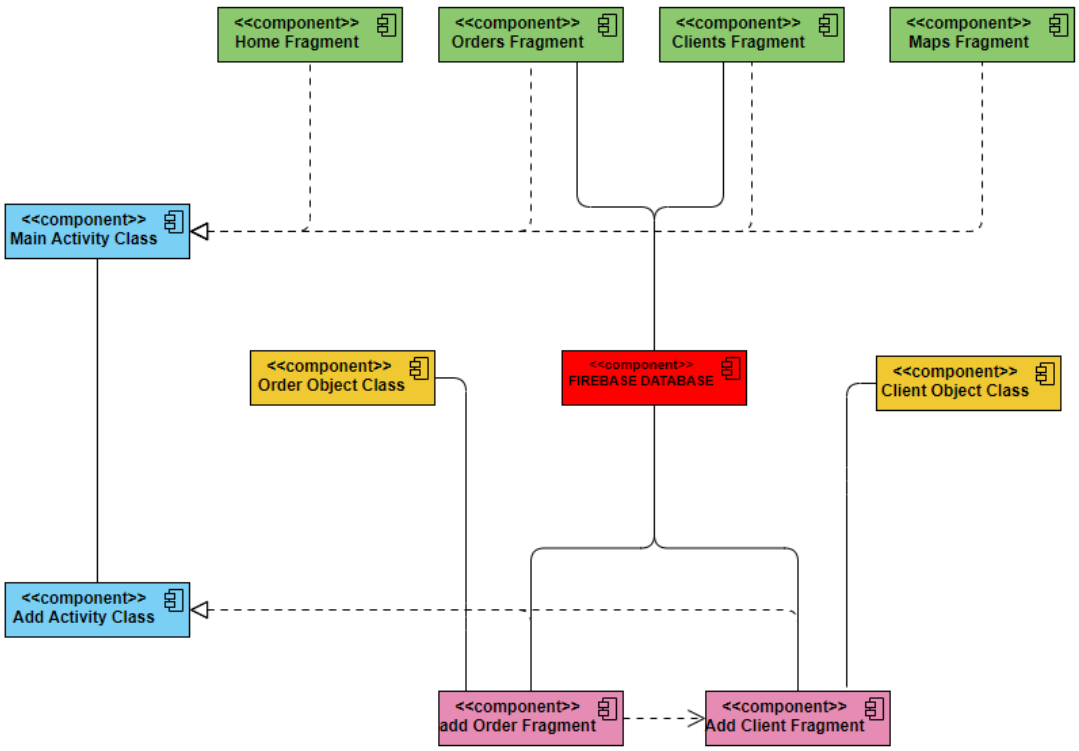
Criterion C – Development

Techniques Used

- Asynchronous Programming
- Error handling using try/catch
- Database Management (Firebase)
- Custom Data Classes
- For each loop
- If and switch statements
- Recyclerview
- Fragment Management
- Activity Management
- Material UI Components
- Intents
- Parsing (e.g. location)
- Google Maps SDK
- Geocoder
- ArrayList

Structure of the Program

Figure: UML Diagram Structure of the Program



I chose this type of program structure because of three main reasons: **Easy to develop, debug and maintain**. I chose to divide the Main activity and Add activity **classes** because of their different

functionality. The Add activity combines **fragments**, which allow the user to only add the orders and clients into the system, whereas the Main Activity combines fragments that contain all the other functions like deleting, calling, and handling map functions.

Classes: Orders and Clients

The empty constructors in both of the classes will be used to create the instance of the class and later on add the values to its variables. These classes will help to create **multiple instances** of the **object** that has the **same attributes**. This makes the process much easier.

```
package com.example.internallfinal.models;

public class Clients {
    private String name;
    private String number;
    private String location;

    public Clients(String location, String name, String number) {

        this.name = name;
        this.number = number;
        this.location = location;

    }

    public Clients() {

    }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public String getNumber() { return number; }

    public void setNumber(String number) { this.number = number; }

    public String getLocation() { return location; }

    public void setLocation(String location) { this.location = location; }
}
```

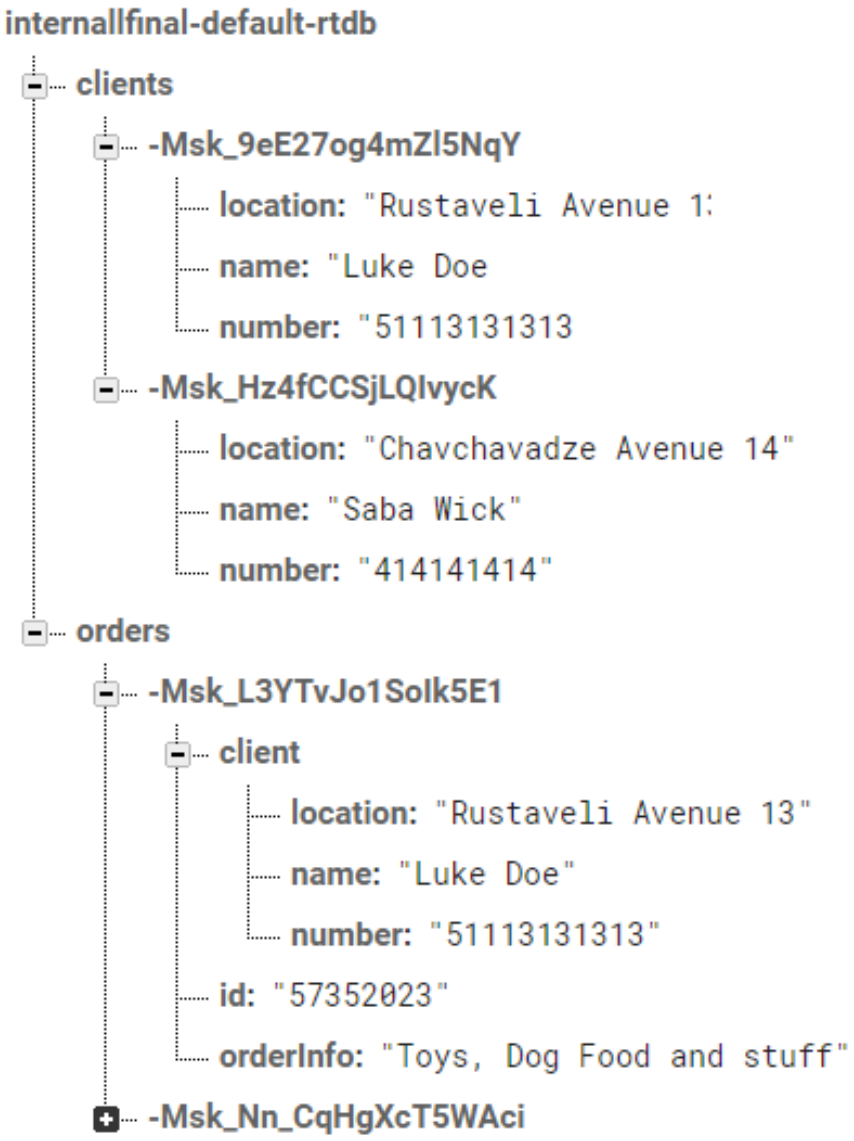
Here we have three String variables that will hold the name of the client, the number of the client, and the address of the client.

```java
package com.example.internallfinal.models;

public class Orders {
    private String id;
    private Clients client;
    private String orderInfo;

    public Orders(String id, Clients client, String orderInfo) {
        this.id = id;
        this.client = client;
        this.orderInfo = orderInfo;
    }

    public Orders() {
    }

    public String getId() { return id; }

    public void setId(String id) { this.id = id; }

    public Clients getClient() { return client; }

    public void setClient(Clients client) { this.client = client; }

    public String getOrderInfo() { return orderInfo; }

    public void setOrderInfo(String orderInfo) { this.orderInfo = orderInfo; }
}
```

Here we have two String variables that will hold the id of the order, the info of the order, and one Clients class variable, which will hold the order's client.

Database Management

I chose Firebase Realtime Database because it can record the connection status and provide updates every time the database state changes. It also holds the data in the format of JSON, which I am familiar with. The database looks like this:

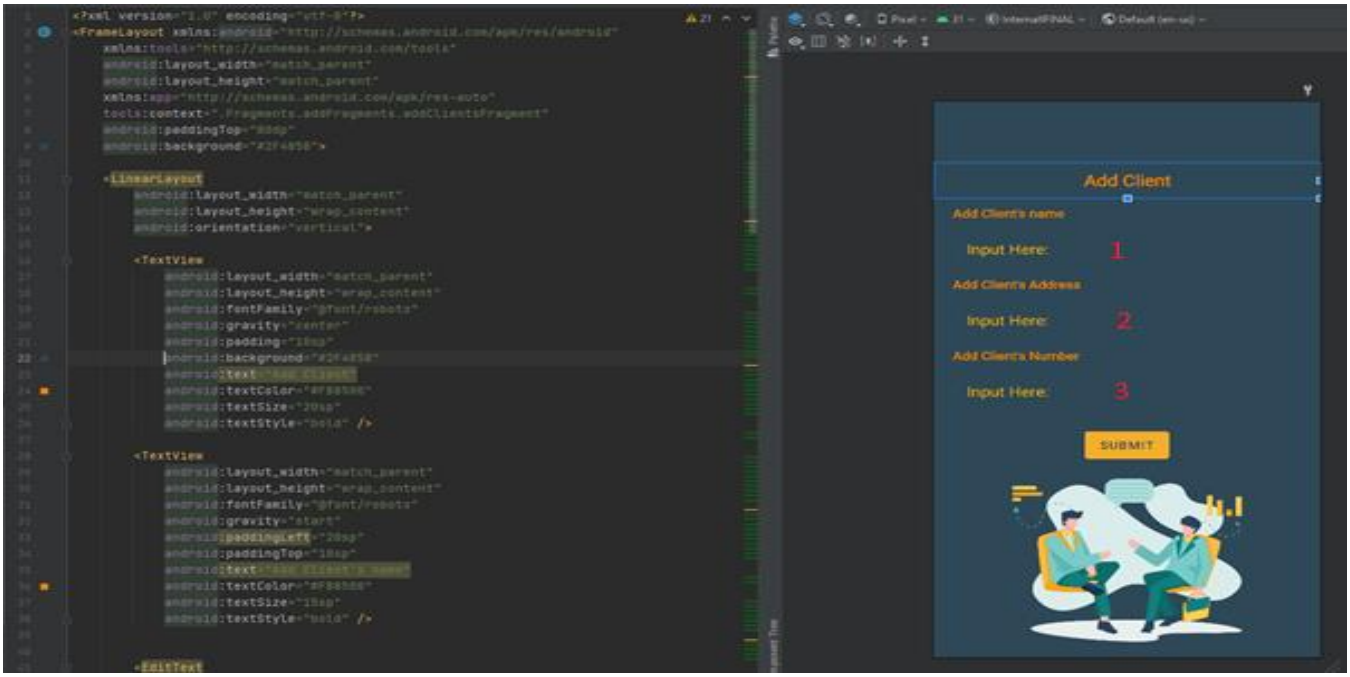


The database has two **child elements**: clients and orders, which contain the list of clients and the list of orders. The client has in this case **two child elements** but it can add up to many more. Each **child element** of clients has a **unique Id** that represents the **instance** of a client. Each of the child element has the value that was inputted by the user.

Input Data into Firebase

The user inputs data from AddOrdersFragment and AddClientsFragment.

Screenshot: addClientsLayoout.xml

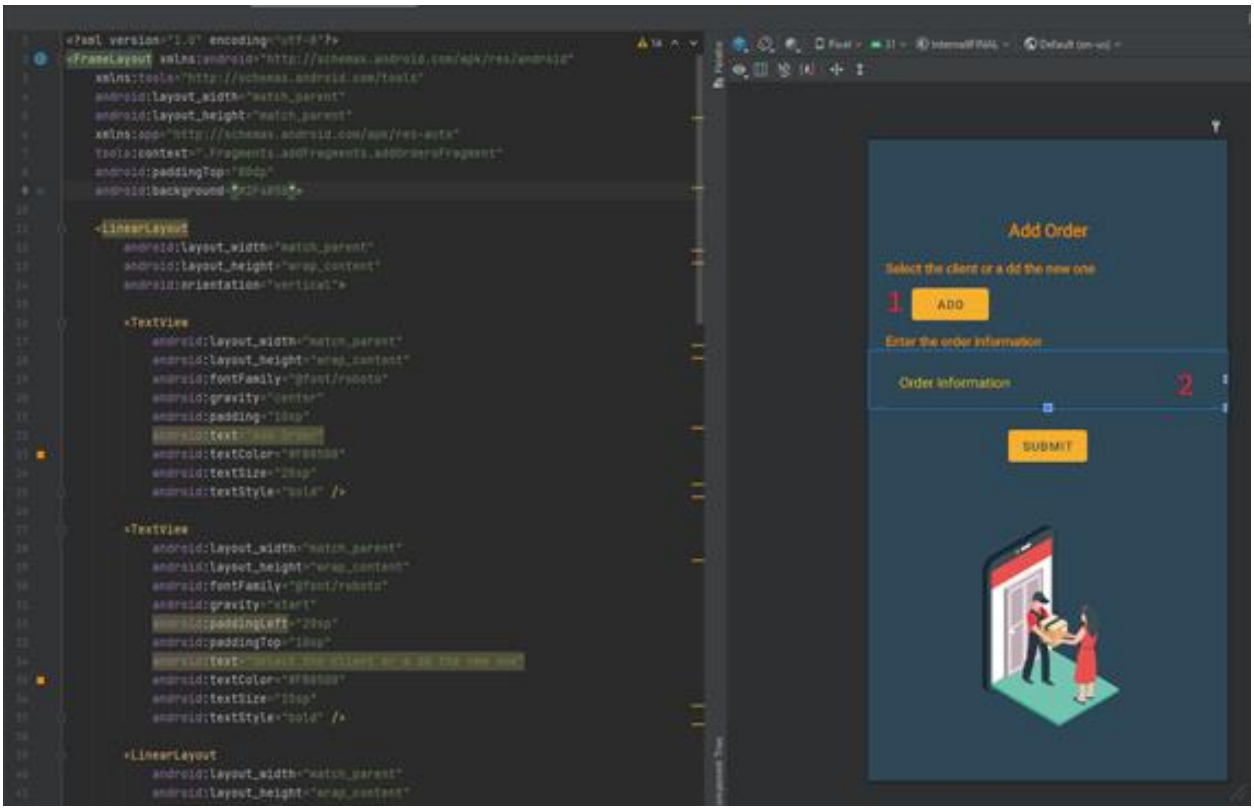


Client Input: User first inputs 1. name, 2. location, and 3. number, and these inputs are assigned to variables which are ultimately combined In a **Clients Object** and **pushed** to the database. I use the **push()** method to insert data because it automatically generates a **unique** Key for the Client in the database as a **root** value on a **button click**.

Screenshot: addClientsFragment.class

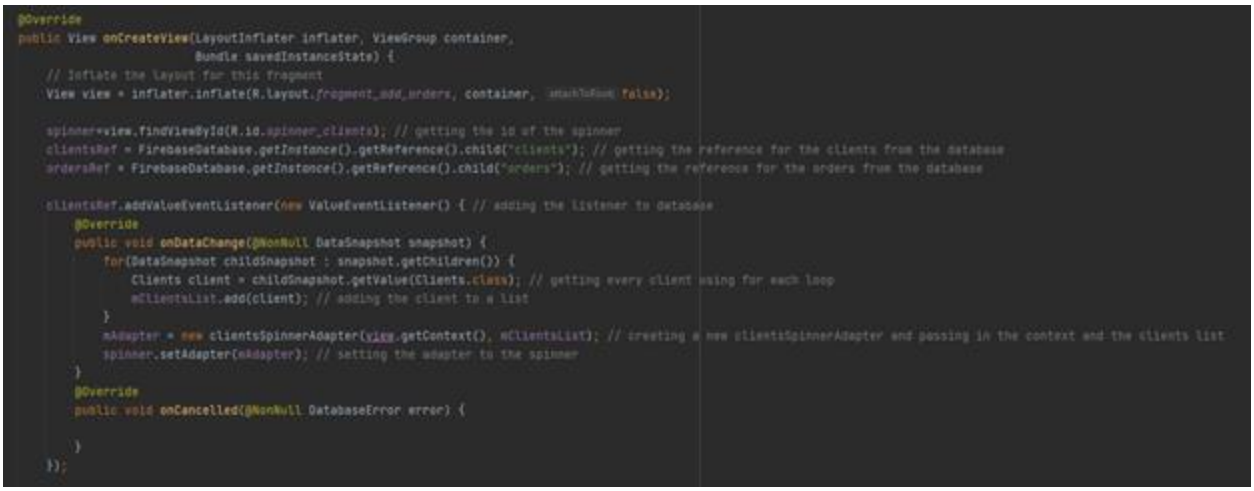
```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
    // Inflate the layout for this fragment
    View view =  inflater.inflate(R.layout.fragment_add_clients, container,  attachToRoot: false);

    clientsRef = FirebaseDatabase.getInstance().getReference().child("clients"); // getting the reference for the clients from the database

    submitButton = view.findViewById(R.id.submit_client); // getting the id of the submit button
    submitButton.setOnClickListener(new View.OnClickListener() { // setting the on click listener on the submit button
        @Override
        public void onClick(View v) {
            EditText clientNameInput = view.findViewById(R.id.clientNameInput); // getting the client name input id
            String name = clientNameInput.getText().toString(); // getting the input from the client name edit text

            EditText clientLocationInput = view.findViewById(R.id.clientLocationInput); // getting the client location input id
            String location = clientLocationInput.getText().toString();// getting the input from the client location edit text

            EditText clientNumberInput = view.findViewById(R.id.clientNumberInput); // getting the client number input id
            String number = clientNumberInput.getText().toString(); // getting the input from the client number edit text

            Clients client = new Clients(location, name, number); // creating a new client

            clientsRef.push().setValue(client); // pushing the new client into the database

            Intent intent = new Intent(view.getContext(), MainActivity.class); // creating the intent to go from addActivity to mainActivity
            startActivity(intent); // initializing intent
        }
    });

    return view;
}
```

Screenshot: addOrdersLayout xml



Order Input: The user first has to choose the client from the 1. **spinner**, which using a **custom spinner adapter class** gets all the clients from the database, adds it into the **ArrayList** which is the Clients type, and ultimately inputs all the elements from the **ArrayList** into the spinner, and then 2. add order information. The inputs are assigned to **variables**, combined into **Orders Object**, which gets **pushed** into the database on a **button click**.

Screenshot: addOrdersFragment class



```
submitButton = view.findViewById(R.id.submit_order); // getting the id of submit button
submitButton.setOnClickListener(new View.OnClickListener() { // setting the on click listener to submit button
    @Override
    public void onClick(View v) {
        TextInputLayout infoInput = view.findViewById(R.id.orderInfoInputLayout); // getting the id of info input
        String info = infoInput.getEditText().getText().toString(); // getting the text fron info input and saving that to a string
        String shortId = RandomStringUtils.randomNumeric( count: 8); // generating a random short id using RandomStringUtils
        Orders order = new Orders(shortId, selectedClient, info); // creating a new order object
        ordersRef.push().setValue(order); // pushing the order to the ddatabase referebce that we  got earlier
        Intent intent = new Intent(view.getContext(), MainActivity.class); // creating the intent to go from addactivity to main activity
        startActivity(intent); // intilizaing the intent
    }
});

addClient = view.findViewById(R.id.add_clients_order); // getting the id of add client button
addClient.setOnClickListener(new View.OnClickListener() { // setting the listener to add client button
    @Override
    public void onClick(View v) { ((addActivity)getActivity()).replaceFragment(); } // on click we initilaize the method created in addActivity
});

return view;
```

Screenshot: clientsSpinnerAdapter class

```
package com.example.internallfinal.Adapters;

import ...

public class clientsSpinnerAdapter extends ArrayAdapter<Clients> {

   public clientsSpinnerAdapter(Context context, ArrayList<Clients> clientsList) {
       super(context, 0, clientsList);
   }

    @NonNull
    @Override
    public View getView(int position, @Nullable View convertView, @NonNull ViewGroup parent) {
        return initView(position, convertView, parent);
    }

    @Override
    public View getDropDownView(int position, @Nullable View convertView, @NonNull ViewGroup parent) {
        return initView(position, convertView, parent);
    }

    private View initView(int position,View convertView, ViewGroup parent) {
       if (convertView== null) {
           convertView = LayoutInflater.from(getContext()).inflate(R.layout.clients_spinner_row, parent,  false); // infating view with the created layout
       }
       TextView name = convertView.findViewById(R.id.client_view_name); // getting the id of the name in from the layout

       Clients currentClient = getItem(position); // getting current client by getting the position of the item

       if (currentClient!= null) {
           name.setText(currentClient.getName()); // getting the name from the client and setting that to layout

       }

       return convertView;
   }
}
```

Output Data from Database

The program outputs data **asychroniosly** into ordersFragment, clientsFragment and mapsFragment using **onDataChange()**. This makes sure that whenever data is added or removed from the database, it is instantly reflected in the app. Both in ordersFragment and clientsFragment data is outputted into a **recyclerview.** Whereas in mapsFragment data is outputted inside Maps as individual Markers.

For ordersFragment and clientsFragment we use **Firebase Recycler Options** to pass **database reference**, the data we want to get, and in which type we want to read the data. **Firebase Recycler Options** is used because it enables implementing **FirebaaseRecyclerAdapter** that handles automatically populating **recyclerview**. The data from the database is read based on the unique **key** that was generated from the push() method.

Screenshot: ordersFragment class

```
protected void onBindViewHolder(@NonNull ordersFragment.ordersViewHolder holder, int position, @NonNull Orders model) {

    String orderId = getRef(position).getKey(); // we get the unige key of order by referencing the database and passing the position which is set as an argument in o
    ordersRef.child(orderId).addListenerForSingleValueEvent(new ValueEventListener() { // we add the listener for ordersRef and passin the unique key to listen to the
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            String id = snapshot.child("id").getValue().toString(); // we get the id in strings
            String infoString = snapshot.child("orderInfo").getValue().toString(); // we get the order info in strings
            String locationString = snapshot.child("client").child("location").getValue().toString(); // and we get the location of the order in strings by accessing

            holder.id.setText("Order Id: " + id); // setting the order it text
            holder.info.setText(infoString); // setting the info text
            holder.location.setText(locationString); // setting the text
        }

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

        }
    });
```

Screenshot: clientsFragment class

```
@Override
public void onStart() { // on start method of fragment
    super.onStart();
    FirebaseRecyclerOptions options = new FirebaseRecyclerOptions.Builder<Clients>()
            .setQuery(clientsRef, Clients.class).build();

    FirebaseRecyclerAdapter<Clients, clientsViewHolder> adapter = new FirebaseRecyclerAdapter<Clients, clientsViewHolder>(options) {
        @Override
        protected void onBindViewHolder(@NonNull clientsViewHolder holder, int position, @NonNull Clients model) {

            String clientId = getRef(position).getKey(); // getting car id from database by getting the key

            clientsRef.child(clientId).addListenerForSingleValueEvent(new ValueEventListener() { // adding listener to listen for data change in car in database
                @Override
                public void onDataChange(@NonNull DataSnapshot snapshot) {
                    String nameString = snapshot.child("name").getValue().toString(); // getting numberplate from database
                    String numberString = snapshot.child("number").getValue().toString(); // getting entrance date from database
                    String locationString = snapshot.child("location").getValue().toString(); // getting color from database


                    holder.name.setText(nameString);
                    holder.number.setText(numberString);
                    holder.location.setText(locationString);
                }
                @Override
                public void onCancelled(@NonNull DatabaseError error) { }
            });
```

For mapsFragment, we use **for each loop** to iterate through every child element of orders inside the database to **read** and assign to **variables**. The address is checked for errors using the **if statement** and **try/catch.** After checking the location new **LatLng type data** is created using **get()** methods, **geocoder,** and added to Google Maps marker with other **variables**.

Screenshot: mapFragment class

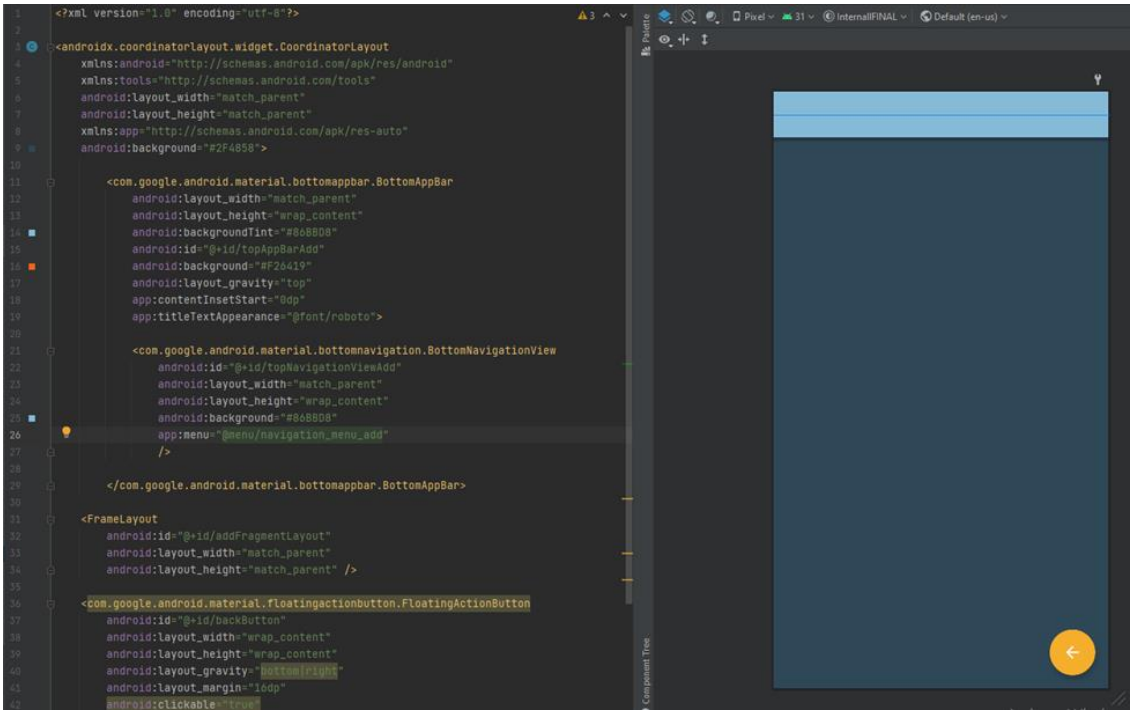
```
if(location==null) {
    Toast.makeText(getContext(), text: "Address Not Found!", Toast.LENGTH_SHORT).show();
} else { // creating goecoder
    Geocoder geocoder = new Geocoder(getContext(), Locale.getDefault());
    try {
        List<Address> listAddresss = geocoder.getFromLocationName(location, maxResults: 1);
        if(listAddresss.size()>0) {
            Address address = listAddresss.get(0); // turnign into address
            LatLng latLng = new LatLng(address.getLatitude(), address.getLongitude()); // turning adress into latLng using get methods
            MarkerOptions markerOptions = new MarkerOptions(); // creating markerOptions
            markerOptions.title(name + "'s order: " + location); // setting title to the marker
            markerOptions.position(latLng); // setting marker position to the latlng we created
            googleMap.addMarker(markerOptions); // addding marker to google maps
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

```
ordersRef = FirebaseDatabase.getInstance().getReference().child("orders"); // getting the database reference for orders
ordersRef.addValueEventListener(new ValueEventListener() { // setting value listener on clients in database
    @Override
    public void onDataChange(@NonNull DataSnapshot snapshot) {
        for (DataSnapshot childSnapshot : snapshot.getChildren()) { // getting every single order

            String location = childSnapshot.child("client").child("location").getValue(String.class); // getting location fo the order
            String name = childSnapshot.child("client").child("name").getValue(String.class); // getting name of the order
```

UI Layout

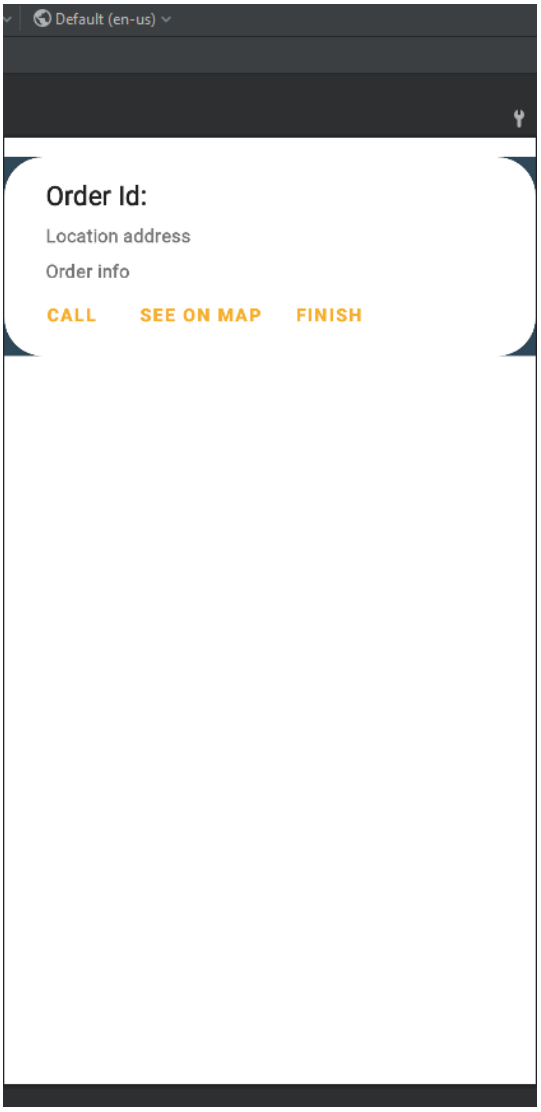
For layout components, I used Material Design and its components library to use pre-defined elements like navigation app bars, buttons, input fields, etc.

To create complex layout structures, I decided to divide them into two separate parts: main layout and child layouts. This structure makes it easy to apply changes and debug the child layout, as we only deal with a small part of the code. The main layout is the one that holds child layouts and these typically are fragment layouts. For example:

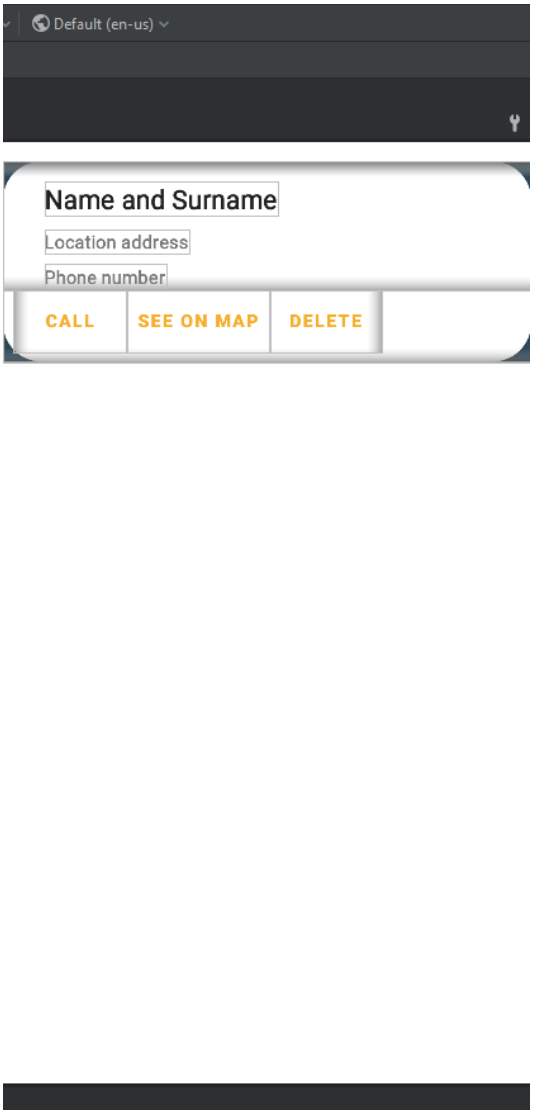


In the case of child layouts, there's an orders layout, which lays out the structure of the order, the client's layout, which lays out the structure of the client, and a navigation layout, which lays out the navigation bar layout.

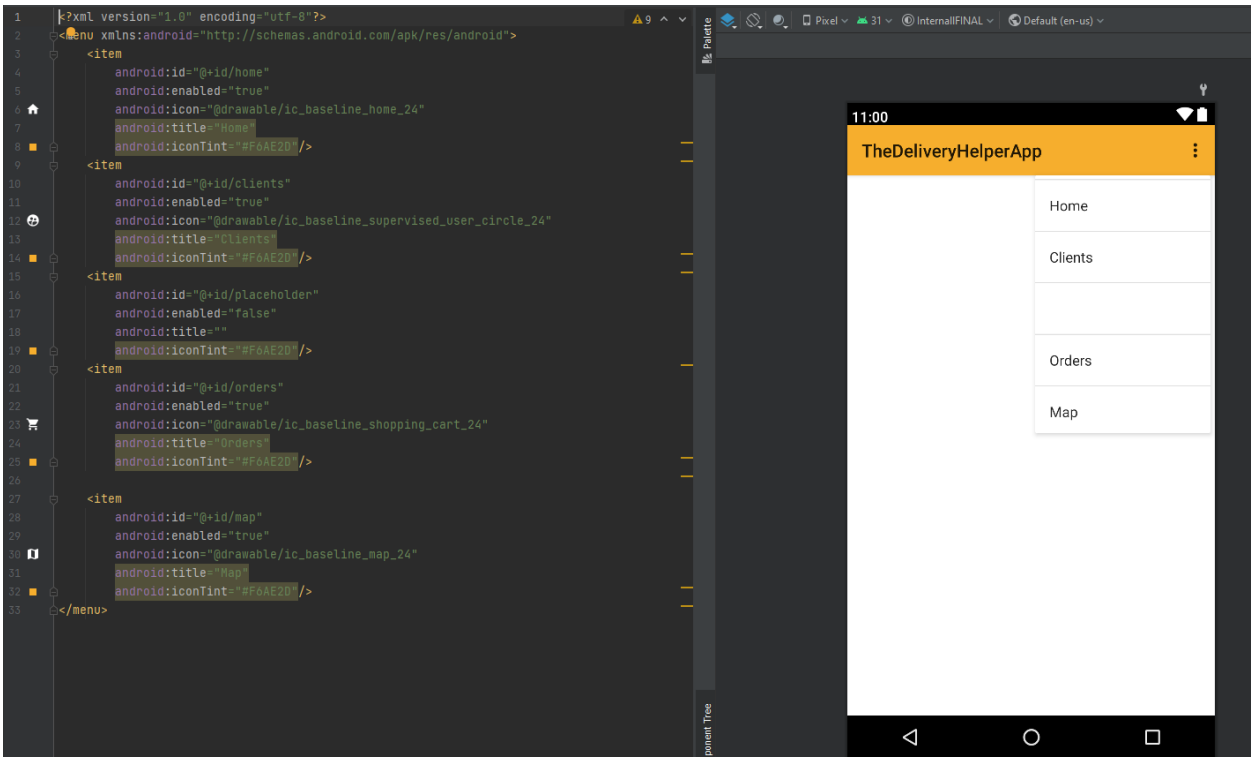
Screenshot: orderLaout.xml



Screenshot: clientLayout.xml



Screenshot: bottomNavigationLayout.xml



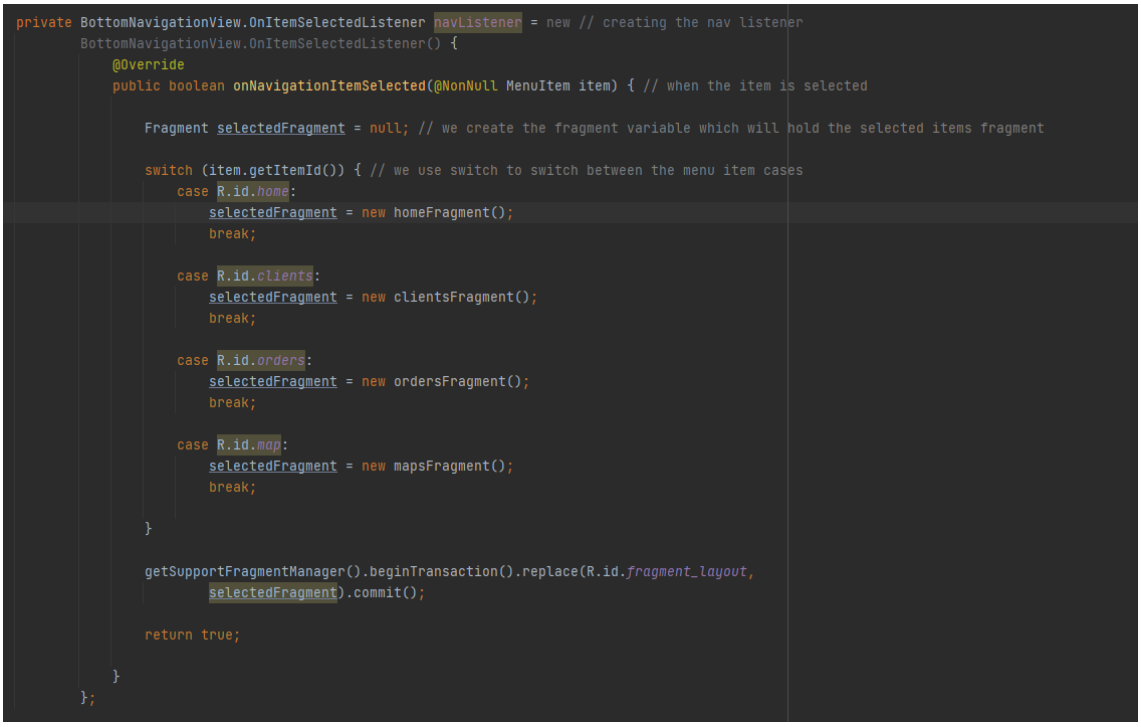
Navigation

Intents are used to navigate through activities. Based on which icon was clicked, the intent will be initiated from current activity to destined activity, thus changing the activity.

```
private void fabStartActivity() {
    FloatingActionButton myFab =findViewById(R.id.fab); // getting the view of floatingActionButton
    myFab.setOnClickListener(new View.OnClickListener() { // setting the onclick listener to handle the click
        public void onClick(View v) {

            Intent intent = new Intent( packageContext: MainActivity.this, addActivity.class); // creating the intent to change activiyu from main to add
            startActivity(intent); // initializing the intent

        }
    });
}
```

To navigate through fragments, the switch statement is used because there is a single expression that changes the fragment, but the fragment Is changeable.

Screenshot: mainActivity.class

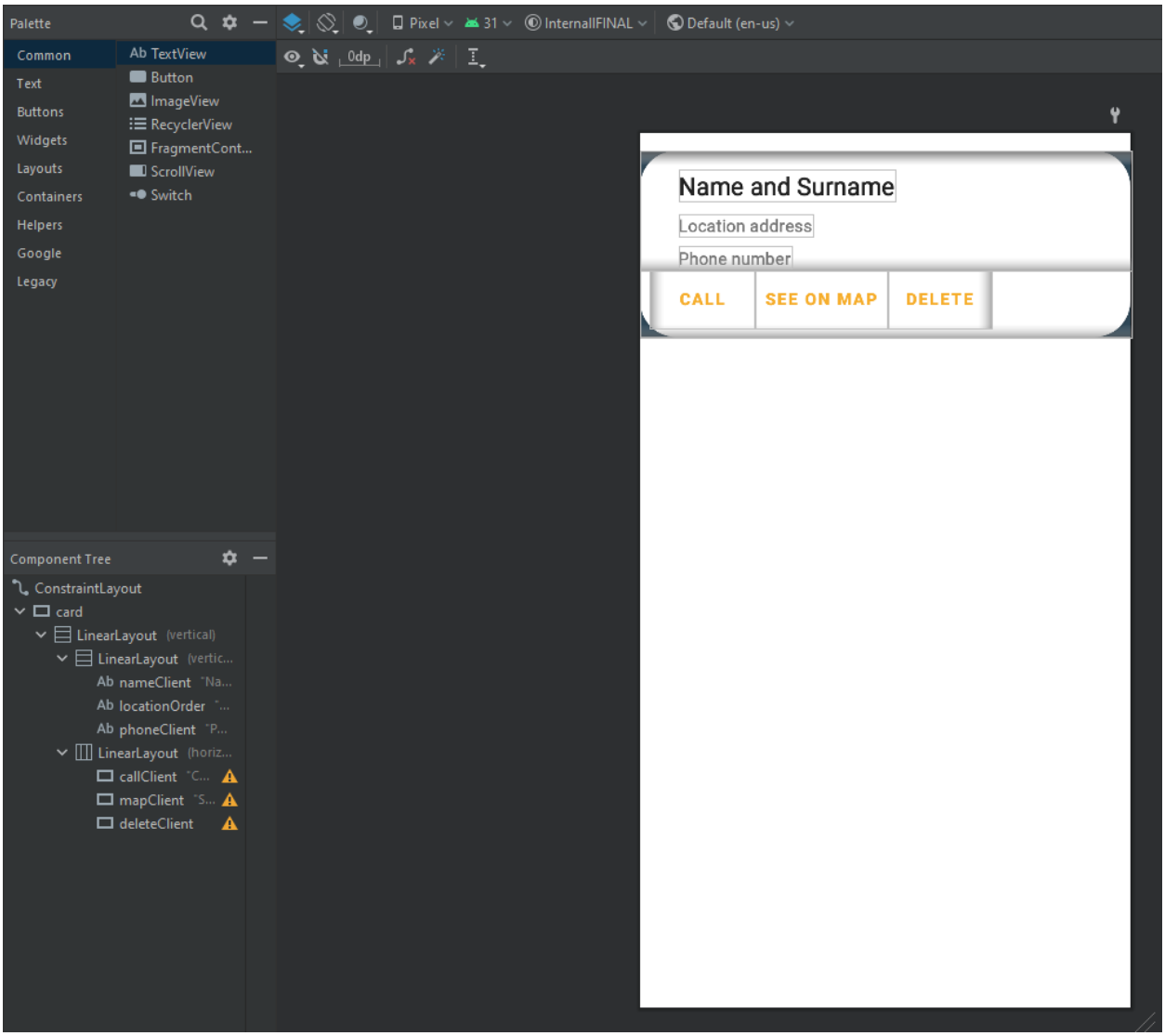


```
private BottomNavigationView.OnItemSelectedListener navListener = new // creating the nav listener
        BottomNavigationView.OnItemSelectedListener() {
            @Override
            public boolean onNavigationItemSelected(@NonNull MenuItem item) { // when the item is selected

                Fragment selectedFragment = null; // we create the fragment variable which will hold the selected items fragment

                switch (item.getItemId()) { // we use switch to switch between the menu item cases
                    case R.id.home:
                        selectedFragment = new homeFragment();
                        break;

                    case R.id.clients:
                        selectedFragment = new clientsFragment();
                        break;

                    case R.id.orders:
                        selectedFragment = new ordersFragment();
                        break;

                    case R.id.map:
                        selectedFragment = new mapsFragment();
                        break;

                }

                getSupportFragmentManager().beginTransaction().replace(R.id.fragment_layout,
                        selectedFragment).commit();

                return true;

            }
        };
```

Other functionality

According to success criteria, the user should be able to remove the clients/orders, make calls from the app and see the clients/orders location on the integrated map.

Screenshot:clientLayout.xml



To make the call, we first listen to the call button click using onClickListener(). In the listener, we get the number in String type from the database, which is referenced to the specifical client in the firebase using the unique key, that is generated by the push() method. We then take the number and use Intent and Uri. parse to initiate the call. To delete the client or finish the order, we first listen to the delete/finish button click and using the removeValue() method we remove the order or client from the database based on the unique key of the element.

Screenshot: clientsFragment.xml

```
holder.callButton.setOnClickListener(onClick(v) -> {
        clientsRef.child(clientId).addListenerForSingleValueEvent(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                String number = snapshot.child("number").getValue(String.class);
                Intent callIntent = new Intent(Intent.ACTION_DIAL);
                callIntent.setData(Uri.parse("tel:"+number));
                startActivity(callIntent);
            }
            @Override
            public void onCancelled(@NonNull DatabaseError error) { }
        });
});

holder.deleteButton.setOnClickListener(onClick(v) -> {
        clientsRef.child(clientId).removeValue();
});
```

To see the location of the order/client on the map, we use the same method as in the mapsFragment, but this time we don't use for each loop, we use the single address that is read from the firebase database in String type.

```
holder.mapButton.setOnClickListener(new View.OnClickListener() { // setting the on click listener on map button
    @Override
    public void onClick(View v) {

        ordersRef.child(orderId).child("client").addListenerForSingleValueEvent(new ValueEventListener() { // setting a value listener on orders client
            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                String location = snapshot.child("location").getValue(String.class); // getting the location of the client from the databsae
                if(location==null) {
                    Toast.makeText(getContext(), text: "Address Not Found!", Toast.LENGTH_SHORT).show(); // toasting if the address equals to null
                } else {
                    Geocoder geocoder = new Geocoder(getContext(), Locale.getDefault()); // creating the geocoder
                    try {
                        List<Address> listAddresss = geocoder.getFromLocationName(location, maxResults: 1); // creating the list of addresses for our adress
                        if(listAddresss.size()>0) {
                            Address address = listAddresss.get(0); // we get the address by getting the first element from the list
                            Uri uri = Uri.parse( // we parse the uri by follwoing parsing sequence
                                    "geo:"    + address.getLatitude() +
                                            ","       + address.getLongitude() +
                                            "?q="     + address.getLatitude() +
                                            ","       + address.getLongitude() +
                                            "("       + address + ")");
                            Intent in = new Intent(Intent.ACTION_VIEW, uri); // creating the intent to open up google maps
                            startActivity(in); // initializing the intent
                        }
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
            }

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

            }
        });
    }
});
```

Word Count: 1056

Bibliography

- *How to Add a Floating Action Button to a Bottom Navigation - Android Studio Tutorial*. *YouTube*. Philipp Lackner, 2020. https://www.youtube.com/watch?v=x6_va1R788&ab_chanhttps%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv=x6-_va1R788&ab_channel=PhilippLacknernel.

- *Firebase Realtime Database Android Endless Scroll Pagination with Recyclerview Example #2*. *YouTube*. Cambo Tutorial, 2021. https://www.youtube.com/watch?v=jIGxgA_wHvg&t=41s&ab_channel=CamboTutorial.

- "Material Components." Material design. Accessed April 1, 2022. https://material.io/components?platform=android.

- "Custom Spinner - Android Studio Tutorial." YouTube. Coding in Flow, December 16, 2017. https://www.youtube.com/watch?v=GeO5F0nnzAw.