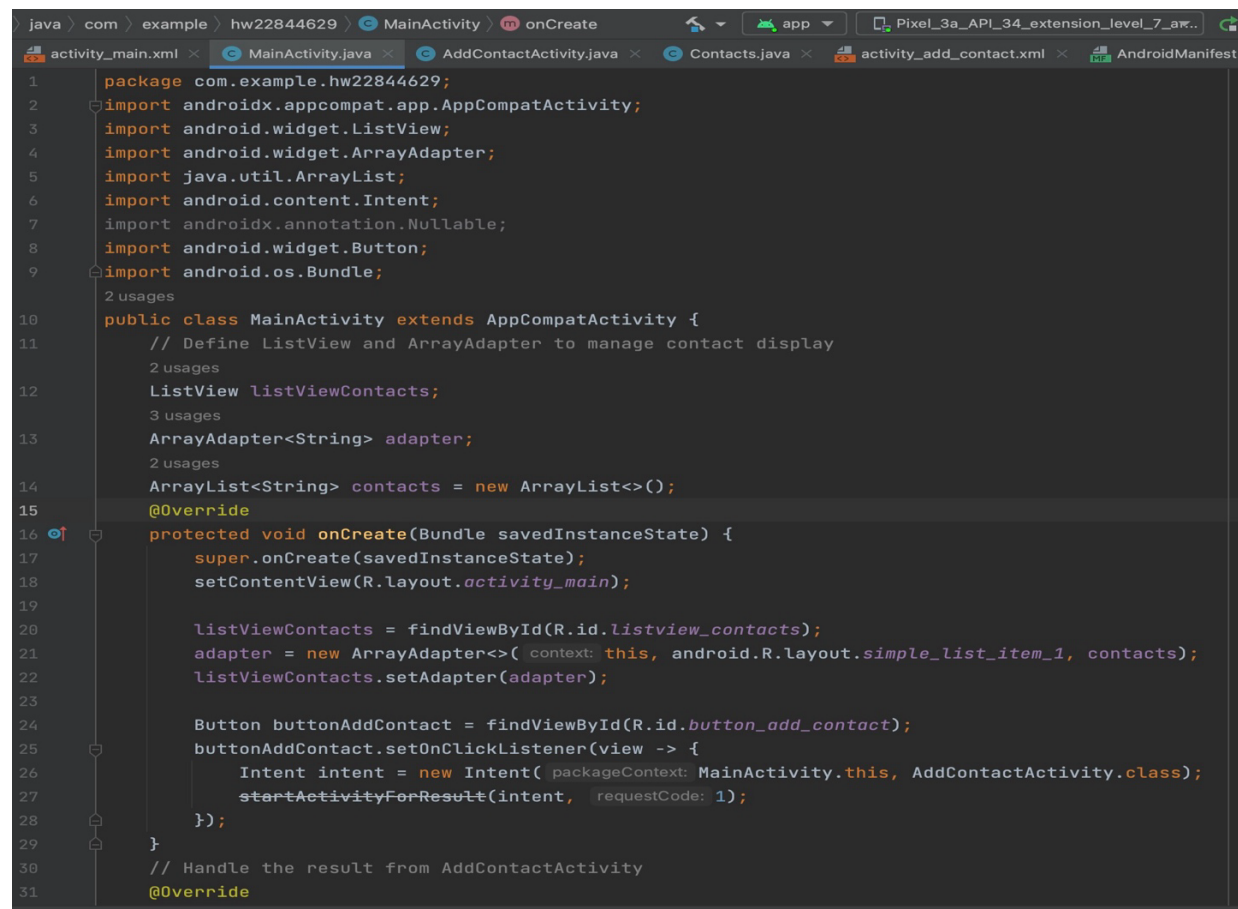


Develop a contact management app version 1:

- The main activity would start with no contact, but it has a button where the user could use to add a new contact.
- On clicking an "Add Contact" button in the main activity, it goes to the second activity, where the user could add the contact information such as name, phone number, email, etc.
- After entering a new contact, the user could click the submit button in the second activity, which the second activity will be closed, and the app goes back to the main activity.
- The main activity now should display the new contact information.

MainActivity.java: This code represents the app's main action. It begins with a "listview" and an "ArrayAdapter" to display the contact information. The "onCreate()" method creates a button that launches the "AddContctActivity" class for adding new contacts, and the "onActivityResult" method handles the results of the "AddContactActivity". As a result, the contact information is presented as a string consisting of name, phone number, and email address.

A screenshot of an IDE window showing the MainActivity.java file. The code is in Java and implements an Android activity. It imports necessary classes like AppCompatActivity, ListView, ArrayAdapter, ArrayList, Intent, and Bundle. The onCreate method initializes a ListView, an ArrayAdapter, and an ArrayList for contacts. It also finds a button in the layout and sets an onClick listener that starts AddContactActivity for a result. The code is as follows:

```
1 package com.example.hw2844629;
2 import androidx.appcompat.app.AppCompatActivity;
3 import android.widget.ListView;
4 import android.widget.ArrayAdapter;
5 import java.util.ArrayList;
6 import android.content.Intent;
7 import androidx.annotation.Nullable;
8 import android.widget.Button;
9 import android.os.Bundle;
10 public class MainActivity extends AppCompatActivity {
11     // Define ListView and ArrayAdapter to manage contact display
12     ListView listViewContacts;
13     ArrayAdapter<String> adapter;
14     ArrayList<String> contacts = new ArrayList<>();
15     @Override
16     protected void onCreate(Bundle savedInstanceState) {
17         super.onCreate(savedInstanceState);
18         setContentView(R.layout.activity_main);
19
20         listViewContacts = findViewById(R.id.listview_contacts);
21         adapter = new ArrayAdapter<>( context: this, android.R.layout.simple_list_item_1, contacts);
22         listViewContacts.setAdapter(adapter);
23
24         Button buttonAddContact = findViewById(R.id.button_add_contact);
25         buttonAddContact.setOnClickListener(view -> {
26             Intent intent = new Intent( packageContext: MainActivity.this, AddContactActivity.class);
27             startActivityForResult(intent, requestCode: 1);
28         });
29     }
30     // Handle the result from AddContactActivity
31     @Override
```

```

24     Button buttonAddContact = findViewById(R.id.button_add_contact);
25     buttonAddContact.setOnClickListener(view -> {
26         Intent intent = new Intent( packageContext: MainActivity.this, AddContactActivity.class);
27         startActivityForResult(intent,    requestCode: 1);
28     });
29 }
30 // Handle the result from AddContactActivity
31 @Override
32 protected void onActivityResult(int requestCode, int resultCode, Intent data) {
33     super.onActivityResult(requestCode, resultCode, data);
34     if (requestCode == 1 && resultCode == RESULT_OK && data != null) {
35         String contactName = data.getStringExtra( name: "CONTACT_NAME");
36         String contactPhone = data.getStringExtra( name: "CONTACT_PHONE");
37         String contactEmail = data.getStringExtra( name: "CONTACT_EMAIL");
38         // Ensure these are not null before adding them to the adapter's list
39         if (contactName != null && contactPhone != null && contactEmail != null) {
40             contacts.add(contactName + " - " + contactPhone + " - " + contactEmail);
41             adapter.notifyDataSetChanged();
42         }
43     }
44 }
45 }
46 }

```

MainActivity.xml: This is the user interface for the contact app's mainactivity.java; we're using a "RelativeLayout" to position a "Button" and a "ListView" which is used to add contacts and can be identified in the code by "@id/button_add_contact" and "@id/listview_contacts" is used to view the list of added contacts, and the "List View" ensures the user interface is well-organized. We can also use "ConstraintLayout" for it, but relative layout is good for simpler layouts with good flexibility and adaptability.

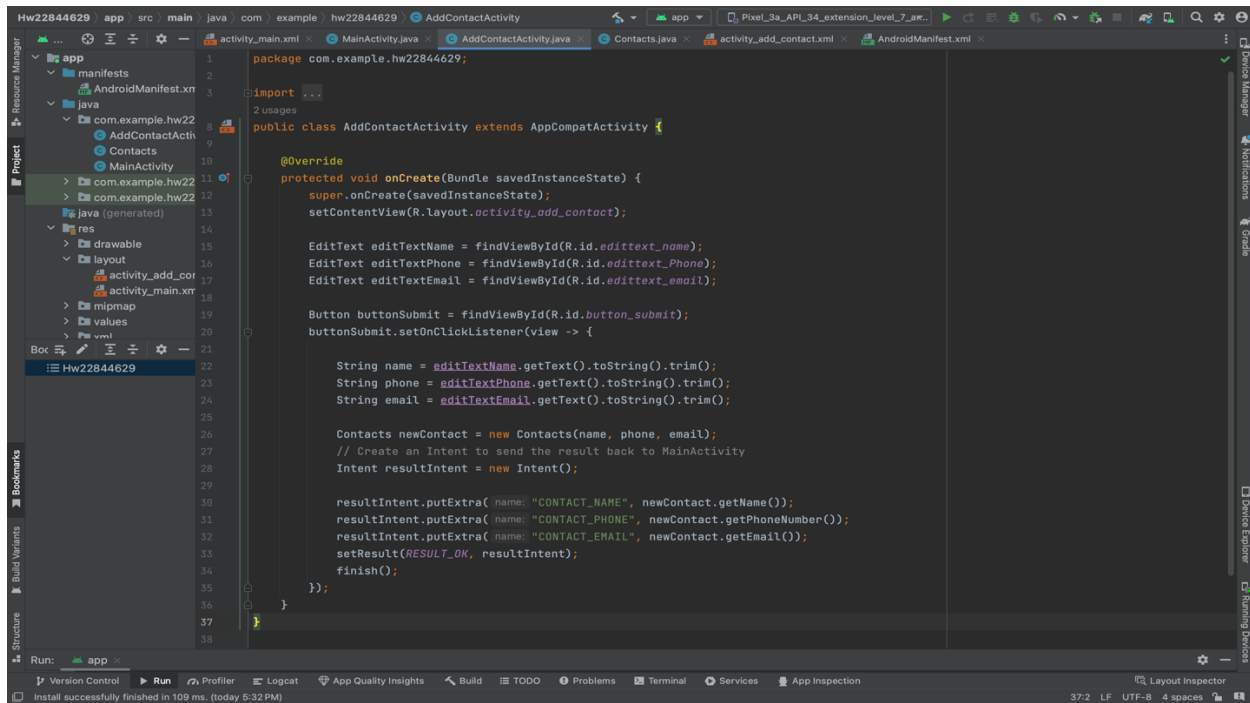
```

1  <?xml version="1.0" encoding="utf-8"?>
2  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent">
5
6      <Button
7          android:id="@+id/button_add_contact"
8          android:layout_width="wrap_content"
9          android:layout_height="wrap_content"
10         android:text="Add Contact" />
11
12     <ListView
13         android:id="@+id/listview_contacts"
14         android:layout_width="match_parent"
15         android:layout_height="match_parent"
16         android:layout_below="@id/button_add_contact" />
17
18 </RelativeLayout>

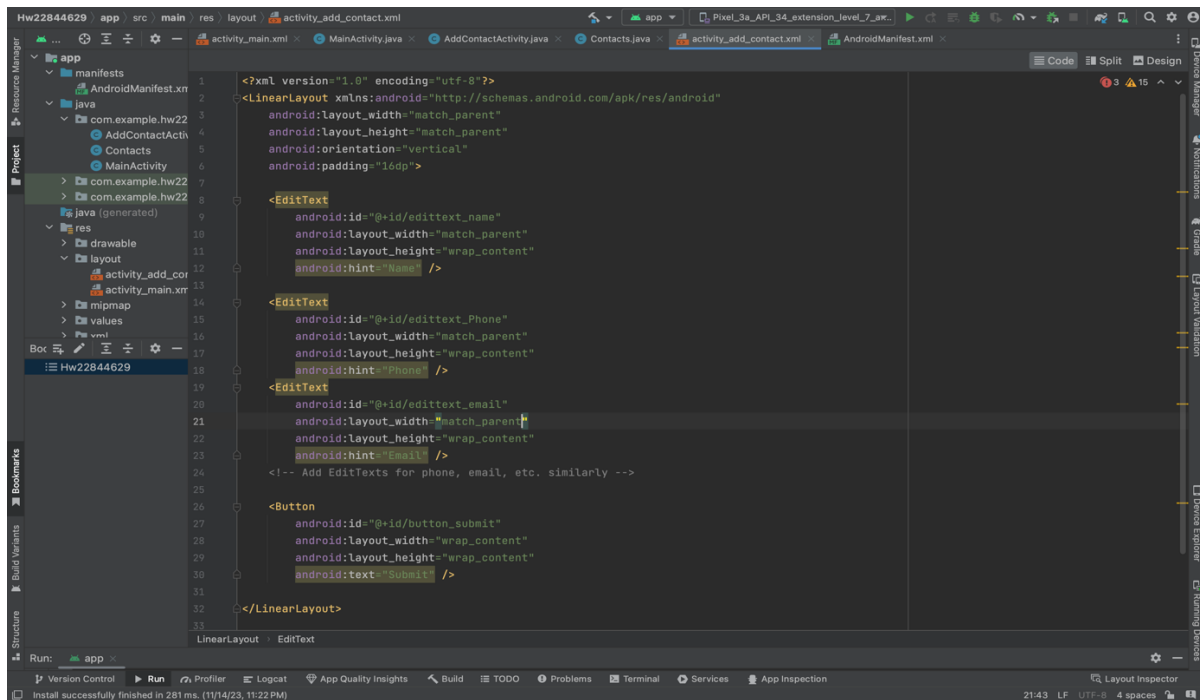
```

AddContactsActivity.java: This class allows users to enter new contact information. It shows a layout with three fields for entering a contact's name, phone number, and email address, which is generated using "EditText". Once the submit button is clicked, the data entered by the user is

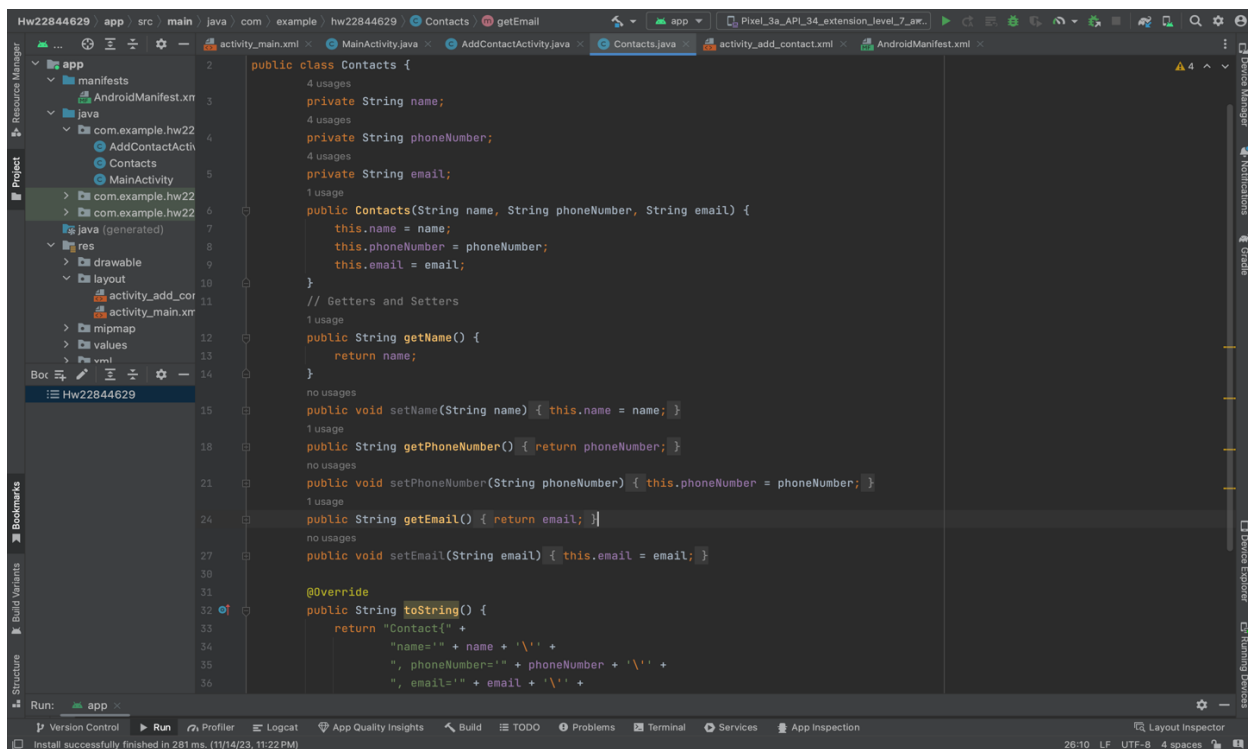
recorded. This submit button is made with the "buttonSubmit" . The click listener for this button produces a new instance of a 'Contacts' object with the entered data, ensuring that user input is wrapped in an organized format. The class then prepares an "Intent" to send back the data to "MainActivity" with all the contact information included as an additional to the "Intent." Finally, "AddContactActivity" completes its operation and returns to "MainActivity," passing along the entered contact information for future use.



AddConactsActivity.xml: We're utilizing "LinearLayout" with a vertical orientation for this user interface for addcontactactivity. This layout features three fields, each with its own unique ID, namely 'edittext_name,' 'edittext_Phone,' and 'edittext_email,' which allow the user to fill up the contact's name, phone number, and email address. These inputs are intended to fill the parent's width. View their height and wrap it to fit the content. It also has a "Button" with the ID "button_submit" that is placed below the input fields to submit the given contact information. This layout has been padded with 16dp to make it more appealing and user pleasant.



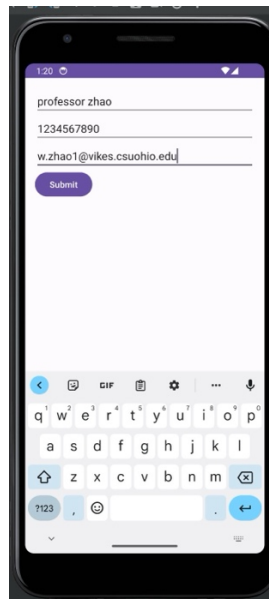
Contacts.java: The "Contacts" class in the application defines a structure for the contacts app, encapsulating name, phone number, and email. It features a constructor that initializes these attributes as well as getter and setter methods for each of them. The method "toString()" generates a formatted string representation of the contacts' details, which aids in data processing and display within the app.

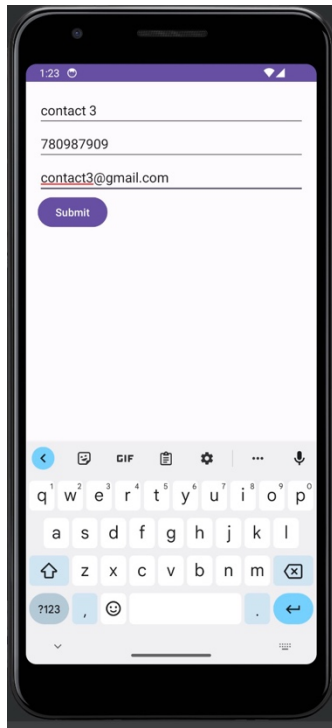


Screenshots



First Screen.

Second Screen
(filling up contact details)Third Screen
(first contact added)



Fourth Screen
(adding second contact)



Fifth Screen(second and third contact added)