Custom project final report

COS30017 Software Development for Mobile Devices 2023

Name: Trung Kien Nguyen
Student ID: 104053642

Table of Contents

Overview of project ***1***

Level 1: Design evidence ***2***

Level 2: App evidence ***5***

Level 3: Extended research evidence ***7***

References ***7***

Appendix 1 - Detailed architecture diagrams ***7***

Getting data from API **7**

Getting user data **7**

Getting user's previous transactions data **8**

Getting user wallet 's crypto balance data **8**

Buying cryptocurrency **8**

Selling cryptocurrency **9**

Appendix 2 - Weekly reports ***9***

Week 7 ..... **9**

Week 8 ..... **10**

Week 9 ..... **11**

Week 10 ..... **12**

Week 11 ..... **16**

Overview of project

My project is a cryptocurrency app using Android Development with Kotlin. Like many typical apps in the market, such as CoinGecko, Coinbase, Kraken, etc., my app has some basic functionalities as follows:

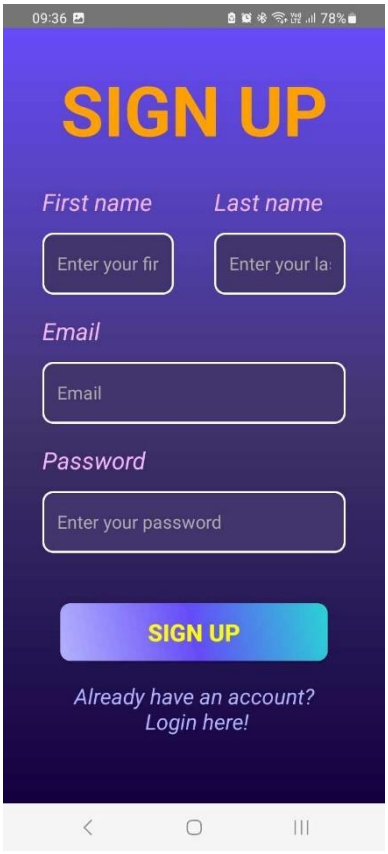
- User authentication: The app allows users to create accounts, log in with them, or reset their password in case of forgetting.
- Wallet managing: It also creates and manages cryptocurrency wallets for different cryptocurrencies, e.g. Bitcoin, Etherium, ….
- Viewing transaction history
- Buy and sell cryptocurrencies as will.

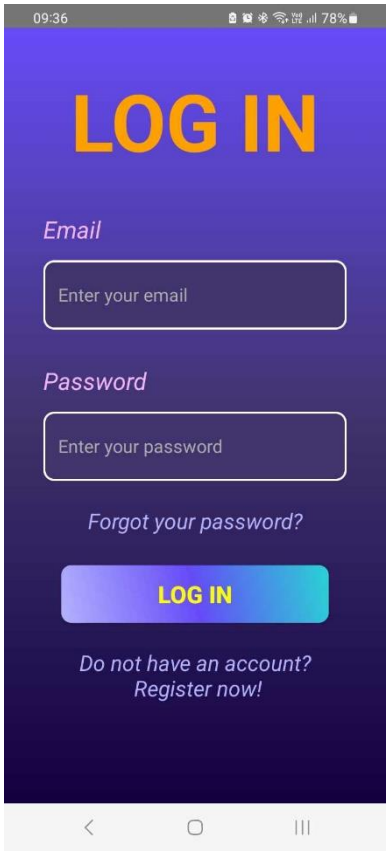
GitHup repository link: https://github.com/SoftDevMobDev-2023-Classrooms/customproject1-trungkiennguyen22082004

Level 1: Design evidence

This following section includes some sketches and basic functionality of the main activities for my app:

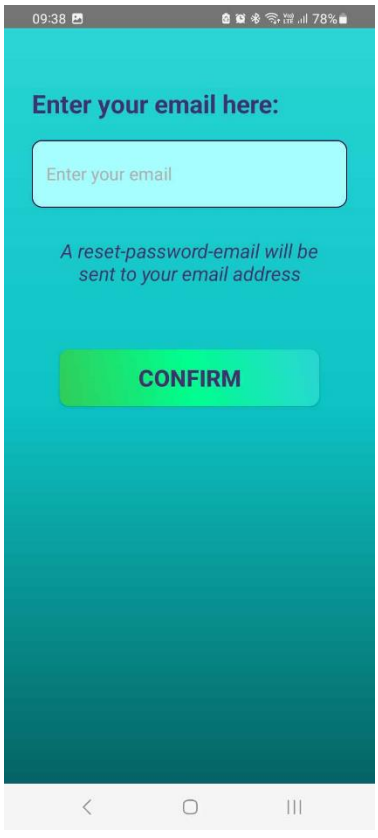
- Login Activity and Signup Activity:



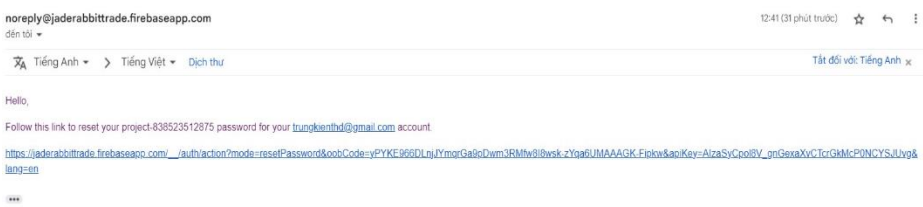


These activities are responsible for signing in and signing up purposes, using the similar layout and theme as the login activity. In each activity, there is a text at the bottom of the layout, linking to each other by clicking on it.

- Forgot Password Activity:



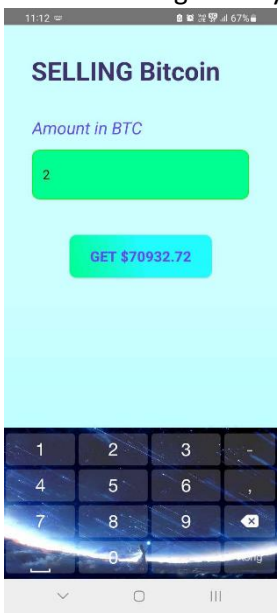
In the Login Activity, when clicking on the text “Forgot your Password?”, it will lead this activity. This includes an EditText view for the user to enter his/her email. When clicking on the Confirm button, an resetting email is sent to then enterred email address. E.g:

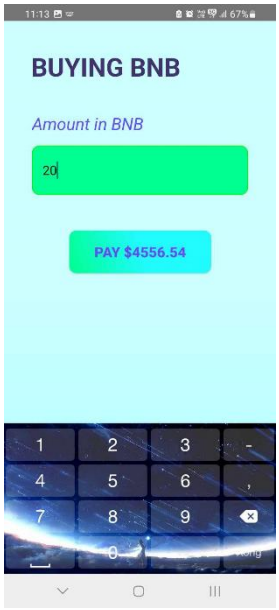


- Home Activity:

This is the base activity of my app, containing the list of purchasable cryptocurrencies in a Recycler View. This has a searching (in the menus) and sorting function (clicking on the buttons of Hot/Price/24h Change).
Each item can be clicked on, leading to Trading Activity to buy the coresponding cyptocurrency

- Trading Activity:





This is a simple-design activity for the user to buy/sell cryptocurrency.

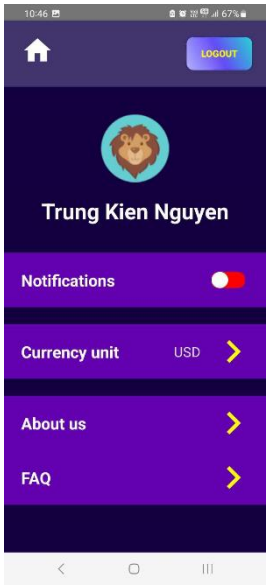
- Profile Activity:

This contains some information of the User, including his/her full name, avatar, total balance, and most importantly, the history of transaction as a Recycler View.

- Wallet Activity:

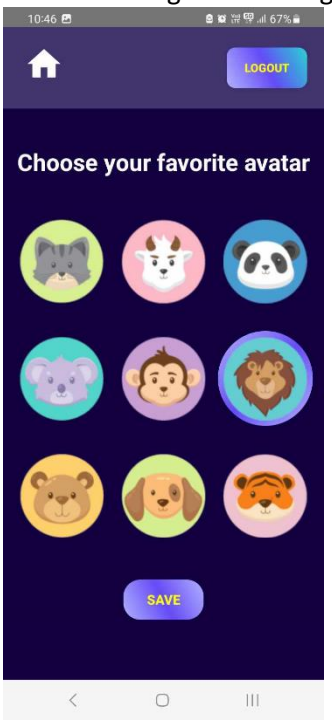
This has a Recycler View, designed for the user to view all of their cryptocurrencies in his/her wallet. User can also click on each item to go to Trading Activity to sell his/her corresponding cryptocurrency

- Settings Activity:



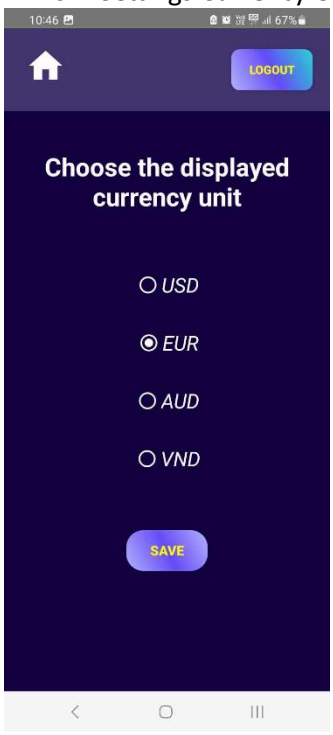
This activity is created for app settings, such as Profiles editting, Notifications allowing, Currency display, Information about the fictional organization, some FAQs, and a Logout buttons.
The avatar ImageView, fullname TextView, purple rectangular ConstraintLayouts (notifications, curency unit, about us, FAQ) are belongs to a fragment (SettingsFragment), which leads to other fragments such as SettingsAvatarFragment, SettingsCurrencyUnitFragment:

  - Settings Avatar Fragment:



User can change their avatar by choosing the favourite one, then clicking on the Save button.

- Settings Currency Unit Fragment:



This fragment has a Radio Group, in which the user can choose their favorite display of currency unit in everywhere of the app (user's total balance, price of the cryptocurrencies, …)

Level 2: App evidence

- Overall Architecture Design (see Appendix 1 – Detailed Architecture Diagrams for detailed design of my app):

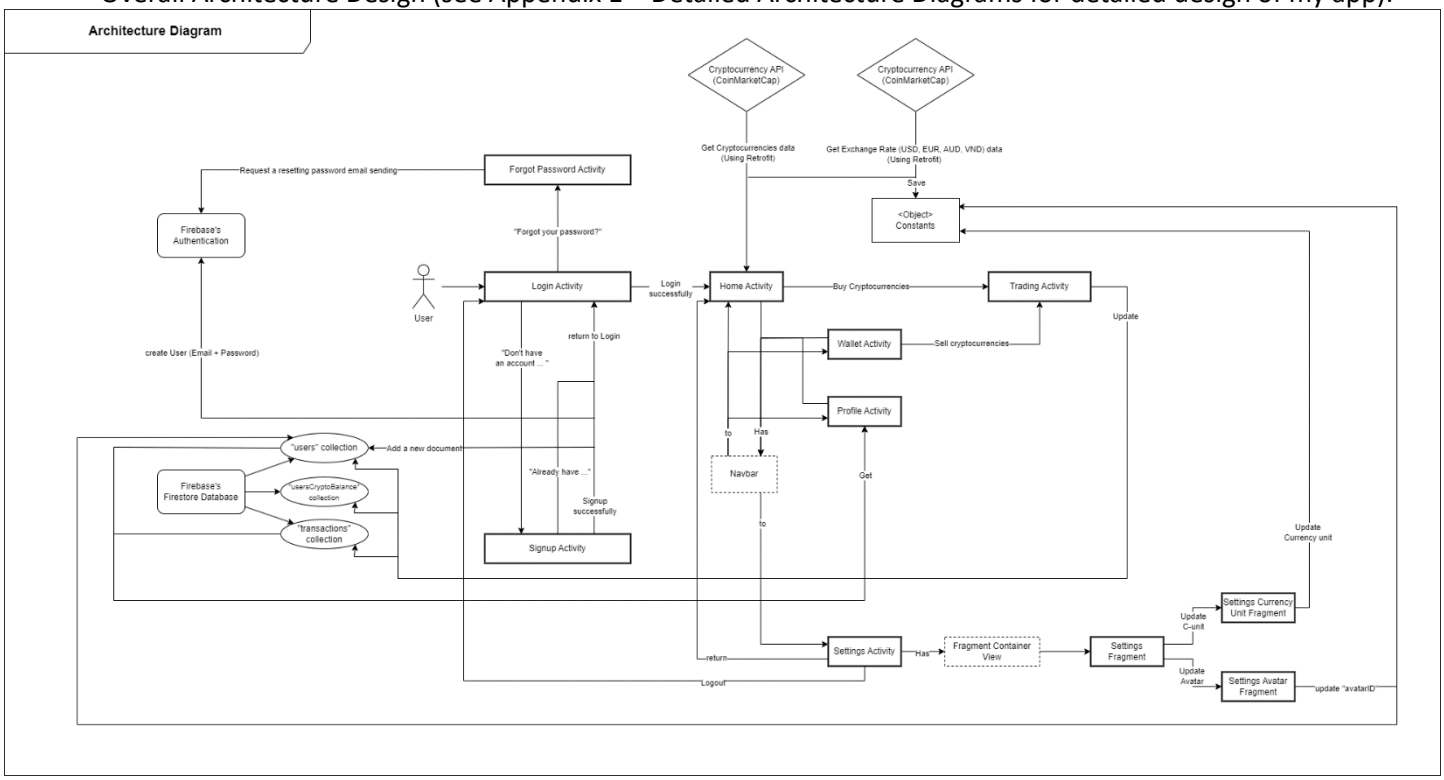


Fig 1. JRTrade's Overall Architecture Design Diagram

- Advanced UI components:
  - Navbar: This is made based on the UI Design Pattern of Navigation Tab mentioned in my Assignment Extension 1. The activities of Home, Profile and Wallet has a navbar, containing the buttons linked to each others, and the Settings Activity. The navbar's layout is in the file "floating_bar.xml", added to the activities by the element "include":

```
<include layout="@layout/floating_bar"/>
```

- RecyclerView: This technique is applied many places in my app, including the HomeActivity (displaying the cryptocurrencies' detail get from the CoinMarketCap's API), WalletActivity (displaying the balance of each cryptocurrency in the user's wallet get from the Firestore's collection of "usersCryptoBalance"), and Profile Activity (displaying the previous transaction get from the Firestore's collection of "transactions")
- Menus in HomeActivity with search function: This is a menus with a search item in the layout (considered as a SearchView), I then used the "setOnQueryListener()" method, and the override method of "onQueryTextChange()" inside, to search for the cryptocurrencies.
- Combining of many different activities and fragments mentioned in the "Level 1" section. The fragments in the Settings Activity are navigated among each others by a navigation XML resource file named "main_nav_graph.xml" in the "res/navigation" directory

- Functionality: I classify my app as a CRUD app, however, it also satisfies some requirements of a Master-Detail app mentioned in the Canvas
  - CRUD requirements:
    - I chose Firebase's Firestore as a database service for my app. My database include of 3 tables/collections, including:
      - "users" – storing users' data (name, email, avatarID, total balance (real currency)) (**See Fig 4 – Appendix 1**)
      - "usersCryptoBalance" – storing user's balance of cryptocurrencies in the wallet. (**See Fig 6 – Appendix 1**)
      - "transaction" – storing the detailed information about the users' previous transaction (buying/selling) (**See Fig 5, 7, 8 – Appendix 1**)
    - The field's data types are also various. They include most of the data types provided by Firestore, including number (int and double), string, timestamp.
    - Here is the diagram of my Firestore's database:

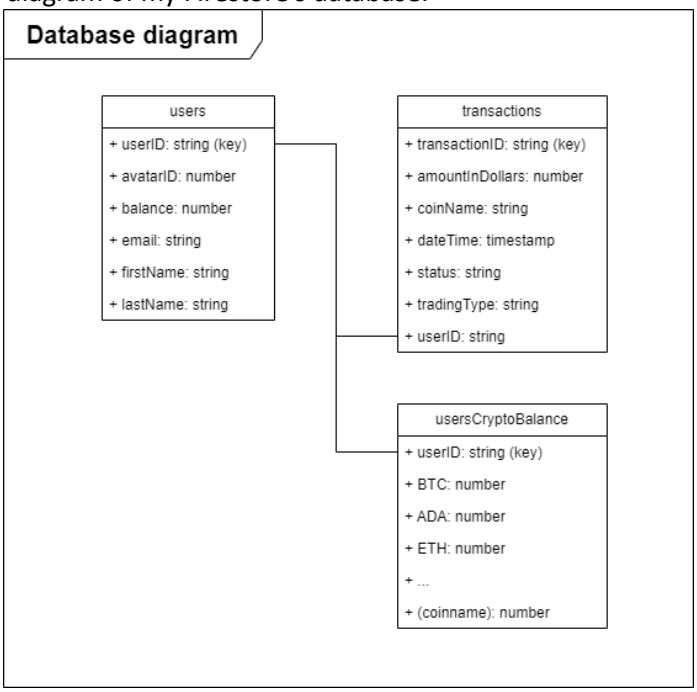


Fig 2. Firestore's database diagram

    - Firestore's database has an significant advantage that I can utilize: In a table/collection, each record/document can be different in terms of number of fields, different of data type in the dame field. E.g. For each user, the number of crypto currency in their wallet can be various.
    - While getting/updating the collections' data, I used try – catch statement of Kotlin to handling errors (In the event of error(s) happening, the error(s) will be shown in the Logcat)

  - I have also used "publicly available APIs". One of them is the Cryptocurrency data's API get from CoinMarketCap, which is used in the RecyclerView's items in the HomeActivity. Another used API in my app is from Exchange Rates API (https://exchangeratesapi.io/), the realtime data of exchange rates (USD, AUD, VND) compared to EUR, which is then manually converted to exchange rates compared to USD after being get using Retrofit, and used for the functionality of "Displaying price in other currency type". **See Fig 3. Diagram of getting data from API – Appendix 1**

Level 3: Extended research evidence

References

- Google. (2020). Documentation | Android Developers. https://developer.android.com/docs
- *Retrofit*. (2013). Github.io. https://square.github.io/retrofit/
- Google. (2019). Firebase Authentication | Firebase. https://firebase.google.com/docs/auth
- Google. (2019). Cloud Firestore | Firebase. https://firebase.google.com/docs/firestore

Appendix 1- Detailed architecture diagrams

Getting data from API

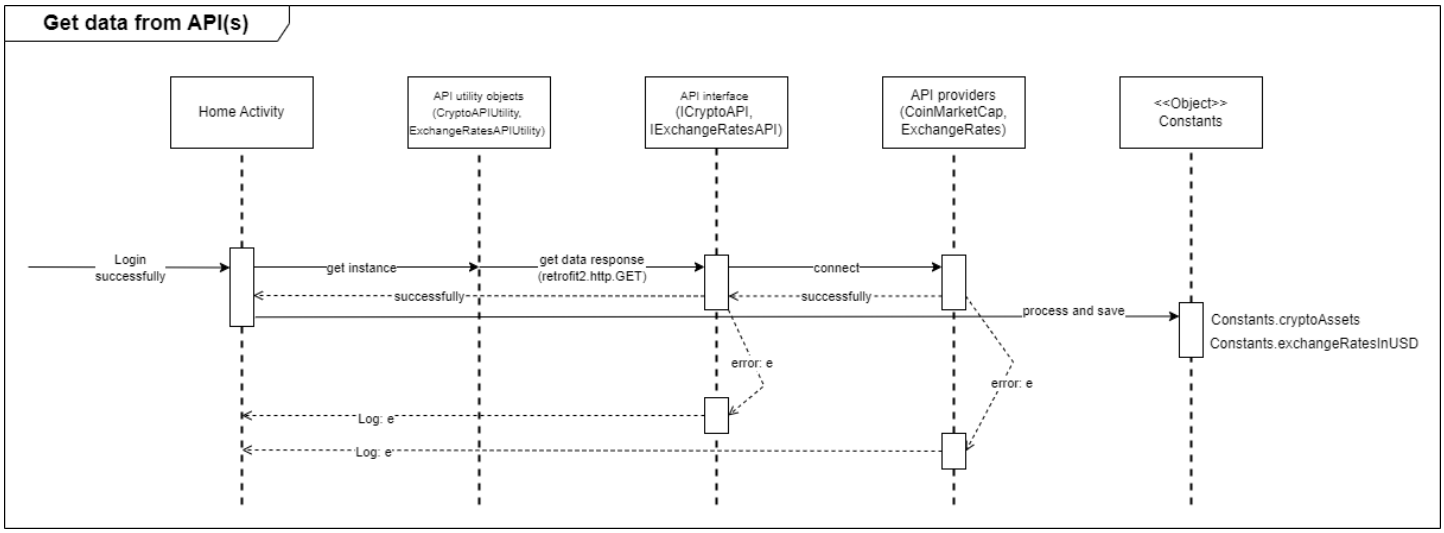


Fig 3. Diagram of getting data from API

Getting user data

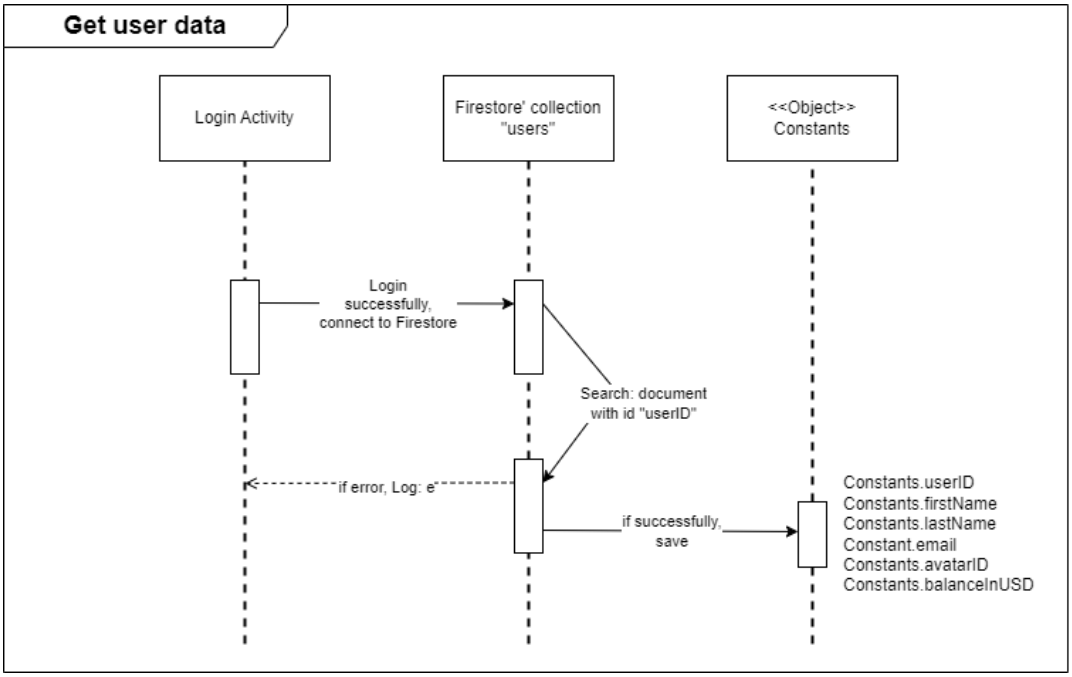


Fig 4. Diagram of getting user data

Getting user’s previous transactions data

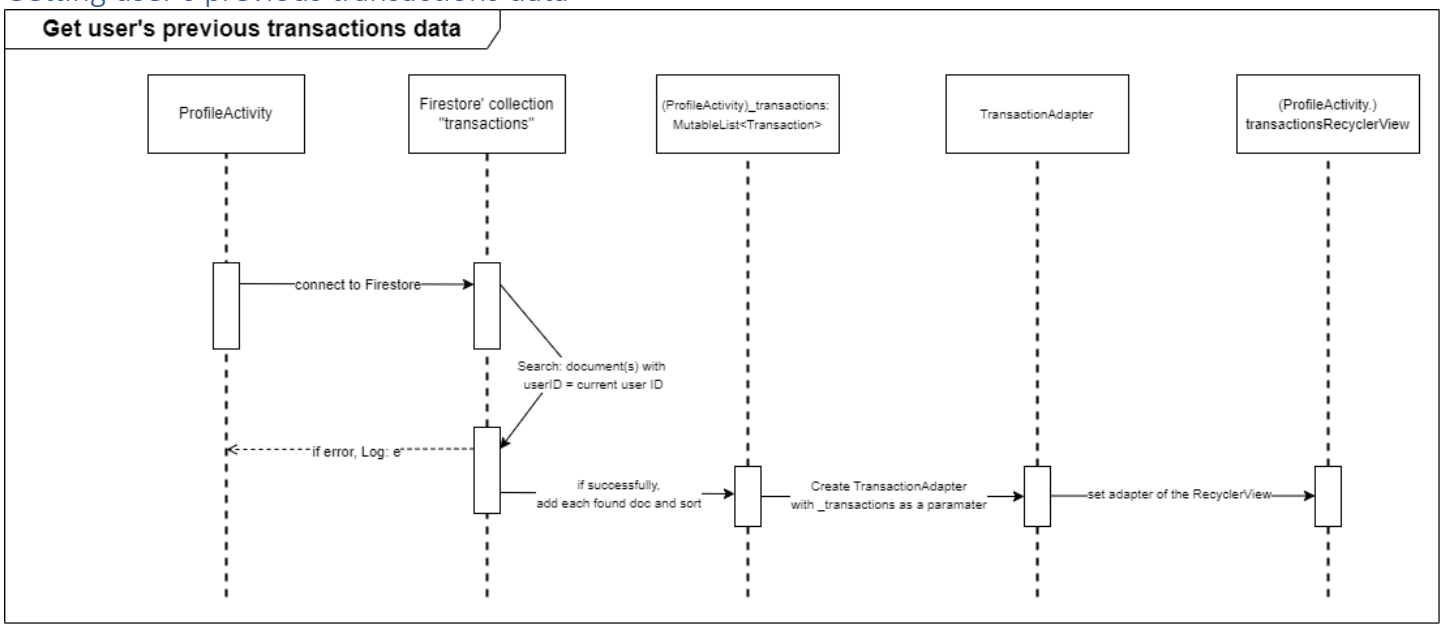


Fig 5. Diagram of getting user’s previous trnasactions data

Getting user wallet 's crypto balance data

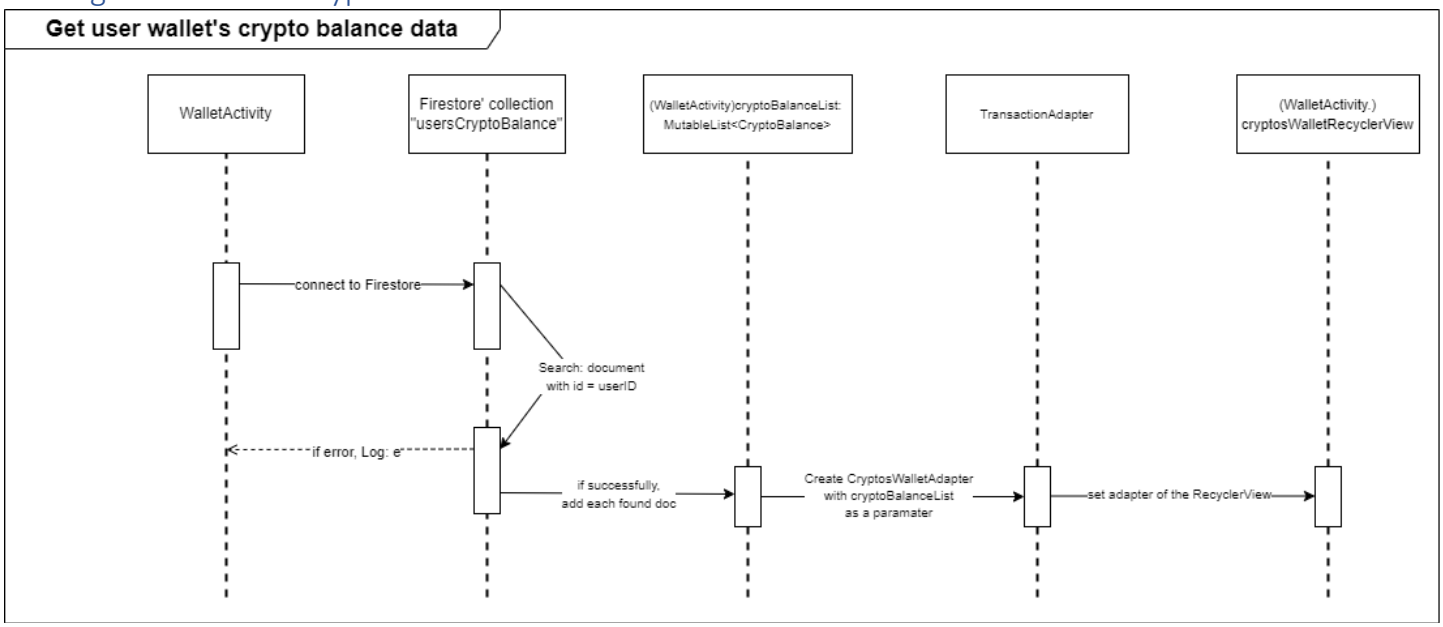


Fig 6. Diagram of getting user wallet 's crypto balance data

Buying cryptocurrency

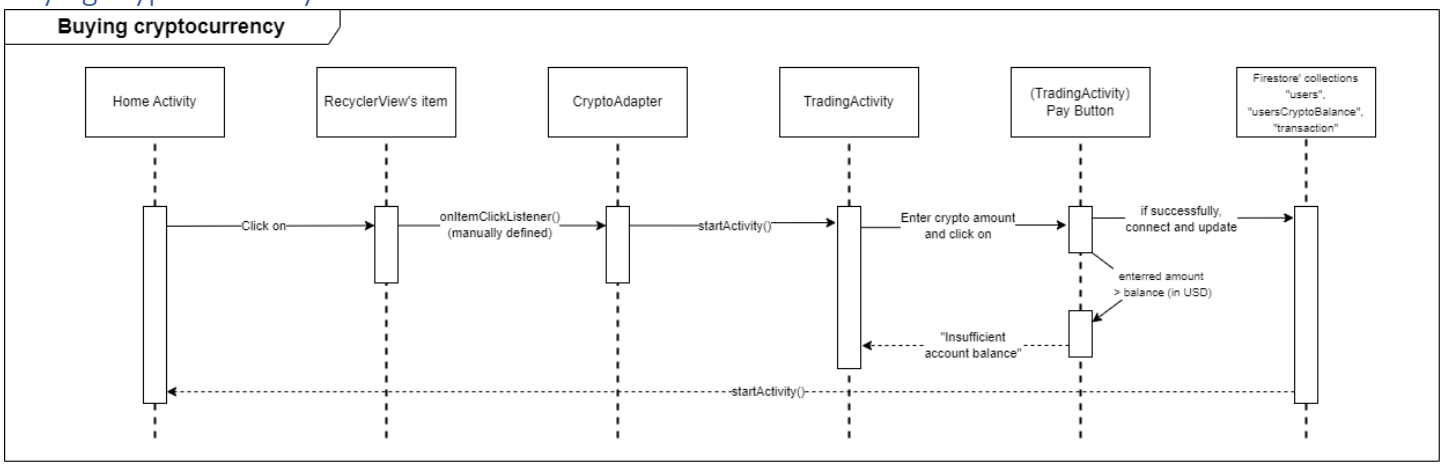


Fig 7. Diagram of buying cryptocurrency

Selling cryptocurrency

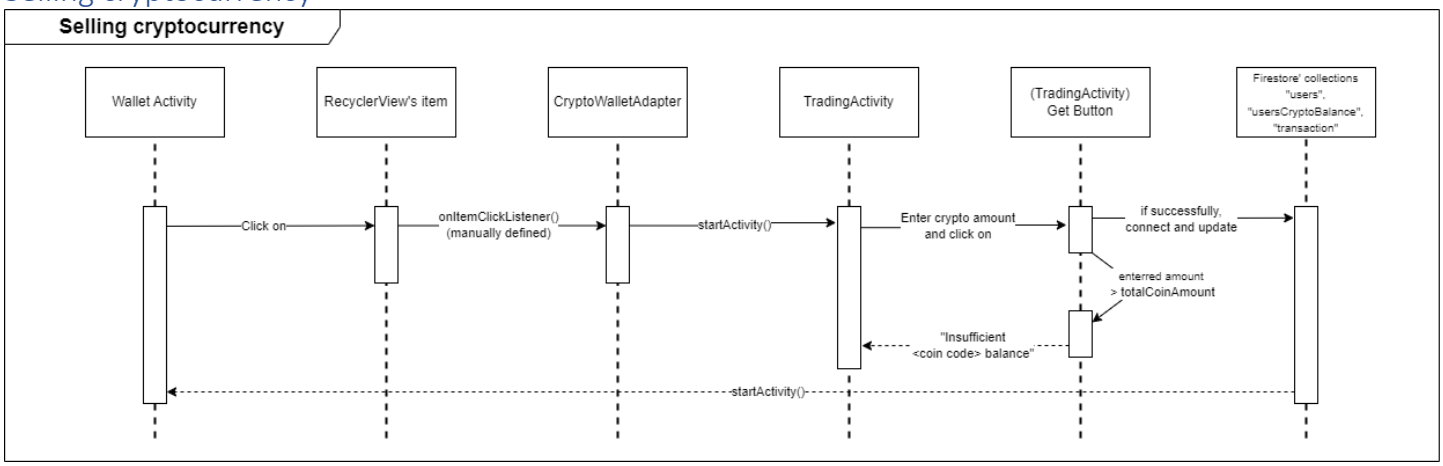


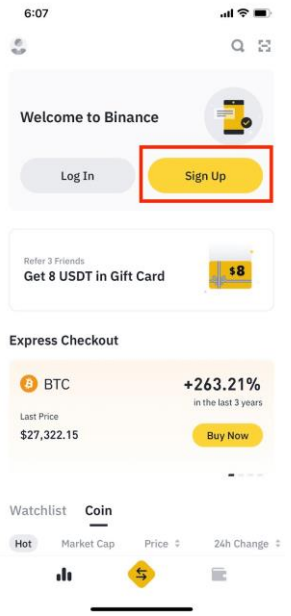
Fig 8. Diagram of selling cryptocurrency

Appendix 2- Weekly reports

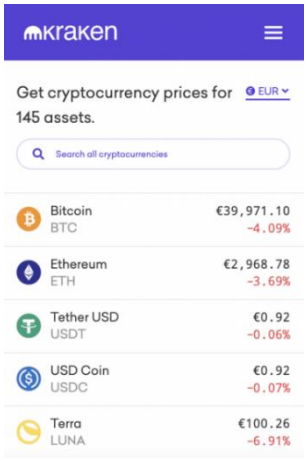
Week 7

Note: I have completed Core 1, submitted Core 2, but not completed the discussion tasks. However, I decided to continue with the custom project as the resubmission for those discussions would not be available until 6 Oct 2023 (end of week 9)
In this step, I focused mainly in searching and referring to the cryptocurrency applications on the market, especially those available on the Android platform:

- Coinbase: In my opinion, this app has an user-friendly interface, ideal for beginners. It offers a secure and regulated platform for buying, selling, and managing cryptocurrencies.



- Binance: This offers not only a wide range of cryptocurrencies for trading, but also additional features like advanced charting tools. It's one of the most popular app among experienced traders.



- Kraken: I firmly believe that this app is well-known for its strong security features, providing access to a variety of cryptocurrencies and trading pairs.

In the following week (Week 8), I think I will try to complete the Authentication feature of the app (Login/Signup). I prefer to use the Google Firebase's authentication functionality.

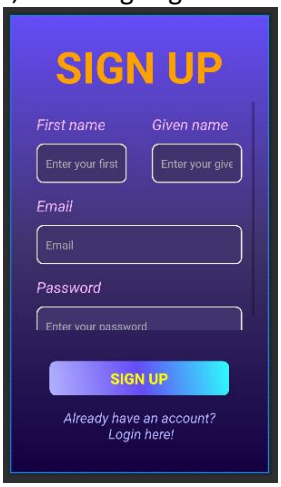
Week 8

Note: I have completed Core 1, submitted Core 2, but not completed the discussion tasks. However, I decided to continue with the custom project as the resubmission for those discussions would not be available until 6 Oct 2023 (end of week 9)

To begin with, I have set up the repository for the Custom Project via the Github Classroom link on Canvas: https://github.com/SoftDevMobDev-2023-Classrooms/customproject1-trungkiennguyen22082004

For the Authentication task, I have implemented two activities, including LoginActivity and SignupActivity.

LoginActivity's layout



SignupActivity's layout

I have use the Google Firebase's functionality of Authentication using Email/Password:

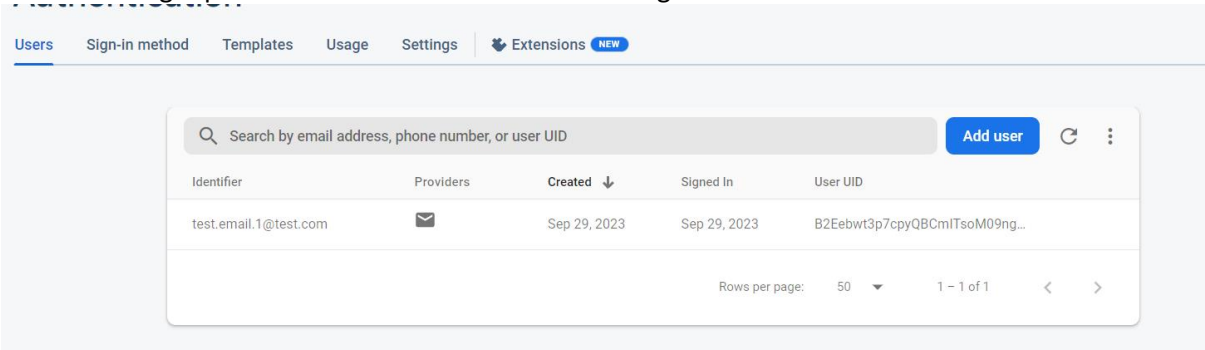
- Login: Using the method "signInWithEmailAndPassword()" of FirebaseAuth

```
_auth.signInWithEmailAndPassword(email, password).addOnCompleteListener(this)
{task ->
    prgBar.visibility = View.GONE

    // If login successfully
    if (task.isSuccessful)
    {
        Log.d(TAG, msg: "signInWithEmail:success")
        Toast.makeText( context: this, text: "Login successfully.", Toast.LENGTH_SHORT).show()

        startActivity(Intent(applicationContext, MainActivity::class.java))
        finish()
    }
    // If login failed
    else
    {
        Log.w(TAG, msg: "signInWithEmail:failure", task.exception)
        Toast.makeText( context: this, task.exception?.localizedMessage, Toast.LENGTH_SHORT).show()
    }
}
```

- Signup: Using the method “createUserWithEmailAndPassword()” of FirebaseAuth

```
_auth.createUserWithEmailAndPassword(email, pwd).addOnCompleteListener(this)
{ task ->
    prgBar.visibility = View.GONE

    if (task.isSuccessful)
    {
        // If sign up successfully
        Log.d(TAG,  msg: "createUserWithEmail:success")
        Toast.makeText( context: this,  text: "Register successful.", Toast.LENGTH_SHORT).show()

        startActivity(Intent(applicationContext, LoginActivity::class.java))
        finish()
    }
    else
    {
        // If sign up failed
        Log.w(TAG,  msg: "createUserWithEmail:failure", task.exception)
        Toast.makeText( context: this, task.exception?.localizedMessage, Toast.LENGTH_SHORT).show()
    }
}
```

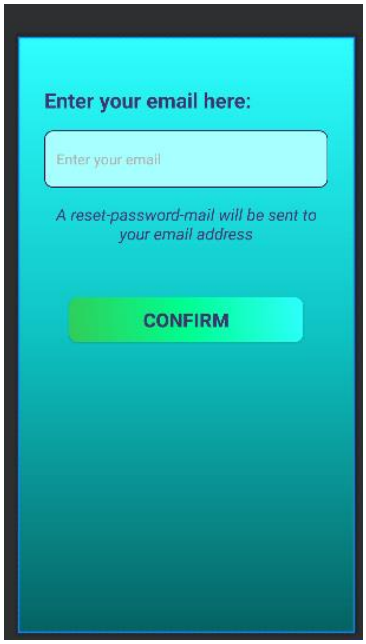
I have use “Signup” to create some new users for testing:



Week 9

Note: I have completed Core 1, completed Core 2, I have just submitted the Redo for discussion tasks since they had been available in the Friday morning.

Firstly, I continued my work on the Authentication functionality of my custom app, adding the forgot-password and reset-password feature:



ForgotPasswordActivity’s layout

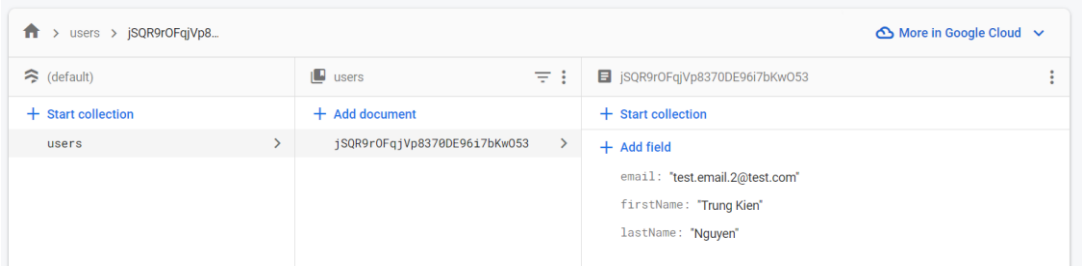
I have used the “sendPasswordResetEmal()” method to sent a resetting-password email to the entered email address:

```
chooseOptionConfirmBtn.setOnClickListener()
{ it: View!
    _auth.sendPasswordResetEmail(enterEdt.text.toString()).addOnCompleteListener()
    {task ->
        if (task.isSuccessful)
        {
            Log.d(TAG,  msg: "Email sent.")
            Toast.makeText( context: this,  text: "Email sent.", Toast.LENGTH_SHORT).show()

            startActivity(Intent(applicationContext, LoginActivity::class.java))
            finish()
        }
        else
        {
            Log.d(TAG,  msg: "Sending email failed.")
            Toast.makeText( context: this,  text: "Sending email failed.", Toast.LENGTH_SHORT).show()
        }
    }
}
```

Next, I stored the User data (First Name, Last Name, …) in Firebase’s feature of Cloud Firestore
After create a Cloud Firestore Database, I have added the “Firestore.DocumentReference.set(user)” method to store the First name, last name and email of the user to the Firebase Cloud Firestore’s database during the signing up process:

```
// Store the user data
val userID: String = _auth.currentUser?.uid ?: ""
val documentReference: DocumentReference = _firestore.collection( collectionPath: "users").document(userID)
val user = HashMap<String, Any>()
user["firstName"] = fName
user["lastName"] = lName
user["email"] = email
documentReference.set(user).addOnSuccessListener()
{ it: Void!
    Log.d(TAG,  msg: "onSuccess:user profile is created for $userID")
    Toast.makeText( context: this,  text: "User Profile is created for $userID", Toast.LENGTH_SHORT).show()
}
```

I have created a new account for testing this function, and this is the output:



In the following weeks, I will continue to make the Market activity (for displaying digital assets of cryptocurrency), Transaction activity (for displaying history transaction of the user), and probably the User Activity (for viewing and modifying the user’s details)

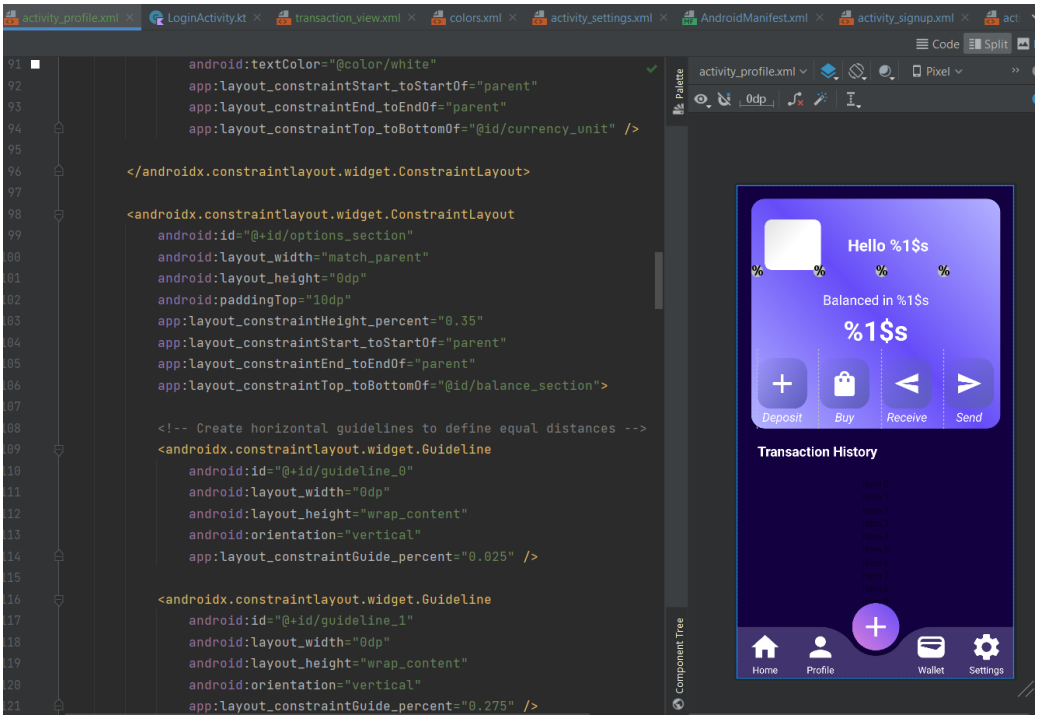
Week 10

[NOTE: as of week 10, if you have not completed Core 1, not completed Core, not submitted Core 3 nor an extension task, you will also need to justify why you should be encouraged to continue with a custom project. There is no point focusing on this task when the basics are not complete and your progress report will be marked as incomplete.]

Note: I have added the design for the main activities for my app in the “Level 1: Design evidence” part

- Profile Activity:

The layout for this activity was quite easily to created for me, using a number of ConstraintLayouts and ConstraintLayout.widget.GuideLines:



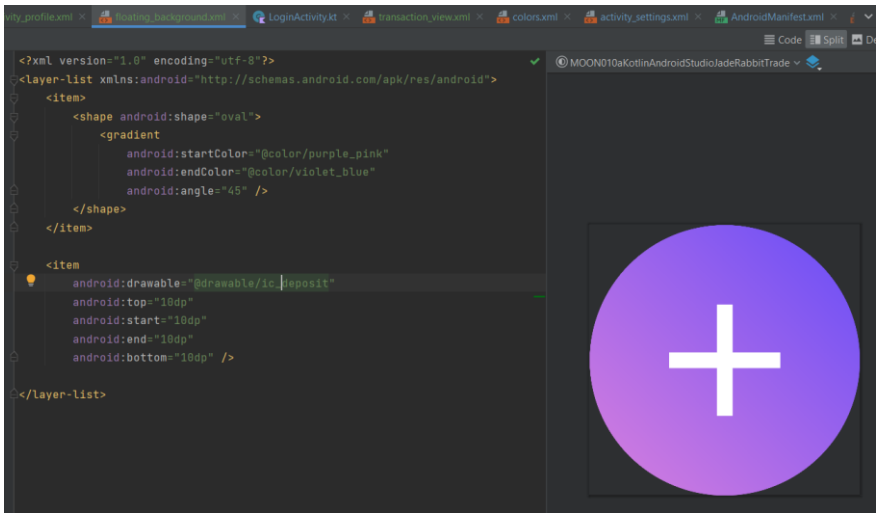
For the Navigation tab, currently I left it inside the Profile Activity instead of creating it in a separate layout file. I have used the BottomAppBar inside a CoordinatorLayout for the base of the navbar:

```
<androidx.coordinatorlayout.widget.CoordinatorLayout
    android:layout_width="match_parent"
    android:layout_height="115dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent">

    <com.google.android.material.bottomappbar.BottomAppBar
        android:id="@+id/bottom_app_bar"
        android:layout_width="match_parent"
        android:layout_marginTop="40dp"
        android:layout_height="75dp"
        android:gravity="bottom"
        android:backgroundTint="@color/violet"
        app:fabAlignmentMode="center"
        app:fabCradleMargin="8dp"
        app:fabCradleRoundedCornerRadius="60dp"
        app:fabCradleVerticalOffset="0dp"
        tools:ignore="VisualLintBottomAppBar">
```

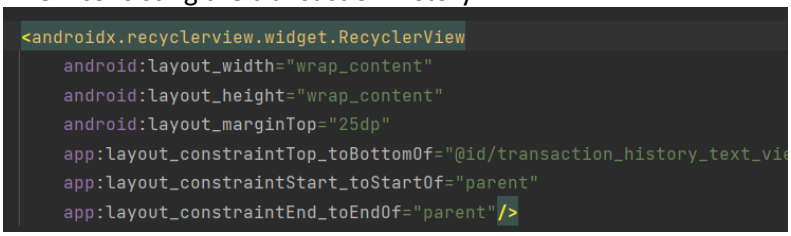
And the ConstraintLayouts, each contains a ImageButton and a TextView, for the functionality buttons:

```
<androidx.constraintlayout.widget.ConstraintLayout
    android:id="@+id/home_button_section"
    android:layout_width="0dp"
    android:layout_height="70dp"
    app:layout_constraintWidth_percent="0.2"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintStart_toStartOf="parent">

    <ImageButton
        android:id="@+id/home_button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="1dp"
        android:contentDescription="Home Btn"
        android:background="@color/transparent"
        android:src="@drawable/ic_home"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Home"
        android:textColor="@color/white"
        app:layout_constraintTop_toBottomOf="@+id/home_button"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

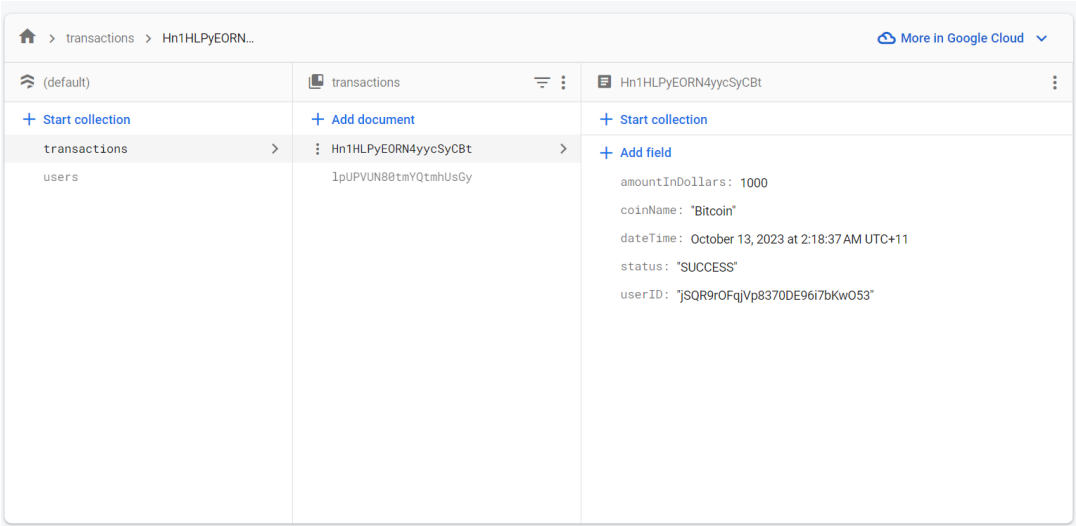
Following that, a float button of the navbar was created using FloatingActionButton, with the drawable of “floating_background” as the foreground, which are shown as below:



The next part is the RecyclerView consisting the transaction history:

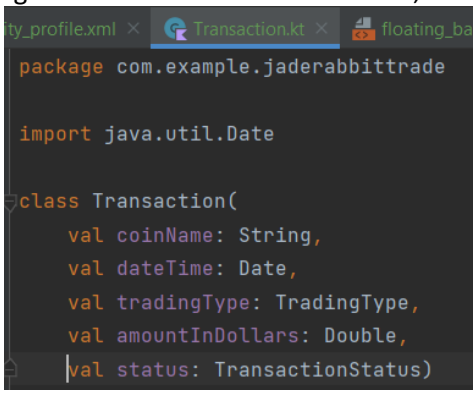


For the database, I have again used the Firestore functionality, creating a database named “transaction”:

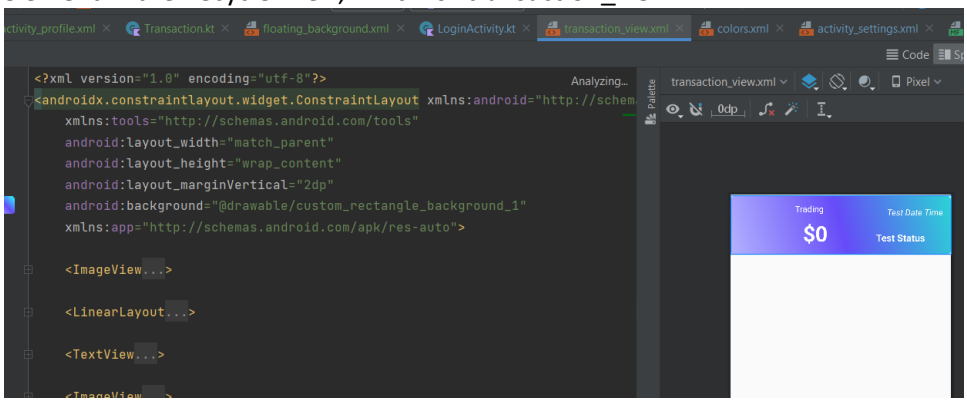


The records were entered manually by me instead of implementing in the code, as the Buy/sell functionality of the HomeActivity had not been done yet

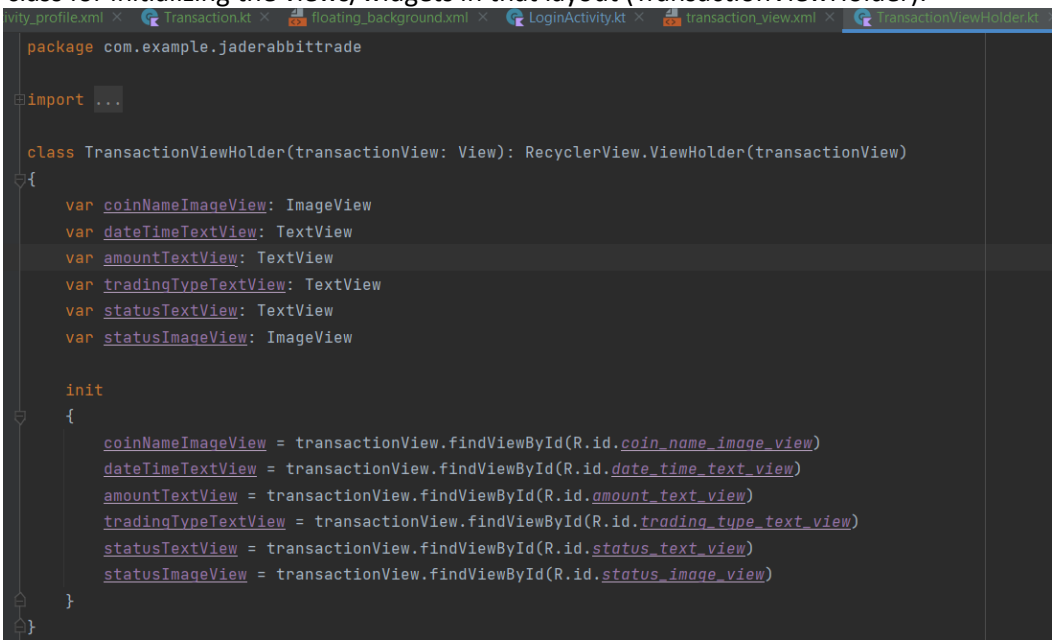
And the corresponding Kotlin class, storing the information of a transaction, named Transaction:



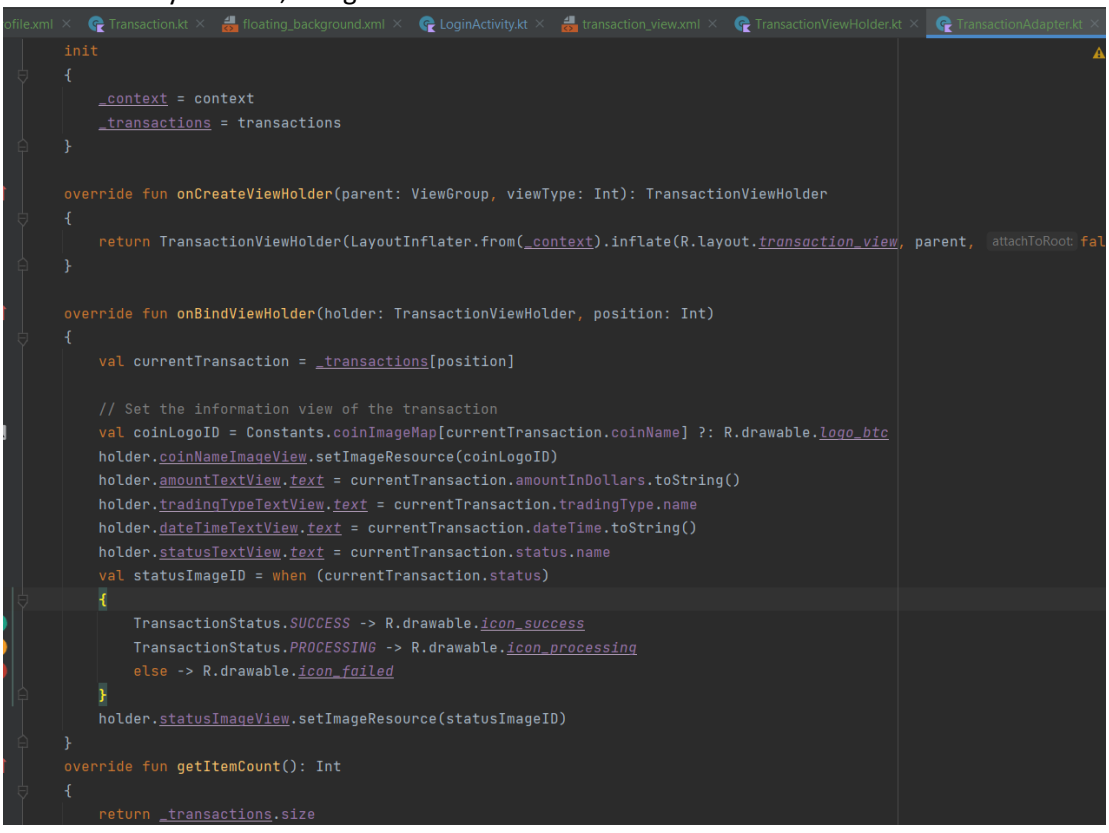
The layout for each element in the RecyclerView, which is “transaction_view.xml”:



The ViewHolder class for initializing the views/widgets in that layout (TransactionViewHolder):



And the adapter for the RecyclerView, using those TransactionViewHolders:



I'm in progress of creating Home and Settings activities with the floating navbar, which is expected to be done by this Sunday (10/15/2023)

My questions for this week:

- For the task "Architectural design, with the diagrams as an appendix", do I have to include them here on the weekly reports, or just add in the final one? If they need to be added here, can I add them in the next (final) weekly report?
- Can you please give some further instructions for this requirement, or may be an example? I have read that "UML might not be suitable for your app", but is it satisfactory to use them?

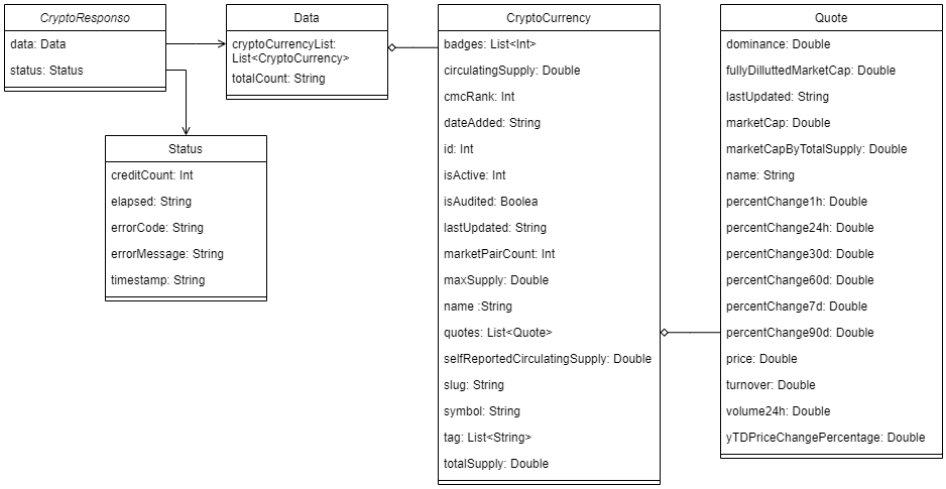
Week 11

[NOTE: as of week 11, if you have not submitted/completed all other Core/Extension tasks, you will also need to justify why you should be encouraged to continue with a custom project. There is no point focusing on this task when the basics are not complete and your progress report will be marked as incomplete.]

My work in this week is mainly focus on the Home Activity, including retrieving the crypto assets from the free API of Coin Market Cap using Retrofit.
The link of the API:
api.coinmarketcap.com/data-api/v3/cryptocurrency/listing?start=1&limit=500


```
{
    "data": {
        "cryptoCurrencyList": [
            {
                "id": 1,
                "name": "Bitcoin",
                "symbol": "BTC",
                "slug": "bitcoin",
                "tags": [
                    "mineable",
                    "pow",
                    "sha-256",
                    "store-of-value",
                    "state-channel",
                    "coinbase-ventures-portfolio",
                    "three-arrows-capital-portfolio",
                    "polychain-capital-portfolio",
                    "binance-labs-portfolio",
                    "blockchain-capital-portfolio",
                    "boostvc-portfolio",
                    "cms-holdings-portfolio",
                    "dcg-portfolio",
                    "dragonfly-capital-portfolio",
                    "electric-capital-portfolio",
                    "fabric-ventures-portfolio",
                    "framework-ventures-portfolio",
                    "galaxy-digital-portfolio",
                    "huobi-capital-portfolio",
                    "alameda-research-portfolio",
                    "a16z-portfolio",
                    "1confirmation-portfolio",
                    "winklevoss-capital-portfolio",
                    "usv-portfolio",
                    "placeholder-ventures-portfolio",
                    "pantera-capital-portfolio",
                    "multicoin-capital-portfolio",
                    "paradigm-portfolio",
                    "bitcoin-ecosystem",
                    "ftx-bankruptcy-estate"
                ],
                "cmcRank": 1,
                "marketPairCount": 10496,
                "circulatingSupply": 19518200,
                "selfReportedCirculatingSupply": 0,
                "totalSupply": 19518200,
                "maxSupply": 21000000,
                "isActive": 1,
                "lastUpdated": "2023-10-19T12:53:00.000Z",
                "dateAdded": "2010-07-13T00:00:00.000Z",
                "quotes": [
                    {
                        "name": "USD",
                        "price": 28533.5789736541,
                        "volume24h": 11189835800.832928,
                        "marketCap": 556924101123.5754,
                        "percentChange1h": 0.3026266,
                        "percentChange24h": 0.69103163,
```


I have also create some classes to get the Crypto Assets Response from the API, which are shown in the following UML Diagram:



To retrieve the data from the API, firstly, I created the Kotlin interface of “ICryptoAPI” with the method “getCryptoAssets()”, defining the endpoint of the api (“data-api/v3/cryptocurrency/listing?start=1&limit=500%22”):

```kotlin
package com.example.jaderabbittrade.crypto_api

import retrofit2.http.GET


interface ICryptoAPI
{
    @GET("data-api/v3/cryptocurrency/listing?start=1&limit=500")
    suspend fun getCryptoAssets(): CryptoResponse
}
```

Then, I created a Kotlin object of CryptoAPIUtility. Its instance return a Retrofit object, with the base URL of the API:

```kotlin
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory

object CryptoAPIUtility
{
    fun getInstance(): Retrofit
    {
        return Retrofit.Builder()
            .baseUrl("https://api.coinmarketcap.com/")
            .addConverterFactory(GsonConverterFactory.create())
            .build()
    }
}
```

Finally, in the Home Activity, I created a method “getTopCurrencyList()” to get the top crypto assets in the API:

```kotlin
private fun getTopCurrencyList()
{
    lifecycleScope.launch()
    {
        val response = try
        {
            CryptoAPIUtility.getInstance().create(ICryptoAPI::class.java).getCryptoAssets()
        }
        catch (e: Exception)
        {
            Log.e( tag: "API Response", e.toString())

            null
        }

        response?.let()
        {cryptoResponse ->
            // All Crypto Assets from the API
            val cryptoAssets = cryptoResponse.data.cryptoCurrencyList

            // The 20 Top Crypto Assets from the above list
            val topCryptoAssets = cryptoAssets.take( n: 20)

            for (cryptoAsset in topCryptoAssets)
            {
                Log.d( tag: "Retrieving Crypto Asset", cryptoAsset.toString())
            }

            _cryptoRecyclerView.adapter = CryptoAdapter(applicationContext, topCryptoAssets.toMutableList())
        }
    }
}
```

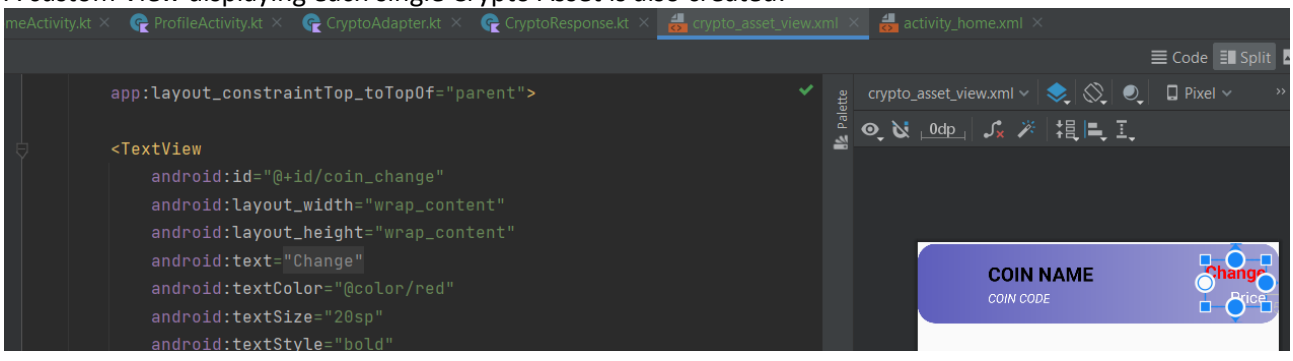
All the code is put in a Coroutine:

- Inside the coroutine, it tries to make a network request to a crypto API using Retrofit
- If response is not null, proceed with processing the data
- Inside the “let” block, it does the following:
    - Retrieve the list of cryptocurrency assets from the API response.
    - Select the top 20 cryptocurrencies from the list (This number can be changed easily to get more or less crypto assets in the API, for this app, I will use 20 top crypto assets from the API for easily visualizaing)
    - Finally, it sets up a RecyclerView to display that top 20 cryptocurrencies using a custom CryptoAdapter (shown as the below illustations). The adapter property of “_cryptoRecyclerView” is set to a new instance of the CryptoAdapter, passing the applicationContext and topCryptoAssets as data to be displayed in the RecyclerView.

```
class CryptoAdapter(context: Context, cryptoAssets: MutableList<CryptoCurrency>) : RecyclerView.Adapter<CryptoAdapter.CryptoViewHo
{
    private var _context: Context
    private var _cryptoAssets: MutableList<CryptoCurrency>

    init
    {...}

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): CryptoAdapter.CryptoViewHolder
    {...}

    override fun onBindViewHolder(holder: CryptoAdapter.CryptoViewHolder, position: Int)
    {
        val currentCryptoAsset = _cryptoAssets[position]

        val coinLogoID = Constants.coinImageMap[currentCryptoAsset.symbol] ?: R.drawable.logo_btc
        holder.coinNameImageView.setImageResource(coinLogoID)

        holder.coinNameTextView.text = currentCryptoAsset.name
        holder.coinCodeTextView.text = currentCryptoAsset.symbol

        val percentChange24h = currentCryptoAsset.quotes[0].percentChange24h
        holder.coinChangeTextView.text = String.format("%.2f", percentChange24h)
        if (percentChange24h >= 0)
        {
            holder.coinChangeTextView.setTextColor(ContextCompat.getColor(holder.itemView.context, R.color.green))
        }
        else
        {
            holder.coinChangeTextView.setTextColor(ContextCompat.getColor(holder.itemView.context, R.color.red))
        }

        holder.coinPriceTextView.text = String.format("%.2f", currentCryptoAsset.quotes[0].price)
    }

    override fun getItemCount(): Int
    {...}


    inner class CryptoViewHolder(cryptoAssetView: View): RecyclerView.ViewHolder(cryptoAssetView)
    {
        var coinNameImageView: ImageView
        val coinNameTextView: TextView
        val coinCodeTextView: TextView
        val coinChangeTextView: TextView
        val coinPriceTextView: TextView

        init
        {
            coinNameImageView = cryptoAssetView.findViewById(R.id.coin_name_image_view)
            coinNameTextView = cryptoAssetView.findViewById(R.id.coin_name)
            coinCodeTextView = cryptoAssetView.findViewById(R.id.coin_code)
            coinChangeTextView = cryptoAssetView.findViewById(R.id.coin_change)
            coinPriceTextView = cryptoAssetView.findViewById(R.id.coin_price)
        }
    }
```

CryptoAdapter’s class

A custom View displaying each single Crypto Asset is also created:



And, this is the output:

The Green/Red Text displays the nearest 24 price change in percentage, while the below text displays the price in USD, which in the following week may swap to other currencies such as EUR (Euro) or AUD (Australian Dollars)

The search/filter ffunctionalities for this RecyclerView, as well as the images (icons) of the Crypto Assets have not been implemented yet, due to lack of time. I may finish them before the start of the week 12.