Name: Sean Baker

ID: sbake021

Class Name: Application Development for Smart Devices

Assignment Name: MyNewContactList App

Computer: MacBook Air M2, macOS Sequoia

IDE: Android Studio Ladybug | 2024.2.1 Patch 1 Build #AI242.19072.14.2412.12360217, built on September 12, 2024

Runtime Version: 17.0.11+0-17.0.11b1207.24-11852314 aarch64

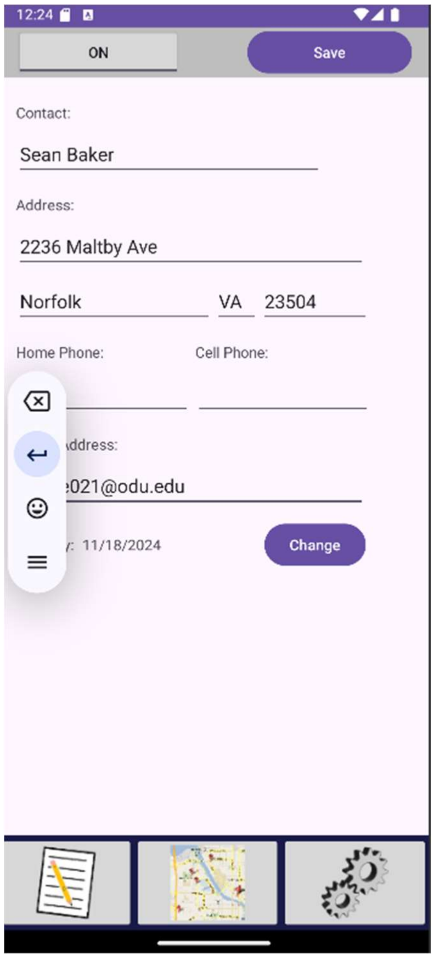
VM: OpenJDK 64-Bit Server VM by JetBrains s.r.o.

Purpose

This project enhances the previous address book application by incorporating several new features and fixing several missing functions. Users can now click on the **Map icon** to view their contact's location on Google Maps or view all their contacts on Google Maps in a list view. A **Date Picker** has been introduced, allowing users to select their birthday through a user-friendly

UI, as shown in **Figure Two**. This version also includes a fixed **ListView** that correctly manages the insertion of users into the database and allows them to be deleted effectively.

These enhancements aim to improve the application's functionality and align it with modern development standards provided by **Android Studio Ladybug** and **SDK 33.0**, ensuring compatibility with Google Play Store requirements.



Contact:
Sean Baker

Address:
2236 Maltby Ave
Norfolk VA 23504

Home Phone: Cell Phone:

ddress:
021@odu.edu

: 11/18/2024

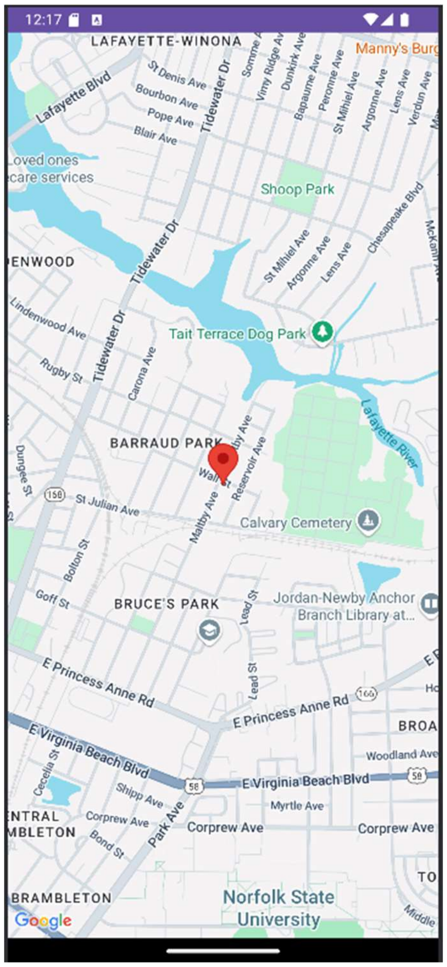
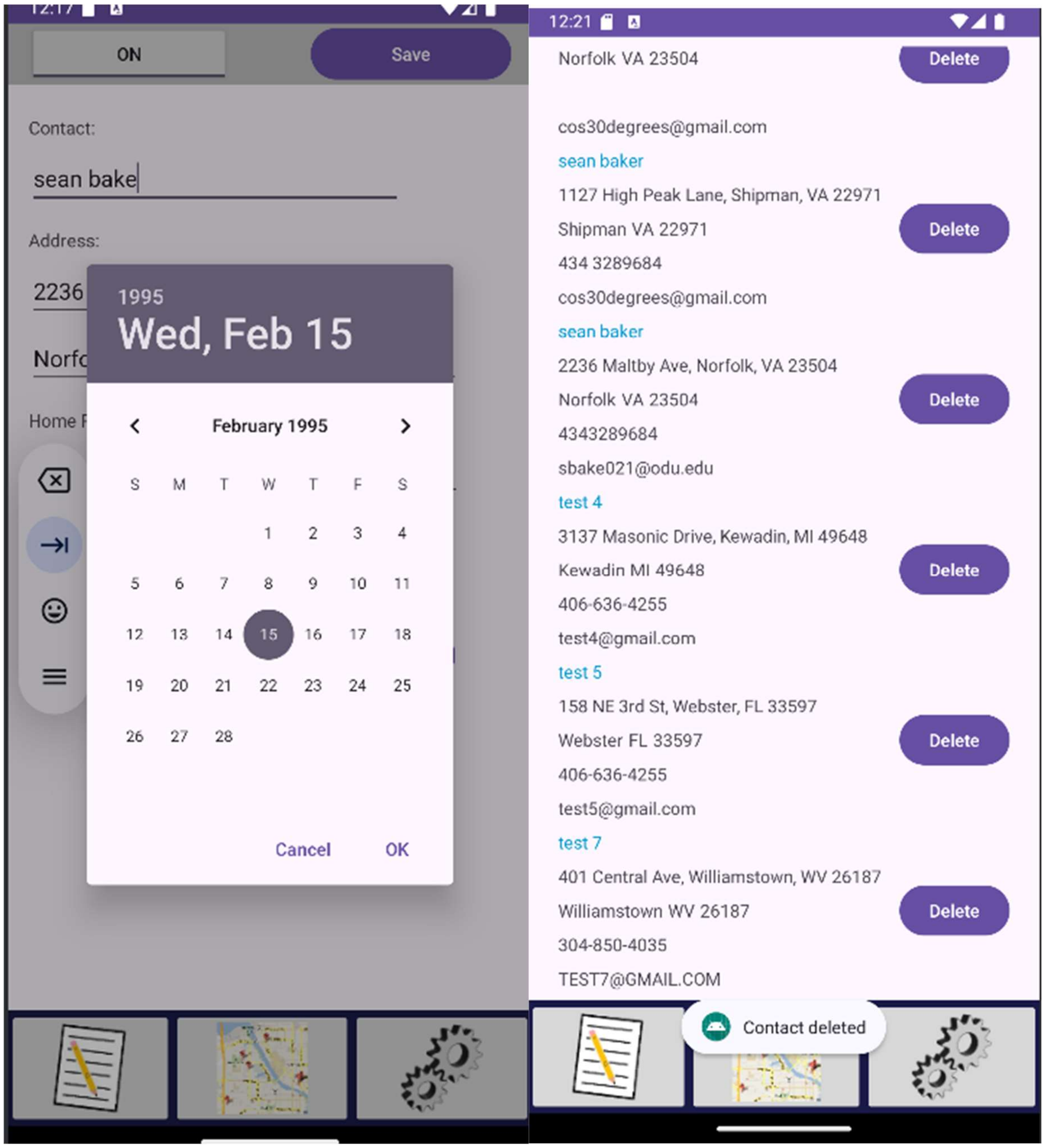


Figure 1 Main Application with new Map View Implementation allowing the user to see the contact addresss on Google Maps

Code Explanation



Contact:
sean bake
Address:
2236
Norfo
Home P

1995
Wed, Feb 15

February 1995

S	M	T	W	T	F	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28				

Cancel OK

Norfolk VA 23504

cos30degrees@gmail.com
sean baker
1127 High Peak Lane, Shipman, VA 22971
Shipman VA 22971
434 3289684
cos30degrees@gmail.com
sean baker
2236 Maltby Ave, Norfolk, VA 23504
Norfolk VA 23504
4343289684
sbake021@odu.edu
test 4
3137 Masonic Drive, Kewadin, MI 49648
Kewadin MI 49648
406-636-4255
test4@gmail.com
test 5
158 NE 3rd St, Webster, FL 33597
Webster FL 33597
406-636-4255
test5@gmail.com
test 7
401 Central Ave, Williamstown, WV 26187
Williamstown WV 26187
304-850-4035
TEST7@GMAIL.COM

Figure 2 Newly implemented date picker and correctly implemented listview with the ability to delete contacts.

Purpose

1 AndroidManifest.xml

This time, I used the Android Ladybug | 2024.2.1. Since the Google Store will not allow apps below SDK 33.0, learning the most relevant SDK would be wise. Even if the compatibility is lower, it does not matter if no one can download it.

Some classes display data as referenced in my Android manifest file. MainActivity ContactListActivity and ContactMapActivity

```
 < activity
    android:name=".MainActivity"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

<!-- Contact List Activity -->
< activity
    android:name=".ContactListActivity"
    android:exported="true" />

<!-- Contact Map Activity -->
< activity
    android:name=".ContactMapActivity"
    android:exported="true" />
```

This is a critical file where you define < activity> elements. These elements declare an activity that is part of your application. Activities represent individual screens or UI components in your app. As you can see here, we have defined three activities: ContactListActivity, ContactMapActivity, and ContactMapActivity. We also use it to store our Google Maps API key and set the Android emulator permissions, which give us access to location data.

2. **MainActivity.java**

The **MainActivity** serves as the app's entry point. It initializes the map and contact list activities, along with several buttons. The onCreate method ensures that all key UI components are initialized and ready for interaction.

```
private ContactDataSource contact data source;
```

```
/**
 * Called when the activity is first created.
 * @param savedInstanceState If the activity is being re-initialized after
being shut down, then this Bundle contains the data it most recently supplied
in onSaveInstanceState(Bundle).
 */
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    contactDataSource = new ContactDataSource(this);

    initListButton();
    initSaveButton();
    initToggleButton();
    initSettingsButton(R.id.imageButtonSettings);
    initMapButton(); // Add this line
    setForEditing(false);
    initChangeDateButton();
}
```

The **onCreate** method in the **MainActivity** class is the entry point for the activity when it is first created. This method is responsible for setting up the user interface and initializing various activity components. First, the method calls super.onCreate(savedInstanceState) to ensure that the base class's implementation of onCreate is executed. It then sets the content view to the layout resource in activity_main.xml.

```
setContentView(R.layout.activity_main);
```

Next, the method initializes the ContactDataSource object, which is used to interact with the database:

```
contactDataSource = new ContactDataSource(this);
```

The method then calls several initialization methods to set up buttons and click listeners. These methods include *initListButton*, *initSaveButton, initToggleButton, initSettingsButton, initMapButton*, and *initChangeDateButton*. Each of these methods is responsible for configuring a specific button and defining its behavior when clicked:

```
initListButton();
initSaveButton();
initToggleButton();
initSettingsButton();
initMapButton();
setForEditing(false);
initChangeDateButton();
```

The *saveContact* method in the **MainActivity** class saves the contact information entered by the user. It first retrieves the contact details from the input fields by calling the *getContactFromInput* method. If the contact details cannot be retrieved (i.e., the method returns null), a Toast message is displayed to inform the user that the location for the Address cannot be fetched, and the method returns early.

```
Contact newContact = getContactFromInput();
if (newContact == null) {
    Toast.makeText(this, "Unable to fetch location for the address.",
Toast.LENGTH_SHORT).show();
    return;
}
```

If the contact details are successfully retrieved, the method uses contactDataSource.java to insert the new contact into the database. After the insertion, the is closed. If the insertion is successful, a *Toast* message is displayed to inform the user that the contact has been saved, and an `Intent` is created to start the **ContactListActivity**. The intent is flagged to clear the top of the activity stack.

```
    contactDataSource.open();
        boolean wasSuccessful = contactDataSource.insertContact(newContact);
        contactDataSource.close();

        if (wasSuccessful) {
            Toast.make text(this, "Contact Saved!",
Toast.LENGTH_SHORT).show();
            Intent intent = new Intent(MainActivity.this,
ContactListActivity.class);
            intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
            startActivity(intent);
        } else {
            Toast.makeText(this, "Error saving contact!",
Toast.LENGTH_SHORT).show();
        }
    } catch (Exception e) {
        Toast.make text(this, "Database Error!", Toast.LENGTH_SHORT).show();
    }
}
```

If an exception occurs during the database operation, a `Toast` message informs the user of a database error.

The *getContactFromInput* method retrieves the contact details from the input fields and creates a **Contact** object. It constructs the full Address from the individual address components. It uses the ***AddressUtils.getLatLngFromAddress*** method to fetch the latitude and longitude for the Address if the geocoding fails (i.e., the Address does not exist, it will return null).

```
String fullAddress = editAddress.getText().toString() + ", " +
        editCity.getText().toString() + ", " +
        editState.getText().toString() + " " +
        editZipCode.getText().toString();

LatLng latLng = AddressUtils.getLatLngFromAddress(full address, this);
if (latLng == null) {
    return null;
}
```

If the geocoding is successful, a new **Contact** object is created and populated with the retrieved details, including the latitude and longitude. The method then returns the **Contact** object.

```
Contact contact = new Contact();
contact.setName(editName.getText().toString());
contact.setAddress(fullAddress);
contact.setCity(editCity.getText().toString());
contact.setState(editState.getText().toString());
contact.setZipCode(editZipCode.getText().toString());
contact.setPhone(editPhone.getText().toString());
contact.setEmail(editEmail.getText().toString());
contact.setLatitude(latLng.latitude);
contact.setLongitude(latLng.longitude);

return contact;
```

If we click the change button seen here:

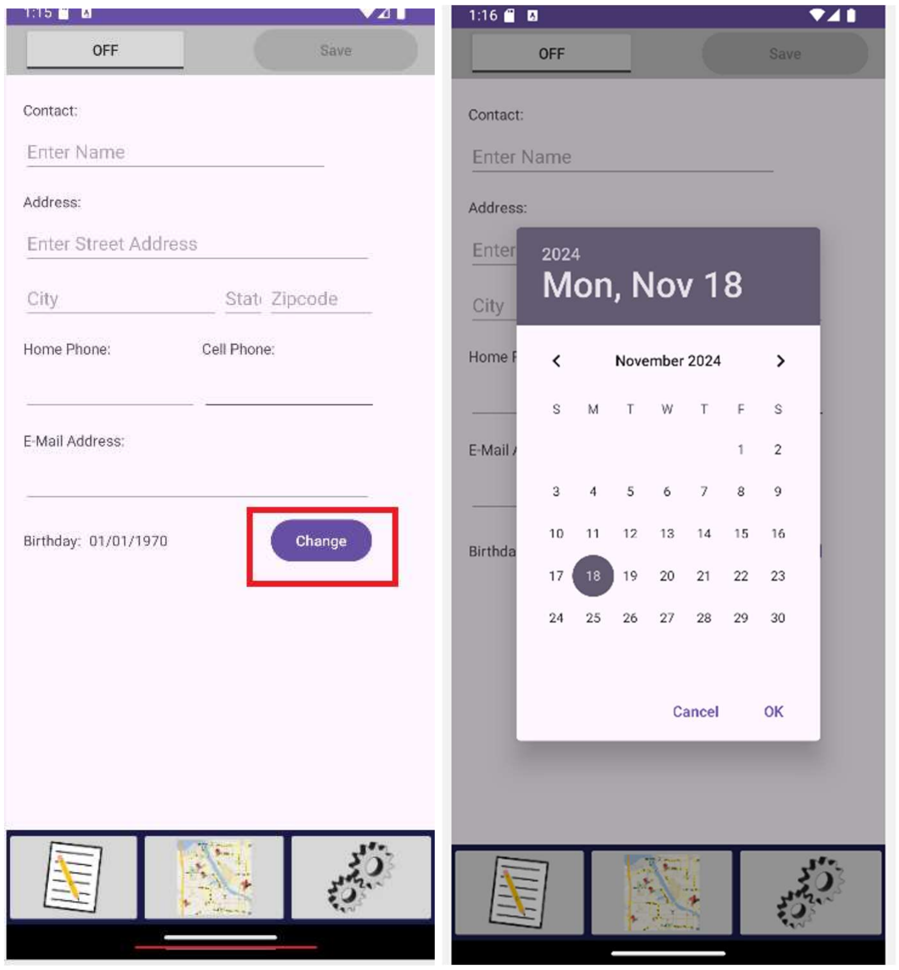


Figure 2.2 DatePicker and datepicker menu

```java
package com.sbake021.myapplication;

import android.app.DatePickerDialog;
import android.app.Dialog;
import android.os.Bundle;
import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.fragment.app.DialogFragment;

import java.util.Calendar;

/**
 * DatePickerFragment is a DialogFragment that displays a date picker dialog.
 * It allows the user to select a date and notify a listener when it is
selected.
 */
public class DatePickerFragment extends DialogFragment {

    /**
     * Interface for listening to date selection events.
     */
    public interface DatePickerListener {
        /**
         * Called when a date is selected.
         * @param selectedDate The selected date.
         */
        void onDateSelected(Calendar selectedDate);
    }

    private DatePickerListener listener;

    /**
     * Sets the listener for date selection events.
     * @param listener The listener to set.
     */
    public void setDatePickerListener(DatePickerListener listener) {
        this.listener = listener;
    }

    /**
     * Called to create the dialog.
     * @param savedInstanceState If the dialog is being re-initialized after
being shut down then this Bundle contains the data it most recently supplied
in onSaveInstanceState(Bundle).
     * @return A new instance of DatePickerDialog.
     */
    @NonNull
    @Override
    public Dialog onCreateDialog(@Nullable Bundle savedInstanceState) {
        Calendar calendar = Calendar.getInstance();
        int year = calendar.get(Calendar.YEAR);
        int month = calendar.get(Calendar.MONTH);
        int day = calendar.get(Calendar.DAY_OF_MONTH);

        return new DatePickerDialog(requireContext(), (view, year1, month1,
dayOfMonth) -> {
            Calendar selectedDate = Calendar.getInstance();
            selectedDate.set(year1, month1, dayOfMonth);
            if (listener != null) {
                listener.onDateSelected(selectedDate);
            }
        }, year, month, day);
    }
}
```

The **DatePickerFragment** is a reusable, modular **DialogFragment** through which users can specify a date via a date picker dialog. It employs a **DatePickerDialog** to instantiate the dialog and a listener pattern to pass the selected date back to listeners. Being lifecycle-aware, the **DatePickerFragment** plays nicely with Jetpack component-based modern Android applications, ensuring separation of concerns by decoupling the logic to display a date picker and handle user input from the calling activity or fragment.

The **DatePickerFragment's heart** is the **DatePickerListener** interface, which defines the ***onDateSelected(Calendar selectedDate)*** callback method handling the selected date. This can be implemented from other external components—activities or fragments—by calling the

setDatePickerListener(DatePickerListener listener) method and registering themselves as listeners. Upon the user's selecting a date, the **DatePickerDialog** intercepts the input, calling the ***onDateSelected*** callback on its registered listeners. This makes the **DatePickerFragment** generic, reusable, and flexible for any use case requiring date selection. The ***onCreateDialog(Bundle savedInstanceState)*** method initializes the date picker with the current date by default, ensuring a seamless user experience while allowing external components to handle the selected date as needed.

The **DatePickerFragment** interfaces seamlessly with the ***didFinishDatePickerDialog(Calendar selectedDate)*** using the **DatePickerListener** interface. Upon selecting a date, the **DatePickerFragment** will pass the selected date, enclosed in a **Calendar** object, to the registered listener using the ***onDateSelected*** method. This, in turn, enables the parent component to process the selected date and carry out further operations like refreshing the user interface.

In particular, the selected **Calendar** object is formatted by the **DateFormat** class in the "MM/dd/yyyy" format in the specific implementation of the ***didFinishDatePickerDialog*** method. The formatted date is dynamically shown in a **TextView** whose ID is `R.id.textBirthday` for immediate feedback to the user. Another fine example of separation of concerns that assures modularity, hence maintainability and reusability in the application design, is shown with **DatePickerFragment** and its corresponding method, ***didFinishDatePickerDialog***.

The Map Button

If we click the map button on the main activity screen, see below: This is due to the *initMapButton* method in the **MainActivity** class, which initializes the map button and sets its click listener. This method lets the user view the contact's location on a map.

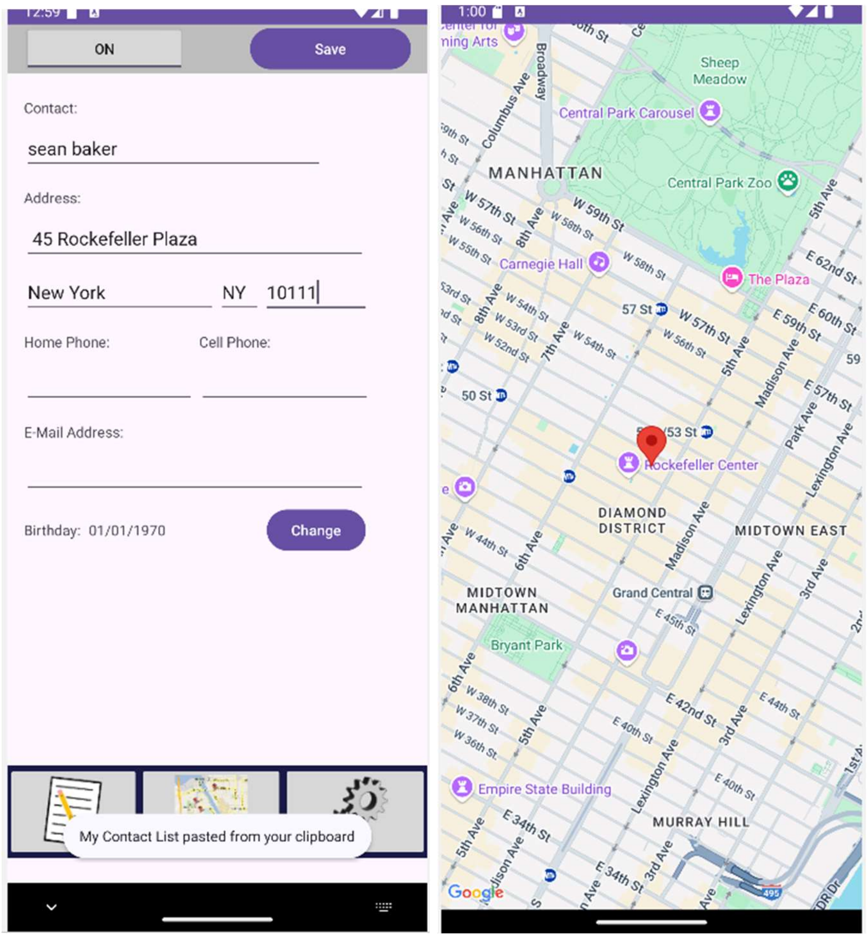

ON
Save
Contact:
sean baker
Address:
45 Rockefeller Plaza
New York NY 10111
Home Phone: Cell Phone:
E-Mail Address:
Birthday: 01/01/1970
Change
My Contact List pasted from your clipboard

Figure 3 Click the Map Button When on the main screen.

The map button in the **MainActivity** class passes information through an Intent. When the map button is clicked, the ***getContactFromInput*** method is called to retrieve the contact details entered by the user. If the contact details are successfully retrieved, an intent is created to start the **ContactMapActivity**. The contact object is passed to the **ContactMapActivity** as an extra, ensuring it is implemented as Parcelable and serializable.

First, the method retrieves the `ImageButton` view for the map button using `findViewById`:

```
ImageButton mapButton = findViewById(R.id.imageButtonMap);
```

Next, it sets an OnClickListener on the map button. When the button is clicked, the getContactFromInput method is called to retrieve the contact details entered by the user:

```
mapButton.setOnClickListener(view -> {
    Contact contact = getContactFromInput();
    if (contact == null) {
        Toast.makeText(this, "Unable to fetch location for the address.",
Toast.LENGTH_SHORT).show();
        return;
    }
```

This setup lets the user view the contact's location on a map by clicking the map button, which starts the **ContactMapActivity** with the contact's details. The map button in the **MainActivity.java** class passes information to the **ContactMapActivity** through an Intent. When the map button is clicked, the ***getContactFromInput*** method is called to retrieve the contact details entered by the user. An Intent is created to start the **ContactMapActivity.java** if the contact details are successfully retrieved. The contact object is then passed to the **ContactMapActivity** as an extra, ensuring it implements Parcelable

```
mapButton.setOnClickListener(view -> {
    Contact contact = getContactFromInput();
    if (contact == null) {
        Toast.make text(this, "Unable to fetch location for the address.",
Toast.LENGTH_SHORT).show();
        return;
    }

    Intent intent = new Intent(MainActivity.this, ContactMapActivity.class);
    intent.putExtra("CONTACT", (Parcelable) contact);
    startActivity(intent);
});
```

Parcelable is an Android-specific interface that efficiently serializes and transfers data between activities, fragments, or services. It allows complex objects to be passed via an Intent or Bundle, making it a commonly used mechanism in Android development for lightweight and efficient object serialization.

I expanded on the application requirements and decided to save the latitude and longitude to the database

```
public class DBContactListHelper extends SQLiteOpenHelper


public static final String COLUMN_LATITUDE = "latitude";
public static final String COLUMN_LONGITUDE = "longitude";
```

Saving Contacts

The saveContact method handles saving user input into the database. It also converts the entered Address into geographic coordinates (latitude and longitude) using the **AddressUtils** class. This is required because google maps requires us to pass it lattitude and longitude. We take advantage of the Androids built in geocoder library to make this conversion.

```
public class AddressUtils {

    /**
     * Converts a given address into its corresponding latitude and longitude.
     *
     * @param Address The Address to be converted.
     * @param context The context in which the geocoder is being used.
     * @return A LatLng object containing the latitude and longitude of the
Address, or null if the Address could not be geocoded.
     */
    public static LatLng getLatLngFromAddress(String address, Context context)
{
        Geocoder geocoder = new Geocoder(context, Locale.getDefault());
        try {
            List< Address> addresses = geocoder.getFromLocationName(address,
1);
            if (addresses != null && !addresses.isEmpty()) {
                Address location = addresses.get(0);
                return new LatLng(location.getLatitude(),
location.getLongitude());
            }
        } catch (IOException e) {
            Log.e("AddressUtils", "Error in geocoding", e);
        }
        return null;
    }
```

I use a static method, so I do not have to initialize the class fully.

```
public static final String COLUMN_LATITUDE = "latitude";
public static final String COLUMN_LONGITUDE = "longitude";
```

In **DBContactHelper** and **ContactDataSource,** I have added latitude and longitude to the database. This allows me to pass all my contact's map information to the map when I want to view them in the ContactListActivity

```
public class ContactListActivity extends AppCompatActivity {
    private ContactDataSource contactDataSource;
    private ContactAdapter adapter;
    private ArrayList<Contact> contactList;

    /**
     * Called when the activity is first created.
     * @param savedInstanceState If the activity is being re-initialized after
previously being shut down then this Bundle contains the data it most recently
supplied in onSaveInstanceState(Bundle).
     */
@Override
protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_contact_list); // Ensure layout has
RecyclerView with id "contactRecyclerView"

        // Initialize RecyclerView
        RecyclerView recyclerView = findViewById(R.id.contactRecyclerView);
        recyclerView.setLayoutManager(new LinearLayoutManager(this));

        // Initialize data source and load contacts
        contactDataSource = new ContactDataSource(this);
        contactList = new ArrayList<>();

        contactDataSource.open();
        contactList = (ArrayList<Contact>) contactDataSource.getAllContacts();
// Ensure this method returns ArrayList<Contact>
        contactDataSource.close();

        // Set up the adapter
        adapter = new ContactAdapter(contactList, this);
        adapter.setDelete(true); // Disable delete functionality for this use
case
        recyclerView.setAdapter(adapter);

        // Initialize buttons
        initSettingsButton();
        initMapButton();
    }

/**
 * Initializes the map button and sets its click listener.
 */
private void initMapButton() {
    ImageButton mapButton = findViewById(R.id.imageButtonMap);
    mapButton.setOnClickListener(view -> {
        Intent intent = new Intent(ContactListActivity.this,
ContactMapActivity.class);
        intent.putParcelableArrayListExtra("CONTACT_LIST", contactList); //
Pass all contacts
        startActivity(intent);
    });
```

The **ContactListActivity** class in this Android application provides functionality to visually display a list of contacts on a map. This is achieved by initializing a button, the **Map Button**, which starts the **ContactMapActivity** when clicked, passing the contact list to it for rendering.

1. Initialization of the Map Button

The ***initMapButton*** method in the **ContactListActivity** lets us view all the contacts on the map. The **ContactMapActivity** is central to displaying contact locations on a Google Map. It leverages data passed through an Intent. The **Contact** class implementing the Parcelable interface is central to this functionality, which provides efficient data serialization. This way, the activity can receive either one or a list of contacts with needed details such as names, addresses, and geolocations. This information is then utilized to set markers on the map and position the camera to ensure good visualization of the map. The *RecyclerView* integrates with **ContactAdapter** and **ContactMapActivity** in this application to provide a coherent and interactive user experience in exploring contact locations.

For example, in **ContactAdapter**, a contact is passed when a user clicks on an item in the *RecyclerView*:

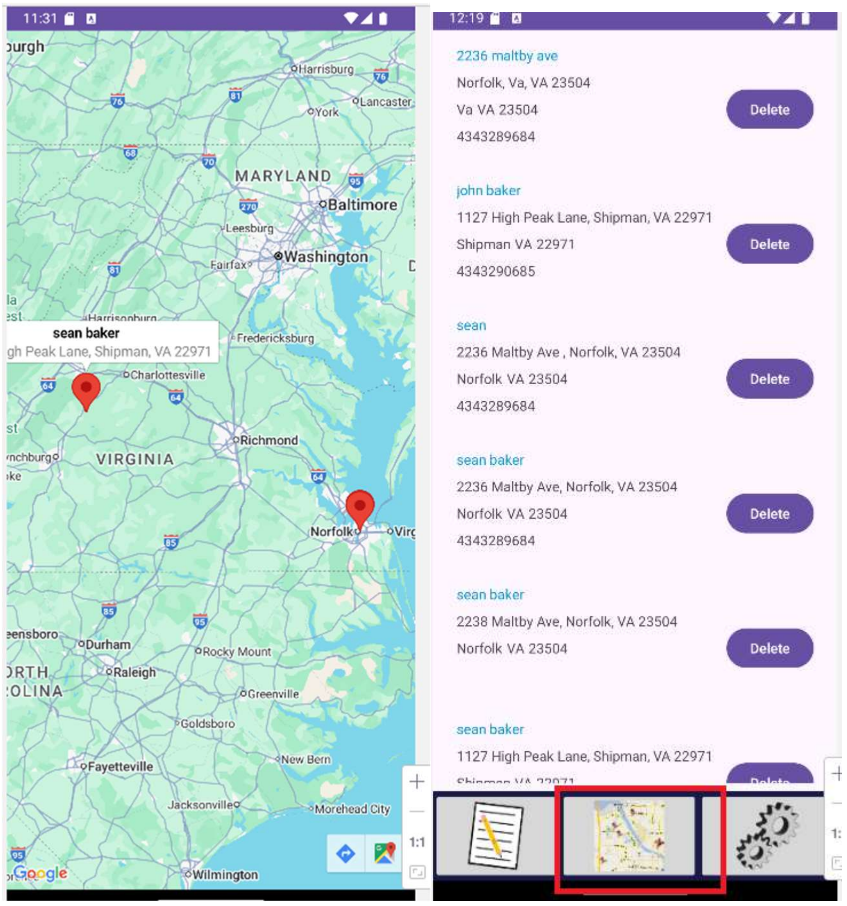
```
/**
 * RecyclerView is called to display the data in the specified position.
 * @param holder The ViewHolder should be updated to represent the item's
contents at the given position in the data set.
 * @param position The position of the item within the adapter's data set.
 */
@Override
public void onBindViewHolder(@NonNull ContactViewHolder holder, int position)
{
    Contact currentContact = contactData.get(position);
    holder.getTextName().setText(currentContact.getName());
    holder.getTextAddress().setText(currentContact.getAddress());
    holder.getTextCity().setText(currentContact.getCity());
    holder.getTextState().setText(currentContact.getState());
    holder.getTextZipCode().setText(currentContact.getZipCode());
    holder.getTextPhone().setText(currentContact.getPhone());
    holder.getTextEmail().setText(currentContact.getEmail());

    if (isDeleting) {
        holder.getDeleteButton().setVisibility(View.VISIBLE);
        holder.getDeleteButton().setOnClickListener(view ->
deleteItem(position));
    } else {
        holder.getDeleteButton().setVisibility(View.INVISIBLE);
    }

    // Set a click listener for the item view
    holder.itemView.setOnClickListener(view -> {
        Intent intent = new Intent(parentContext, ContactMapActivity.class);
        intent.putExtra("SELECTED_CONTACT", (Parcelable) currentContact); //
Pass the selected contact
        parentContext.startActivity(intent);
    });
}
```

In particular, if the interaction starts from either the **ContactListActivity** or **ContactAdapter**, where user actions are initiated and the contacts' data are transferred. **ContactAdapter**, upon clicking a contact in the *RecyclerView*, configures an intent that starts **ContactMapActivity** and attaches the selected contact as an extra. Suppose there is more than one number of contacts to be displayed. In that case, **ContactListActivity** sends an entire contact list through a method intended to send Parcelable arrays: This sets up the button responsible for this functionality. When the button is clicked, it creates an Intent to launch the **ContactMapActivity**. Using putParcelableArrayListExtra, the method passes the contactList to the new activity. This mechanism ensures that the **ContactMapActivity** receives the complete list of contacts for map visualization.

```
mapButton.setOnClickListener(view -> {
    Intent intent = new Intent(ContactListActivity.this,
ContactMapActivity.class);
    intent.putParcelableArrayListExtra("CONTACT_LIST", contactList); // Pass
all contacts
    startActivity(intent);
});
```



2236 maltby ave
Norfolk, Va, VA 23504
Va VA 23504
4343289684

john baker
1127 High Peak Lane, Shipman, VA 22971
Shipman VA 22971
4343290685

sean
2236 Maltby Ave , Norfolk, VA 23504
Norfolk VA 23504
4343289684

sean baker
2236 Maltby Ave, Norfolk, VA 23504
Norfolk VA 23504
4343289684

sean baker
2238 Maltby Ave, Norfolk, VA 23504
Norfolk VA 23504

sean baker
1127 High Peak Lane, Shipman, VA 22971

Figure 4 Mapview from Activity List

Upon reaching **ContactMapActivity**, the data will be retrieved and processed to determine whether a list of contacts or a single contact will be displayed. This flexibility allows both individual and group locations to be visualized efficiently.

ContactMapActivity extends ***AppCompatActivity*** and implements ***OnMapReadyCallback***, which ties the activity lifecycle with the lifecycle of the Google Map. The activity uses a MapView widget to render a map in its layout, providing a GoogleMap object for interactions such as adding markers and moving the camera. In the onCreate method, the activity performs the usual binding to the UI XML layout file, extracts the contact data from the intent, and initializes the MapView widget: The **ContactListActivity** class shows a list of contacts in a map view by using the ***initMapButton*** method. This method sets up a button that, when clicked, starts the **ContactMapActivity** and passes the list of contacts to it.

The **ContactMapActivity** class displays a list of contacts on a map view using Google Maps. It handles both individual and multiple contact locations. Here is the whole class.

```
public class ContactMapActivity extends AppCompatActivity implements
OnMapReadyCallback {

    private MapView mapView;
    private GoogleMap googleMap;
    private Contact contact;

    private static final String MAPVIEW_BUNDLE_KEY = "MapViewBundleKey";

    /**
     * Called when the activity is first created.
     * @param savedInstanceState If the activity is being re-initialized after
being shut down then this Bundle contains the data it most recently supplied
in onSaveInstanceState(Bundle).
     */
    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_contact_map);

        // Initialize MapView
        mapView = findViewById(R.id.mapView);
        Bundle mapViewBundle = savedInstanceState != null ?
savedInstanceState.getBundle(MAPVIEW_BUNDLE_KEY) : null;
        mapView.onCreate(mapViewBundle);

        // Fetch the contact passed via intent
        contact = (Contact) getIntent().getSerializableExtra("CONTACT");
        if (contact == null) {
            Toast.makeText(this, "No contact information available to display
on the map.", Toast.LENGTH_SHORT).show();
            finish(); // Exit the activity if no contact is available
            return;
        }

        // Initialize the map
        mapView.getMapAsync(this);
    }

    /**
     * Called when the map is ready to be used.
     * @param map A non-null instance of a GoogleMap associated with the
MapView.
     */
    @Override
    public void onMapReady(@NonNull GoogleMap map) {
        googleMap = map;

        // Create LatLng from contact data
        LatLng location = new LatLng(contact.getLatitude(),
contact.getLongitude());

        // Add marker for the contact
        googleMap.addMarker(new MarkerOptions()
                .position(location)
                .title(contact.getName())
                .snippet(contact.getAddress()));
        // Center the camera on the contact
        googleMap.moveCamera(CameraUpdateFactory.newLatLngZoom(location, 15));
    }

    /**
     * Called when the activity is resumed.
     */
    @Override
    protected void onResume() {
        super.onResume();
        mapView.onResume();
    }

    /**
     * Called when the activity is paused.
     */
    @Override
    protected void onPause() {
        super.onPause();
        mapView.onPause();
    }

    /**
     * Called when the activity is destroyed.
     */
    @Override
    protected void onDestroy() {
        super.onDestroy();
        mapView.onDestroy();
    }

    /**
     * Called when the system is running low on memory.
     */
    @Override
    public void onLowMemory() {
        super.onLowMemory();
        mapView.onLowMemory();
    }

    /**
     * Called to retrieve per-instance state from an activity before being
killed so that the state can be restored in onCreate(Bundle) or
onRestoreInstanceState(Bundle).
     * @param outState Bundle in which to place your saved state.
     */
    @Override
    protected void onSaveInstanceState(@NonNull Bundle outState) {
        super.onSaveInstanceState(outState);
        Bundle mapViewBundle = outState.getBundle(MAPVIEW_BUNDLE_KEY);
        if (mapViewBundle == null) {
            mapViewBundle = new Bundle();
            outState.putBundle(MAPVIEW_BUNDLE_KEY, mapViewBundle);
        }
        mapView.onSaveInstanceState(mapViewBundle);
    }
```

```
}
```
The previous methods handle lifecycle management when showing a **Google Map** in an Android activity. Up to now, the method ***onMapReady(GoogleMap map)*** does the work of setting up the map: it adds a marker at the contact's latitude and longitude, setting its title and snippet from the **Contact** name and Address, respectively; it also centers the camera on the location with a zoom level of 15. The methods ***onResume()***, ***onPause()***, ***onDestroy()***, and ***onLowMemory()*** are also implemented to manage the **MapView** component correctly, ensuring lifecycle events are passed down to prevent memory leaks or misuse of resources.

The ***onSaveInstanceState(Bundle outState)*** method saves the map's state in a bundle so that when the activity is restarted, the map does not need to reload completely, enhancing the user experience. Together, these methods manage the map's initialization, state preservation, and resource efficiency throughout its lifecycle, ensuring a seamless and robust implementation of Google Map functionality.

Activity lifecycle callbacks in Android:

1. **onCreate()**: This method is called when the activity is first created. It is where you should initialize the user interface and perform any setup tasks.
2. **onStart()**: This method is called when the activity becomes visible to the user but is not in the foreground.

3. **onResume()**: This method is called when the activity is brought to the foreground and becomes the active activity.
4. **onPause()**: This method is called when the activity loses focus and becomes partially visible.
5. **onStop()**: This method is called when the activity is no longer visible to the user.
6. **Restart ()**: This method is called when the activity is stopped and then started again.
7. **onDestroy()**: This method is called when the activity is being destroyed and is about to be removed from memory.

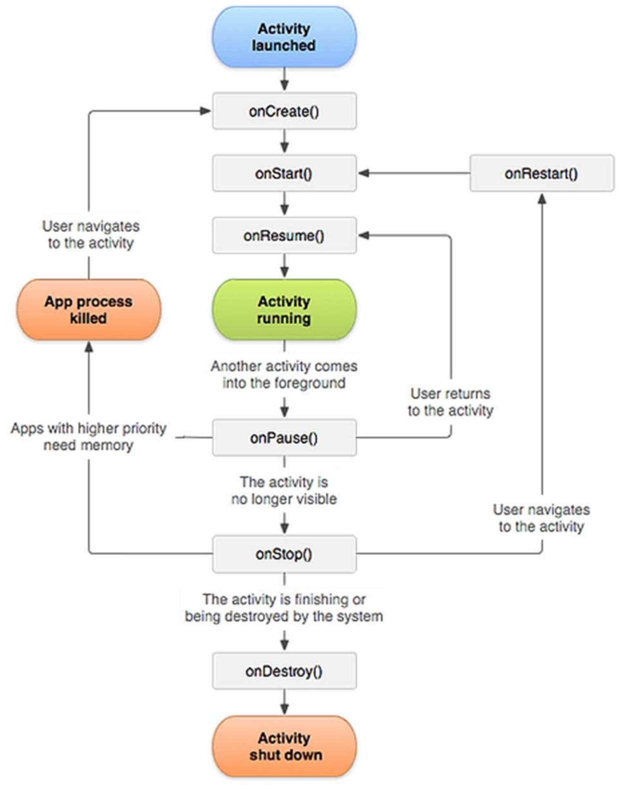

Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Figure 3.1 Oncreate method

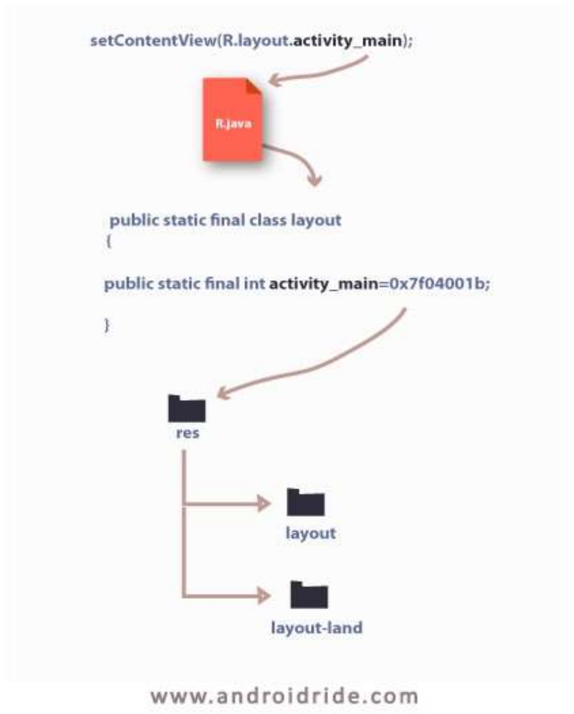

setContentView(R.layout.activity_main);
R.java
public static final class layout
{
public static final int activity_main=0x7f04001b;
}
res
layout
layout-land
www.androidride.com

Figure 3.2 Android Lifecycle

What Is An Android Activity | Robots.net. Retrieved September 29, 2024, from https://robots.net/tech/what-is-an-android-activity

Goodbye to Activity Lifecycle and Hello to Compose Lifecycle | by PINAR TURGUT | Medium. Retrieved September 29, 2024, from https://pinarturgut09.medium.com/goodbye-toactivity-lifecycle-and-hello-to-compose-lifecycle-6eaaf8270580

Geocoder. (n.d.). Android Developers. https://developer.android.com/reference/android/location/Geocoder

Admin. (2020, January 9). *Why setcontentview() in Android had been so popular till now?*. AndroidRide. https://androidride.com/what-setcontentview-android-studio/

Google Developers. (n.d.). *Get started with Google Maps Platform*. Retrieved from https://developers.google.com/maps/documentation/android-sdk/start

Android Developers. (n.d.). *Create dynamic lists with RecyclerView*. Retrieved from https://developer.android.com/guide/topics/ui/layout/recyclerview

GeeksforGeeks. (n.d.). *Difference between Serializable and Parcelable in Android*. Retrieved from https://www.geeksforgeeks.org/difference-between-serializable-and-parcelable-in-android/

Vogella. (n.d.). *Android Parcelable tutorial*. Retrieved from https://www.vogella.com/tutorials/AndroidParcelable/article.html

Johncarl81. (n.d.). *Parceler*. Retrieved from https://github.com/johncarl81/parceler

Android Developers. (n.d.). *Pickers*. Retrieved from https://developer.android.com/develop/ui/views/components/pickers

Android Developers. (n.d.). *Geocoder*. Retrieved from https://developer.android.com/reference/android/location/Geocoder

Android Developers. (n.d.). *Understand the activity lifecycle*. Retrieved from https://developer.android.com/guide/components/activities/activity-lifecycle

Google Developers. (n.d.). *Google Maps Utility Library for Android*. Retrieved from https://developers.google.com/maps/documentation/android-sdk/utility/setup

Android Developers Blog. (n.d.). *Updates and resources for Android development*. Retrieved from https://android-developers.googleblog.com/