University of Edinburgh

School of Informatics

Informatics Large Practical

INFR09051

Coursework 2

# COINZ REPORT

–

Erik Kvasnicka

S1660343

## 1 Introduction

This report provides detailed information about the “Coinz” application, description of algorithms and data structures used for main functions, design decisions and how they changed from first design to final implementation along with description of additional bonus features. Then the screenshots of the application in use are presented, where each feature of the app is shown and described. Lastly, the parts of code and dependencies used from other sources are acknowledged. Android Emulator was used to test app performance, specifically Nexus 5X, API level 27 with Android 8.1.

## 2 Description of algorithms and data structures used

In my application I have used data structures so that the computations can be computed efficiently and that the whole code would look clean at the same time. I have used many approaches, but the ones described below have shown to be most appropriate.

### 2.1 Downloading of the GeoJson map from the server, reading it and parsing the data to the map.

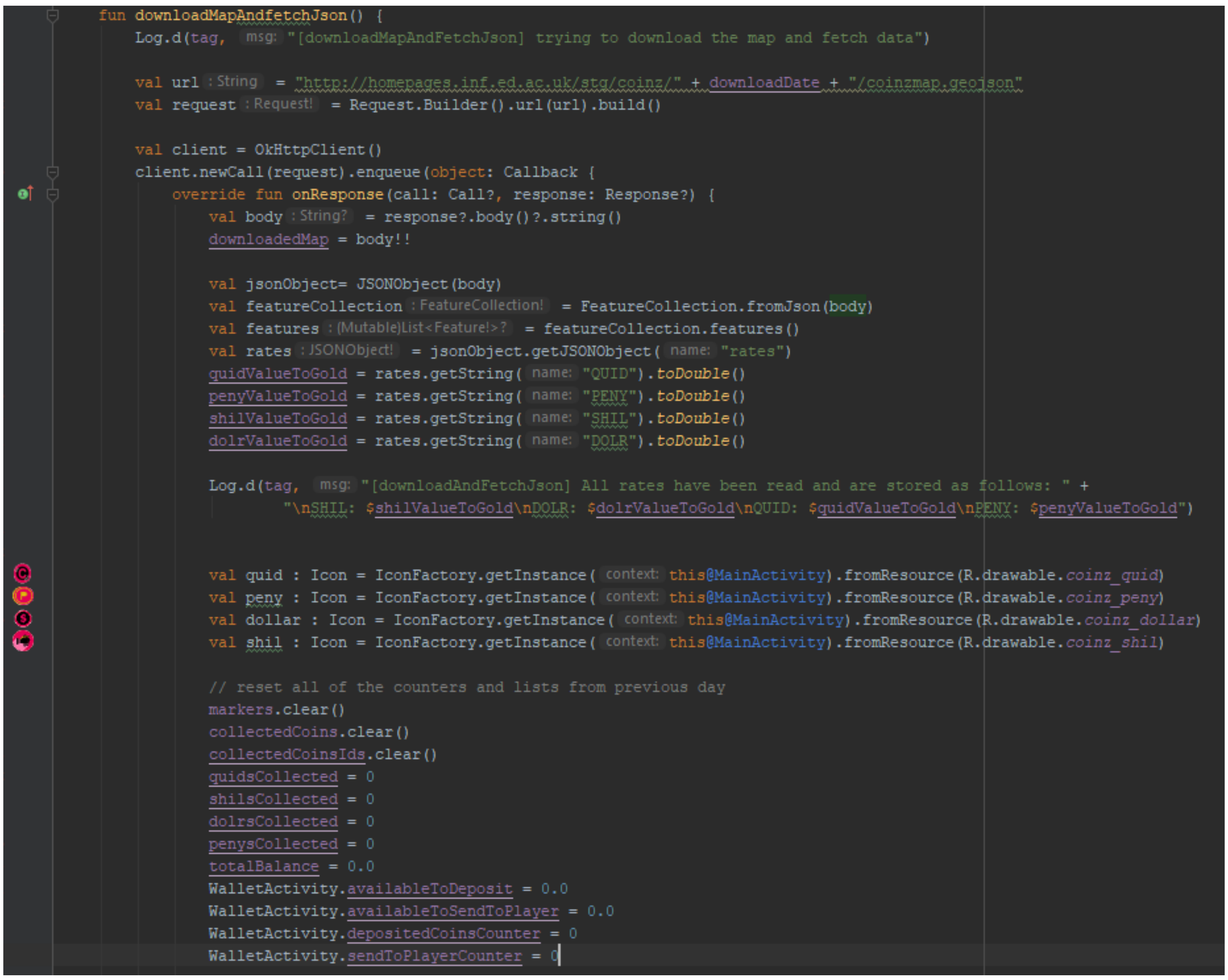
First, I have used the dynamic URL address, where the downloadDate was changing with respect to the current date and last map downloaded date.

The OkHttp3 library was used to access the desired URL. OkHttp provides an implementation of the HttpUrlConnection interface. So as a client, OkHttpClient was used to connect to the server. We built a request object with our URL and opened a connection. We then performed an asynchronous network call so that the process is running on a separate thread. A `Call` object was created, using the enqueue() method, and passing an anonymous Callback object that implements both onFailure() and onResponse() methods. This way we get the map from the server.

In the onResponse() method we then can access that string and using Json library, FeatureCollection library from Json and other we are able to convert the string to JsonObject. So now we have access to the features with the method features() and all other necessary values of the JsonObject.

For creation of the icons, IconFactory library was used and with methods getInstance() and fromResource() we were able to show our designed icons on the map.

Once the map is downloaded, that means that it is a first time per that day the app was used by the player’s device and so coins collected, balances gained from the previous day needs to be set again to 0. At this point, these are cleared, our MutableLists <Marker_defined> that hold our collected and uncollected coins are also cleared.

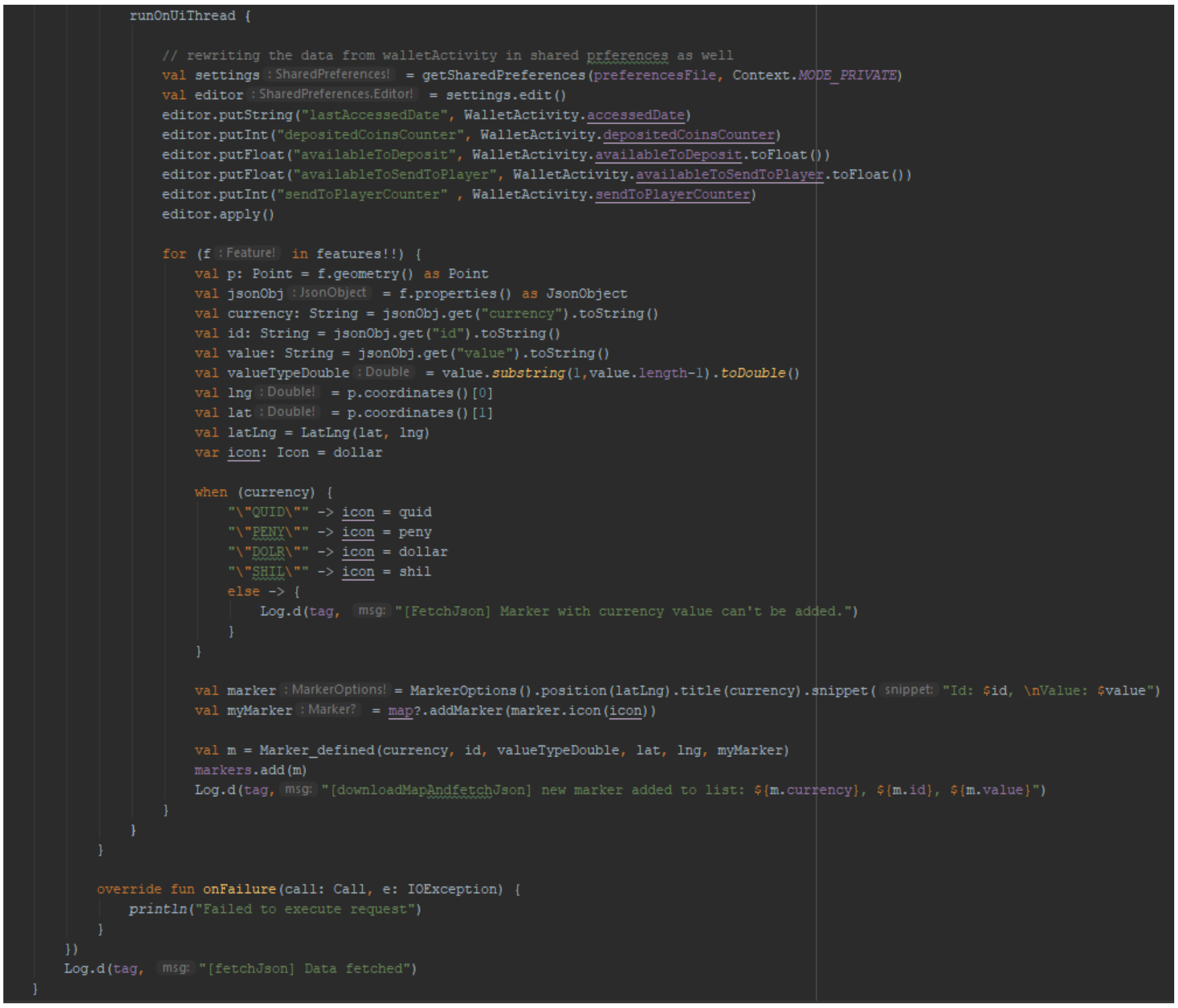


```
fun downloadMapAndfetchJson() {
    Log.d(tag, msg: "[downloadMapAndFetchJson] trying to download the map and fetch data")

    val url : String = "http://homepages.inf.ed.ac.uk/stg/coinz/" + downloadDate + "/coinzmap.geojson"
    val request : Request! = Request.Builder().url(url).build()

    val client = OkHttpClient()
    client.newCall(request).enqueue(object: Callback {
        override fun onResponse(call: Call?, response: Response?) {
            val body : String? = response?.body()?.string()
            downloadedMap = body!!

            val jsonObject= JSONObject(body)
            val featureCollection : FeatureCollection! = FeatureCollection.fromJson(body)
            val features : (Mutable)List<Feature!>? = featureCollection.features()
            val rates : JSONObject! = jsonObject.getJSONObject( name: "rates")
            quidValueToGold = rates.getString( name: "QUID").toDouble()
            penyValueToGold = rates.getString( name: "PENY").toDouble()
            shilValueToGold = rates.getString( name: "SHIL").toDouble()
            dolrValueToGold = rates.getString( name: "DOLR").toDouble()

            Log.d(tag, msg: "[downloadAndFetchJson] All rates have been read and are stored as follows: " +
                    "\nSHIL: $shilValueToGold\nDOLR: $dolrValueToGold\nQUID: $quidValueToGold\nPENY: $penyValueToGold")


            val quid : Icon = IconFactory.getInstance( context: this@MainActivity).fromResource(R.drawable.coinz_quid)
            val peny : Icon = IconFactory.getInstance( context: this@MainActivity).fromResource(R.drawable.coinz_peny)
            val dollar : Icon = IconFactory.getInstance( context: this@MainActivity).fromResource(R.drawable.coinz_dollar)
            val shil : Icon = IconFactory.getInstance( context: this@MainActivity).fromResource(R.drawable.coinz_shil)

            // reset all of the counters and lists from previous day
            markers.clear()
            collectedCoins.clear()
            collectedCoinsIds.clear()
            quidsCollected = 0
            shilsCollected = 0
            dolrsCollected = 0
            penysCollected = 0
            totalBalance = 0.0
            WalletActivity.availableToDeposit = 0.0
            WalletActivity.availableToSendToPlayer = 0.0
            WalletActivity.depositedCoinsCounter = 0
            WalletActivity.sendToPlayerCounter = 0
```

Image 1 – downloadMapAndFetchJson()

Up to this point, all was done on a separate thread so that we don’t slow down the processes on the main thread. Now, we use method runOnUiThread() so that we switch back to main UI thread once the download and everything mentioned earlier is done. The balances stored in our Shared Preferences from previous day are also updated at this point.

The for loop is executed, looping over every feature of our feature collection object. For each feature we get its currency, id, value, latitude, longitude. Once we have these values, we run while loop (when loop in Kotlin), so that based on the currency string value we know what icon to use. Once we also have the right icon chosen, using MarkerOptions library we create a MarkerOptions object, create Marker object and by calling our map we use method addMarker() with our previously specified MarkerOptions object as a instance for the method.  Now, the marker is on the map, all that needs to be done is to keep track of it, so we add it to our list, MutableList<Marker_defined>, so that we could access it in the future.

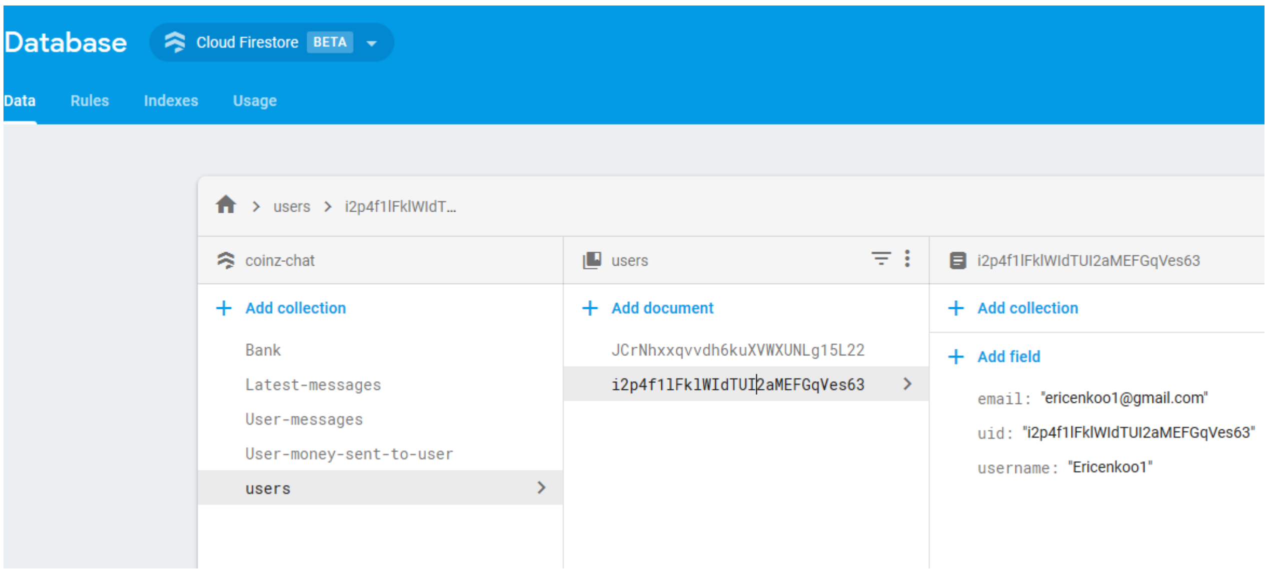


```
            runOnUiThread {

                // rewriting the data from walletActivity in shared prferences as well
                val settings : SharedPreferences! = getSharedPreferences(preferencesFile, Context.MODE_PRIVATE)
                val editor : SharedPreferences.Editor! = settings.edit()
                editor.putString("lastAccessedDate", WalletActivity.accessedDate)
                editor.putInt("depositedCoinsCounter", WalletActivity.depositedCoinsCounter)
                editor.putFloat("availableToDeposit", WalletActivity.availableToDeposit.toFloat())
                editor.putFloat("availableToSendToPlayer", WalletActivity.availableToSendToPlayer.toFloat())
                editor.putInt("sendToPlayerCounter" , WalletActivity.sendToPlayerCounter)
                editor.apply()

                for (f : Feature! in features!!) {
                    val p: Point = f.geometry() as Point
                    val jsonObj : JsonObject = f.properties() as JsonObject
                    val currency: String = jsonObj.get("currency").toString()
                    val id: String = jsonObj.get("id").toString()
                    val value: String = jsonObj.get("value").toString()
                    val valueTypeDouble : Double = value.substring(1,value.length-1).toDouble()
                    val lng : Double! = p.coordinates()[0]
                    val lat : Double! = p.coordinates()[1]
                    val latLng = LatLng(lat, lng)
                    var icon: Icon = dollar

                    when (currency) {
                        "\"QUID\"" -> icon = quid
                        "\"PENY\"" -> icon = peny
                        "\"DOLR\"" -> icon = dollar
                        "\"SHIL\"" -> icon = shil
                        else -> {
                            Log.d(tag, msg: "[FetchJson] Marker with currency value can't be added.")
                        }
                    }

                    val marker : MarkerOptions! = MarkerOptions().position(latLng).title(currency).snippet( snippet: "Id: $id, \nValue: $value")
                    val myMarker : Marker? = map?.addMarker(marker.icon(icon))

                    val m = Marker_defined(currency, id, valueTypeDouble, lat, lng, myMarker)
                    markers.add(m)
                    Log.d(tag, msg: "[downloadMapAndfetchJson] new marker added to list: ${m.currency}, ${m.id}, ${m.value}")
                }
            }
        }

        override fun onFailure(call: Call, e: IOException) {
            println("Failed to execute request")
        }
    })
    Log.d(tag, msg: "[fetchJson] Data fetched")
}
```

Image 2 – downloadMapAndFetchJson() – runOnUiThread()

## 2.2 Persistent storage of the user's progress in playing the game

Two options were used for storing user's progress during the gameplay.
First, Firebase Firestore was used to store user's information such as register information, bank balance, money sent to players, messages sent between players, as Image 3 shows.
Second, Shared Preferences were used to store variables such as lastDownloadDate, downloadedMap string, but others such as total daily balance of the wallet, number of coins collected, quids collected, dollars collected, ids of the collected coins and more. This way the app was able to track user's progress per each day and prevent swindling at the same time. (Explained more in section 3.)



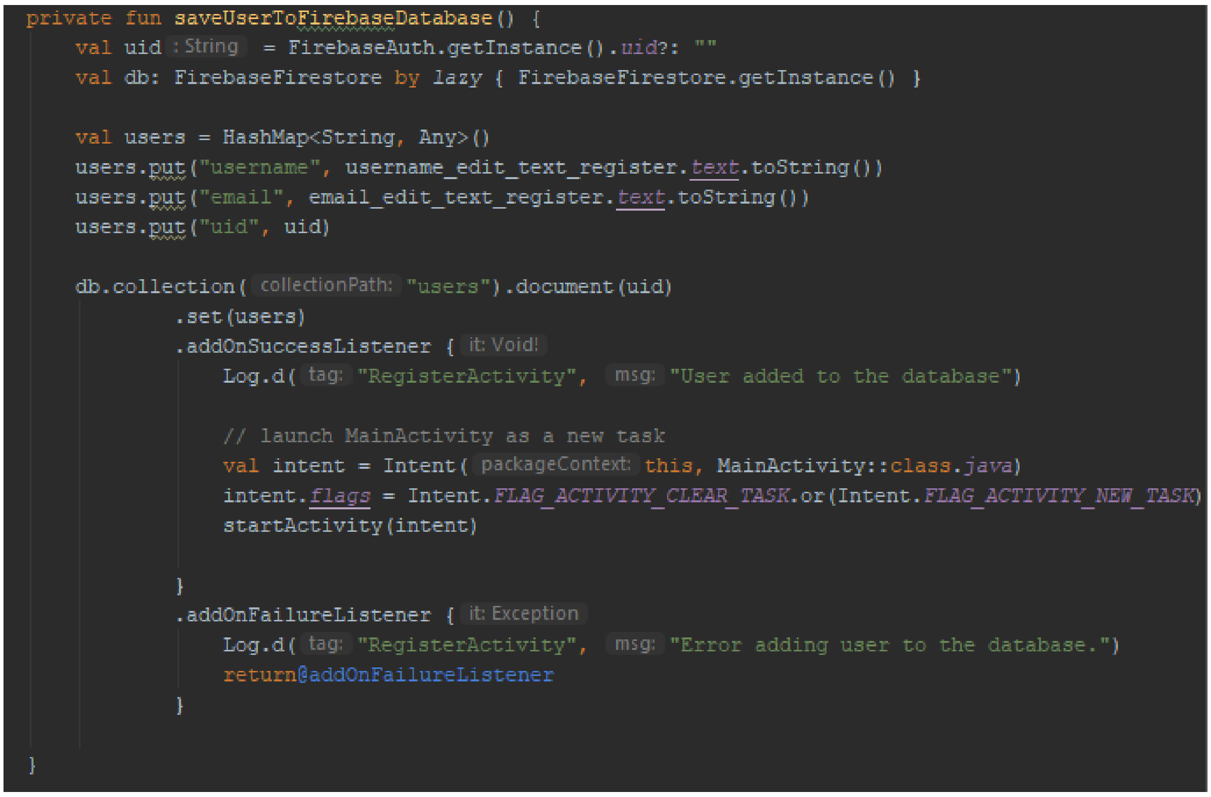
Bank
Latest-messages
User-messages
User-money-sent-to-user
users

JCrNhxxqvvdh6kuXVWXUNLg15L22
i2p4f1lFklWIdTUI2aMEFGqVes63

email: "ericenkoo1@gmail.com"
uid: "i2p4f1lFklWIdTUI2aMEFGqVes63"
username: "Ericenkoo1"

Image 3 – Firebase Cloud Firestore Database

Saving data to Firestore was done by Firebase Firestore libraries, such as FirebaseAuth. Once we initiated an instance for the database, we used HashMap<String, Any>() and put our data we wanted to store to that list. Calling method set(HashMap<>()) on our specified database instance with specified collection and document path we were able to add the data to the database. This process was followed in the cases we were trying to store data to the database. See Image 4 for reference.

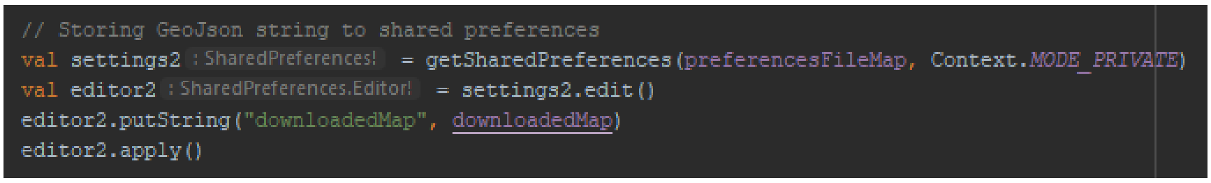


```
private fun saveUserToFirebaseDatabase() {
    val uid :String  = FirebaseAuth.getInstance().uid?: ""
    val db: FirebaseFirestore by lazy { FirebaseFirestore.getInstance() }

    val users = HashMap<String, Any>()
    users.put("username", username_edit_text_register.text.toString())
    users.put("email", email_edit_text_register.text.toString())
    users.put("uid", uid)

    db.collection( collectionPath: "users").document(uid)
            .set(users)
            .addOnSuccessListener { it: Void!
                Log.d( tag: "RegisterActivity",  msg: "User added to the database")

                // launch MainActivity as a new task
                val intent = Intent( packageContext: this, MainActivity::class.java)
                intent.flags = Intent.FLAG_ACTIVITY_CLEAR_TASK.or(Intent.FLAG_ACTIVITY_NEW_TASK)
                startActivity(intent)

            }
            .addOnFailureListener { it: Exception
                Log.d( tag: "RegisterActivity",  msg: "Error adding user to the database.")
                return@addOnFailureListener
            }

}
```

Image 4 – Adding data to Cloud Firestore

Listening for database changes was done via SnapshopListener(EventListener <DocumentSnapshot>) called on database instance.

For storing data in Shared preferences, we created our SharedPreferences object and using edit() method on that object we are able to putString or other value to Shared preferences file. Called apply() on our editor object stored the data to the file and we were able to access it later in the onCreate() method. This way we were able to store downloaded map to shared preferences file as a String and access it when the user opened the app again, read that string from shared preferences with Json library, convert it to Json object and fetch the Json to the map.

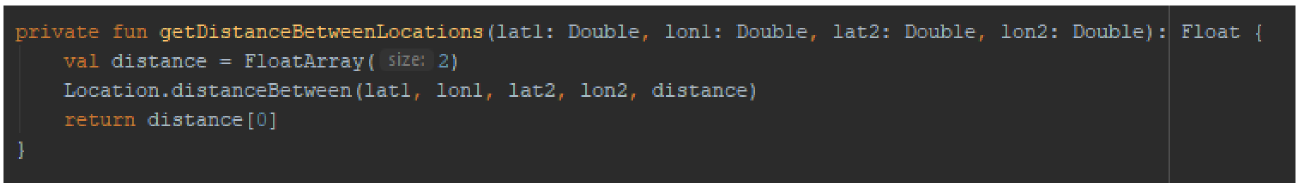


```
// Storing GeoJson string to shared preferences
val settings2 :SharedPreferences!  = getSharedPreferences(preferencesFileMap, Context.MODE_PRIVATE)
val editor2 :SharedPreferences.Editor!  = settings2.edit()
editor2.putString("downloadedMap", downloadedMap)
editor2.apply()
```

Image 5 – Adding data to Shared Preferences

### 2.3 Detection of the distance of the coins from user's location

For detecting the distance between 2 locations, the method distanceBetween() was used. We used latitude and longitude of the respective coin and latitude and longitude of the user's location to calculate this value. Function where it is computed can be seen below at Image 6.



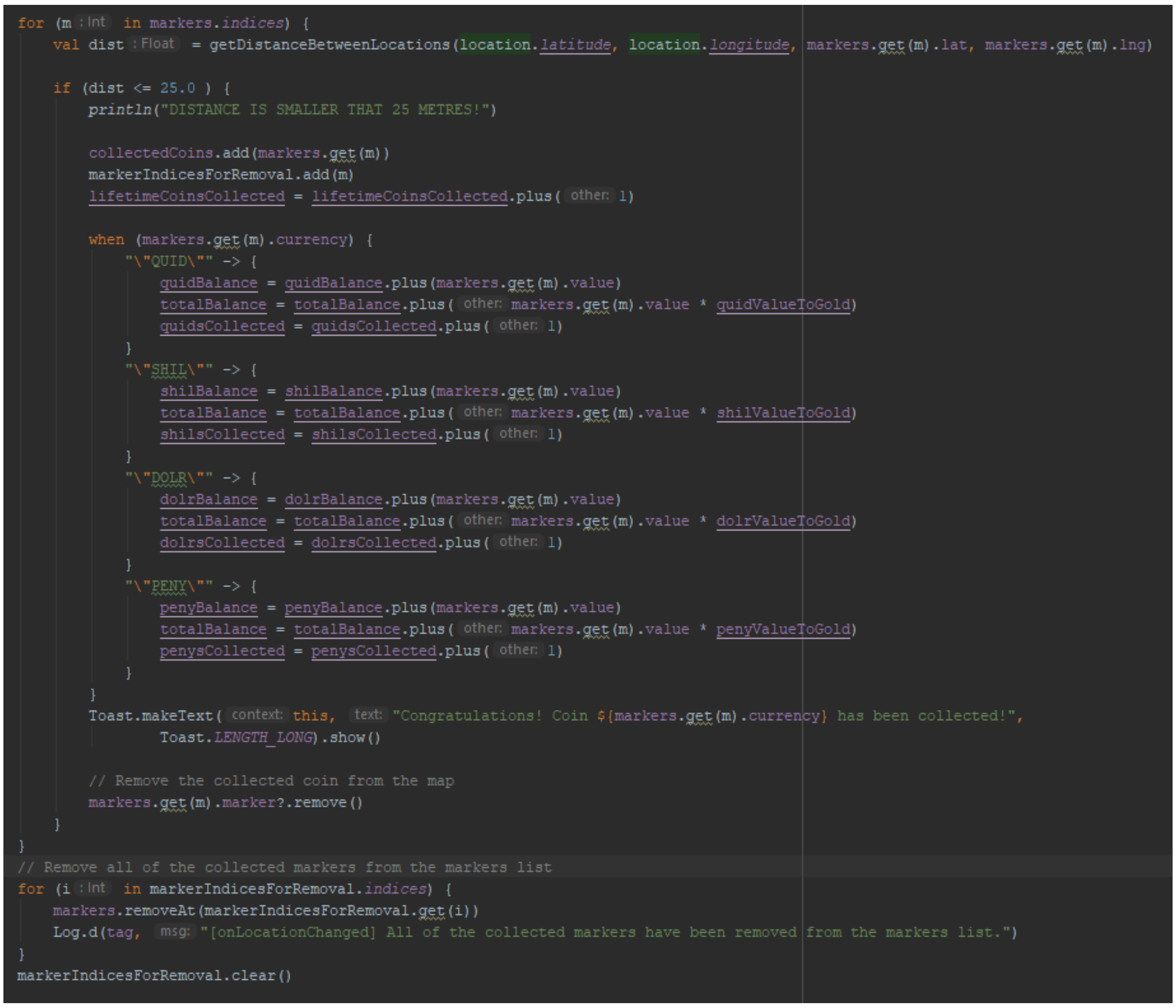
```kotlin
private fun getDistanceBetweenLocations(lat1: Double, lon1: Double, lat2: Double, lon2: Double): Float {
    val distance = FloatArray( size: 2)
    Location.distanceBetween(lat1, lon1, lat2, lon2, distance)
    return distance[0]
}
```

Image 6 – getDistanceBetweenLocations()

At the onLocationChanged() function, we can access the user's location pretty easily. From there, by using for loop we iterate over every instance stored in our markers list (MutableList<Markers_defined>()), where all of the uncollected coins are stored. We compute the distance in metres of each marker to the user's location. If the distance is smaller than 25 metres, we add the object to collectedCoins list, remember the index of the iteration for later removal from the markers list. Using when loop, we get the respective currency and based on the result we know exactly what balances we need to add up, what counters to increment. Once both loops are done, we use for loop to iterate over remembered indices for removal and remove these marker objects from the map, and from the markers list as well, so that they are not added to the map later.  Once the coin is collected, the toast is presented for the user with the information that he collected the coin. The procedure can be seen in Image 7.

## 3 Design and features of the application

Coinz project, that has been designed at first, was very important part of the implementation process and paved the process of decision making. However, there have been things and features, that were not implemented, changed or adjusted. Many of them were improved so that the game can be played easily and more effectively.



```kotlin
for (m: Int in markers.indices) {
    val dist: Float = getDistanceBetweenLocations(location.latitude, location.longitude, markers.get(m).lat, markers.get(m).lng)

    if (dist <= 25.0 ) {
        println("DISTANCE IS SMALLER THAT 25 METRES!")

        collectedCoins.add(markers.get(m))
        markerIndicesForRemoval.add(m)
        lifetimeCoinsCollected = lifetimeCoinsCollected.plus( other: 1)

        when (markers.get(m).currency) {
            "\"QUID\"" -> {
                quidBalance = quidBalance.plus(markers.get(m).value)
                totalBalance = totalBalance.plus( other: markers.get(m).value * quidValueToGold)
                quidsCollected = quidsCollected.plus( other: 1)
            }
            "\"SHIL\"" -> {
                shilBalance = shilBalance.plus(markers.get(m).value)
                totalBalance = totalBalance.plus( other: markers.get(m).value * shilValueToGold)
                shilsCollected = shilsCollected.plus( other: 1)
            }
            "\"DOLR\"" -> {
                dolrBalance = dolrBalance.plus(markers.get(m).value)
                totalBalance = totalBalance.plus( other: markers.get(m).value * dolrValueToGold)
                dolrsCollected = dolrsCollected.plus( other: 1)
            }
            "\"PENY\"" -> {
                penyBalance = penyBalance.plus(markers.get(m).value)
                totalBalance = totalBalance.plus( other: markers.get(m).value * penyValueToGold)
                penysCollected = penysCollected.plus( other: 1)
            }
        }
        Toast.makeText( context: this,  text: "Congratulations! Coin ${markers.get(m).currency} has been collected!",
                Toast.LENGTH_LONG).show()

        // Remove the collected coin from the map
        markers.get(m).marker?.remove()
    }
}
// Remove all of the collected markers from the markers list
for (i: Int in markerIndicesForRemoval.indices) {
    markers.removeAt(markerIndicesForRemoval.get(i))
    Log.d(tag,  msg: "[onLocationChanged] All of the collected markers have been removed from the markers list.")
}
markerIndicesForRemoval.clear()
```

Image 7 – onLocationChanged() – Collecting coins

## 3.1 First design decisions not implemented

The first point mentioned in design of the app was that the user can select the number of cryptocurrency coins he wants to deposit to the bank. The money should have been then transferred to GOLD and deposited. This have changed and the user can’t choose what coins to deposit. It is done rather automatically based on what coins he collects, in order, counting from 1 to 25. The user is depositing the first 25 coins to the bank. Only after the 25 coins have been collected, then he can send the remaining coins to other players, counting from 26 to 50. This was changed as well. The reason for that is straightforward. Make the game simple, easy to use and still very competitive.

Design was also mentioning “Spare change wallet”. This was omitted and now the balances and coins are stored in one wallet. The implementation has shown, that it is more than enough, having 2 wallets would be a waste of space, would make the app slower and this way it improves the user experience a lot.

Levelling up as a bonus feature was implemented and followed the design decisions closely, but the slight variations were made. Only 2 levels have been presented, “runner” and “pro”. Runners are players, who collected less than 1000 coins in a lifetime. These players can send GOLD just to 1 other player per day. Pro players are the ones who collected more than 1000 coins in a lifetime and so as a reward can send money to 2 players per day. This way they can compete even more, make the game engaging and rewarding.

Other bonus features, such as “time king” were not implemented, because of a complexity of a more engaging, useful bonus feature, user to user chat. More about it will be presented in section 3.2. The previously mentioned bonus features and other improvements could be implemented in next app version.

Lastly, expirations of the coins were changed. Now, the collected coins and balances gained throughout the day expires at the end of the calendar day. This way the players are more inclined to return to the game, deposit their money to the bank or play more to send the money to different players.

## 3.2 Additional features not realised before

New bonus feature was implemented in order to gain more user attention by providing easy communication and private conversations for the users. User to user chat. In addition to sending money between users, users can use separate window for personal messages. Each player can message to every other registered player by clicking on his image icon or name in New Message section. Once user chooses the user to send the message to, Chat Log window is opened with their past discussions and he can write new message. This message automatically shows to the other user in his “Messages” section. Sending money between users was changed in a way that quite similar approach was used. Once user clicks send money to another player button, it redirects him to another window where he chooses the player by clicking in the name or icon of the user that he wants to send the money to. Then the money is automatically sent and will show on other player's wallet as Money received from another player.

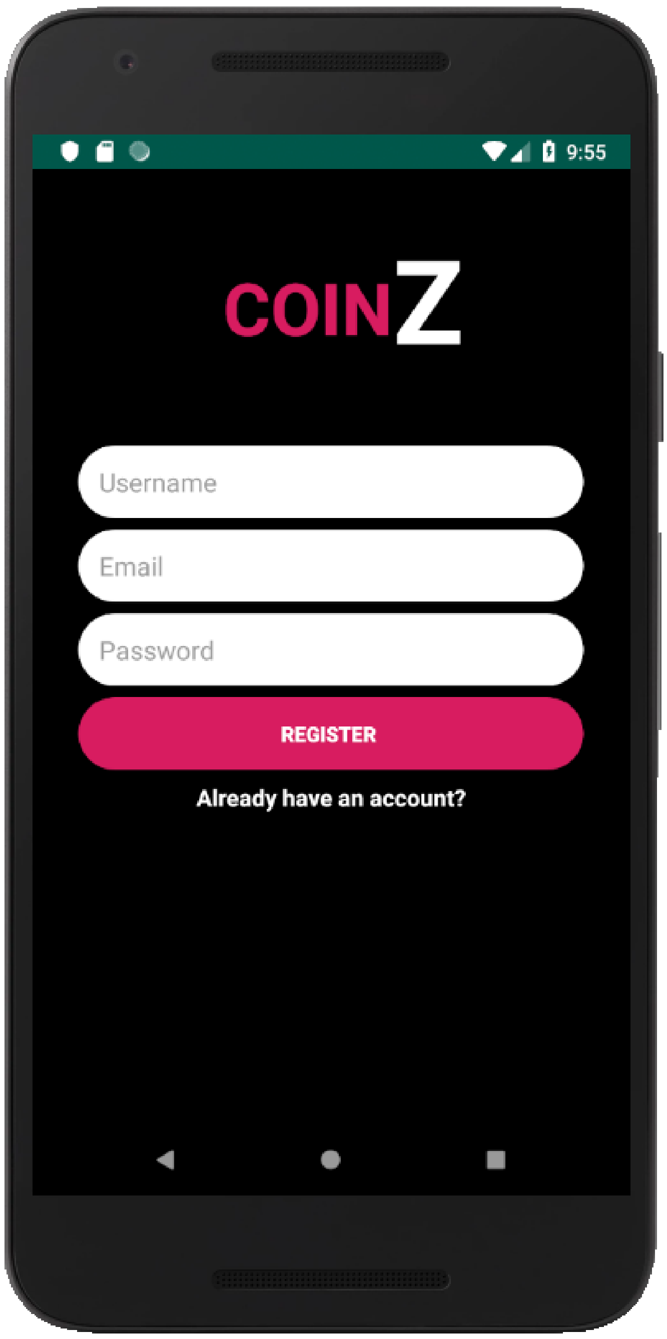
Another feature not presented before, but key one for the whole game is "Help". This feature provides basic rules of the game so that the players know exactly how to play the game and what is permitted. If the player can't execute the action, for example deposit money or send more money, he can look it up in Help and he will know the answer why it is so. Help provides information about levelling, sending money, coins expirations and more.

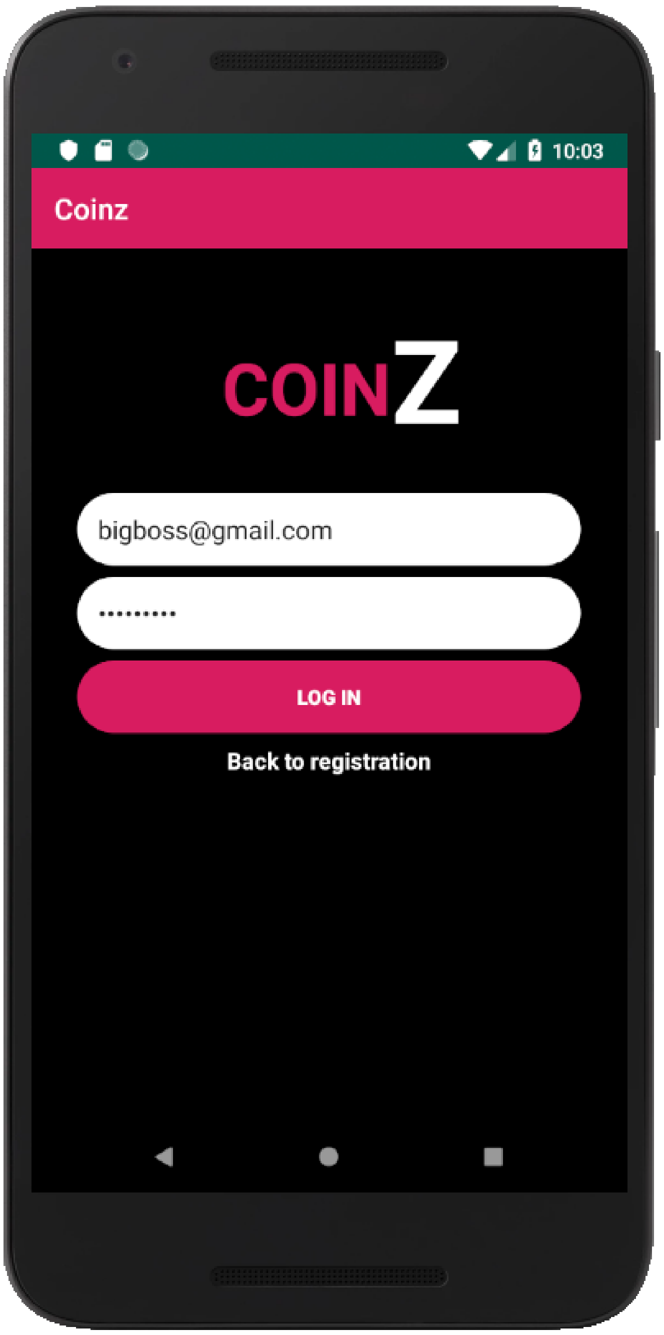
To prevent swindling and creation of many accounts of one user, the appropriate steps were also made. The downloaded map is stored in shared preferences file, along with collected coins ids, balances, etc. That means that all of these are used for one device. Since we assume that each player has his own device and we also present it in the Coinz rules, the player won't be able to collect more than 50 coins per day with multiple accounts per device, nor increase his chances of depositing more to his bank.

Quick access wallet was also added as a new feature. This quick wallet can be accessed from the Main Activity, so while the player is collecting the coins. It provides quick access for the user to see his balance for the day, without going to the main wallet. That way the user can continue collecting the coins, track his progress quicker and make sure he doesn't miss any coin.

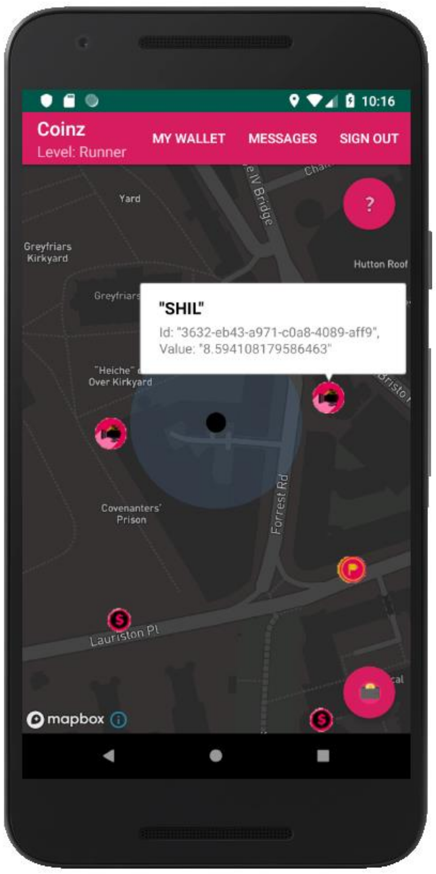
## 4 Application in use

Below, the screenshots of main features from the gameplay are presented.

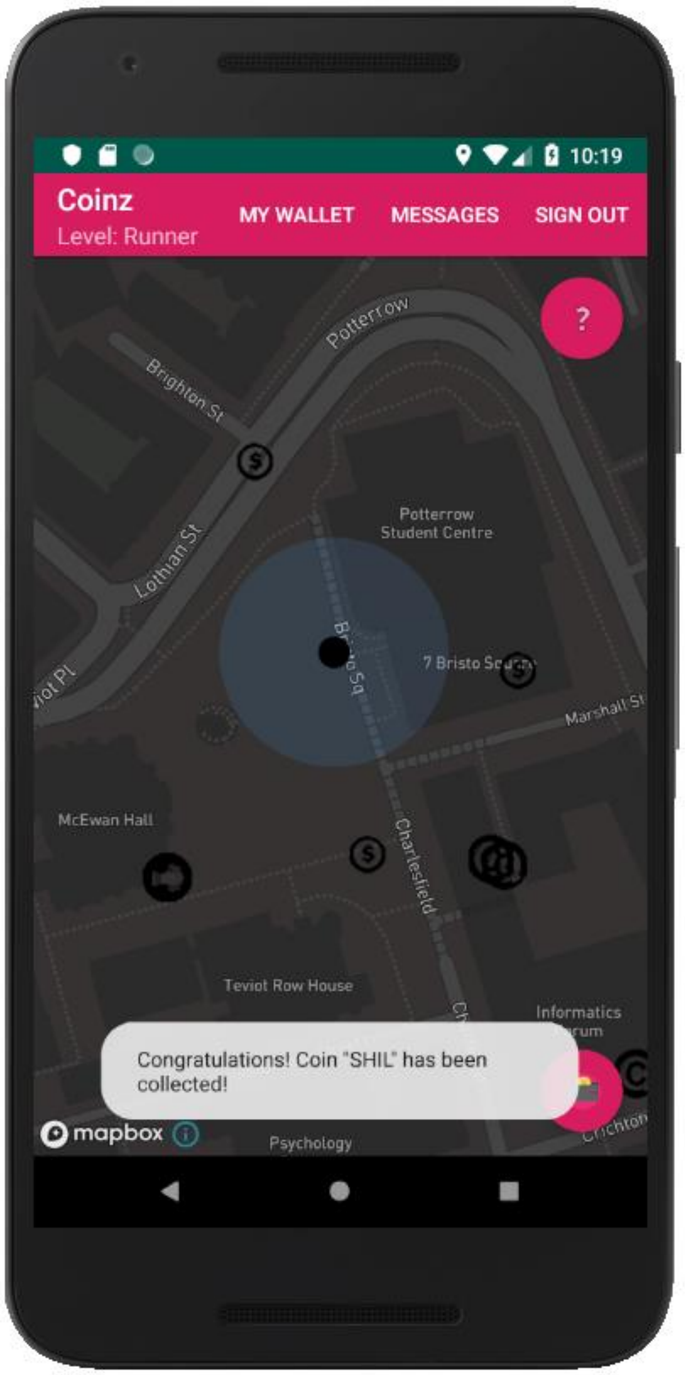




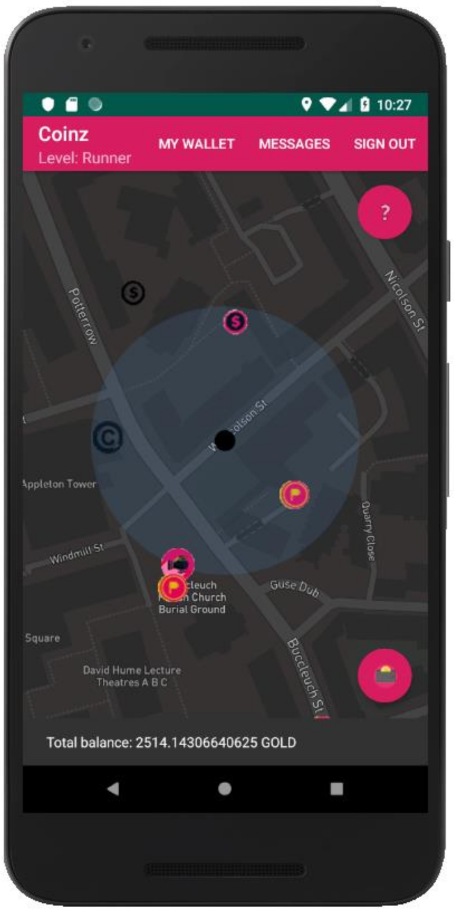
At the top left screenshot, the register screen is presented. User need to register with username, email and password in order to play. If the user has an account, he can press the button "Already have an account?" and be redirected to Login screen, which can be seen on the top right corner. Once the user is logged in, he does not have to log in again, unless he presses the sign out button.



"SHIL"
Id: "3632-eb43-a971-c0a8-4089-aff9",
Value: "8.594108179586463"

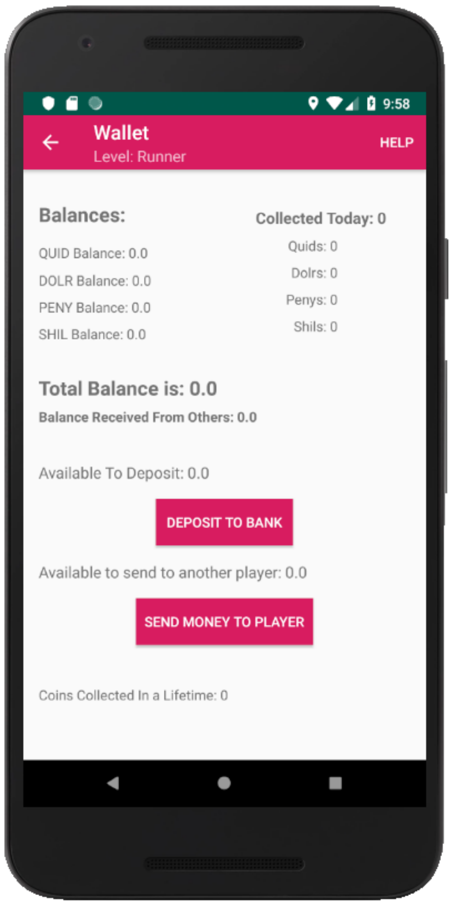


Congratulations! Coin "SHIL" has been collected!



Total balance: 2514.14306640625 GOLD

Map is shown right when the user is logged in. Each marker can be clicked and info about the marker are shown. User can access quick wallet presented in bottom left screenshot, to see his balance without accessing main wallet. Once the user is closer than 25 metres to the coin, coin is collected automatically and toast is presented, as can be seen in top right screenshot. Level of the user can be seen just below the game name in the main menu.


9:58

Wallet
Level: Runner
HELP

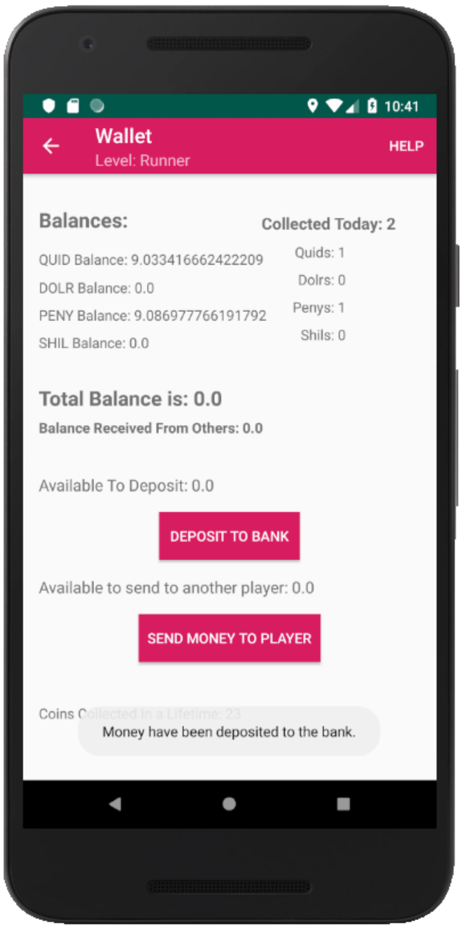
Balances:
QUID Balance: 0.0
DOLR Balance: 0.0
PENY Balance: 0.0
SHIL Balance: 0.0

Collected Today: 0
Quids: 0
Dolrs: 0
Penys: 0
Shils: 0

Total Balance is: 0.0
Balance Received From Others: 0.0

Available To Deposit: 0.0
DEPOSIT TO BANK
Available to send to another player: 0.0
SEND MONEY TO PLAYER

Coins Collected In a Lifetime: 0


10:41

Wallet
Level: Runner
HELP

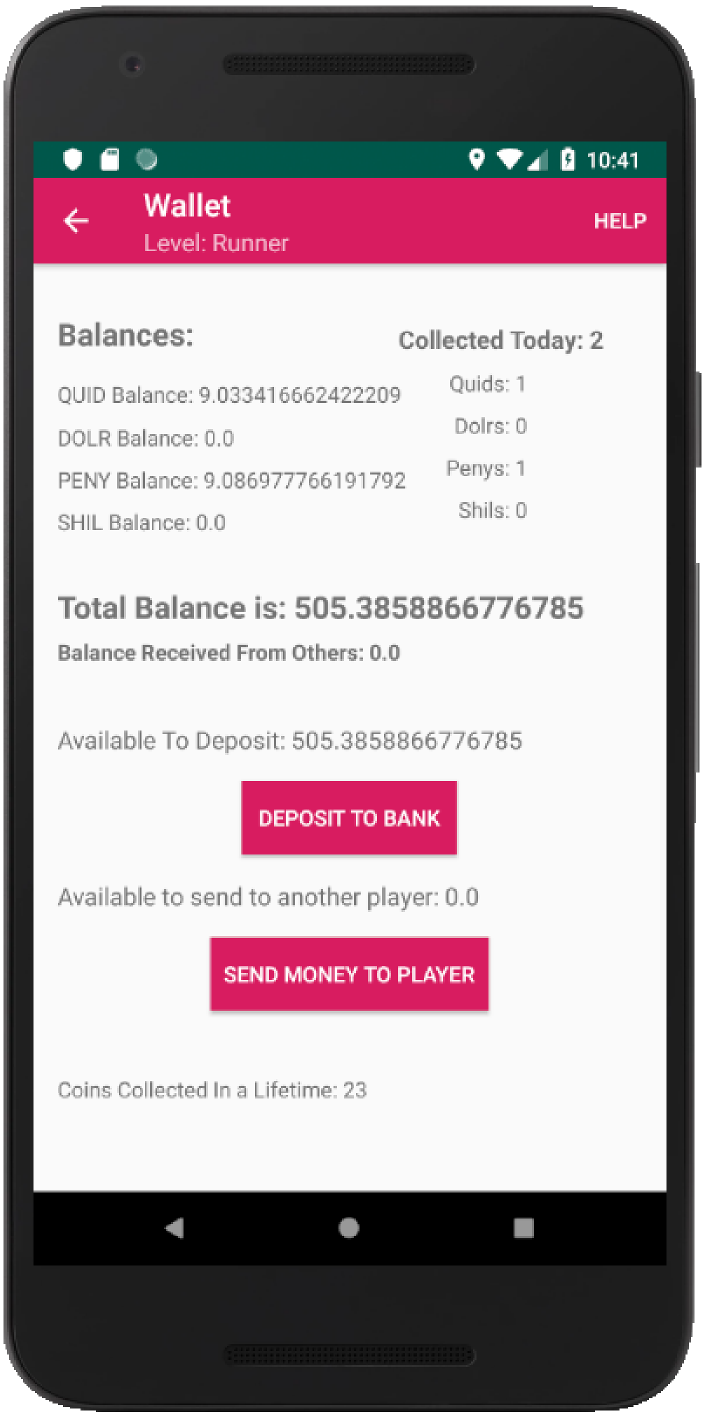
Balances:
QUID Balance: 9.033416662422209
DOLR Balance: 0.0
PENY Balance: 9.086977766191792
SHIL Balance: 0.0

Collected Today: 2
Quids: 1
Dolrs: 0
Penys: 1
Shils: 0

Total Balance is: 0.0
Balance Received From Others: 0.0

Available To Deposit: 0.0
DEPOSIT TO BANK
Available to send to another player: 0.0
SEND MONEY TO PLAYER

Coins Collected In a Lifetime: 23
Money have been deposited to the bank.


10:41

Wallet
Level: Runner
HELP

Balances:
QUID Balance: 9.033416662422209
DOLR Balance: 0.0
PENY Balance: 9.086977766191792
SHIL Balance: 0.0

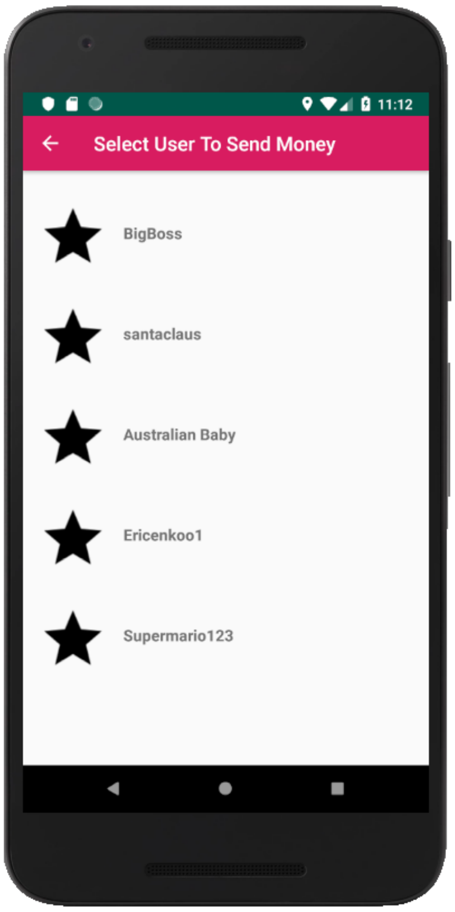
Collected Today: 2
Quids: 1
Dolrs: 0
Penys: 1
Shils: 0

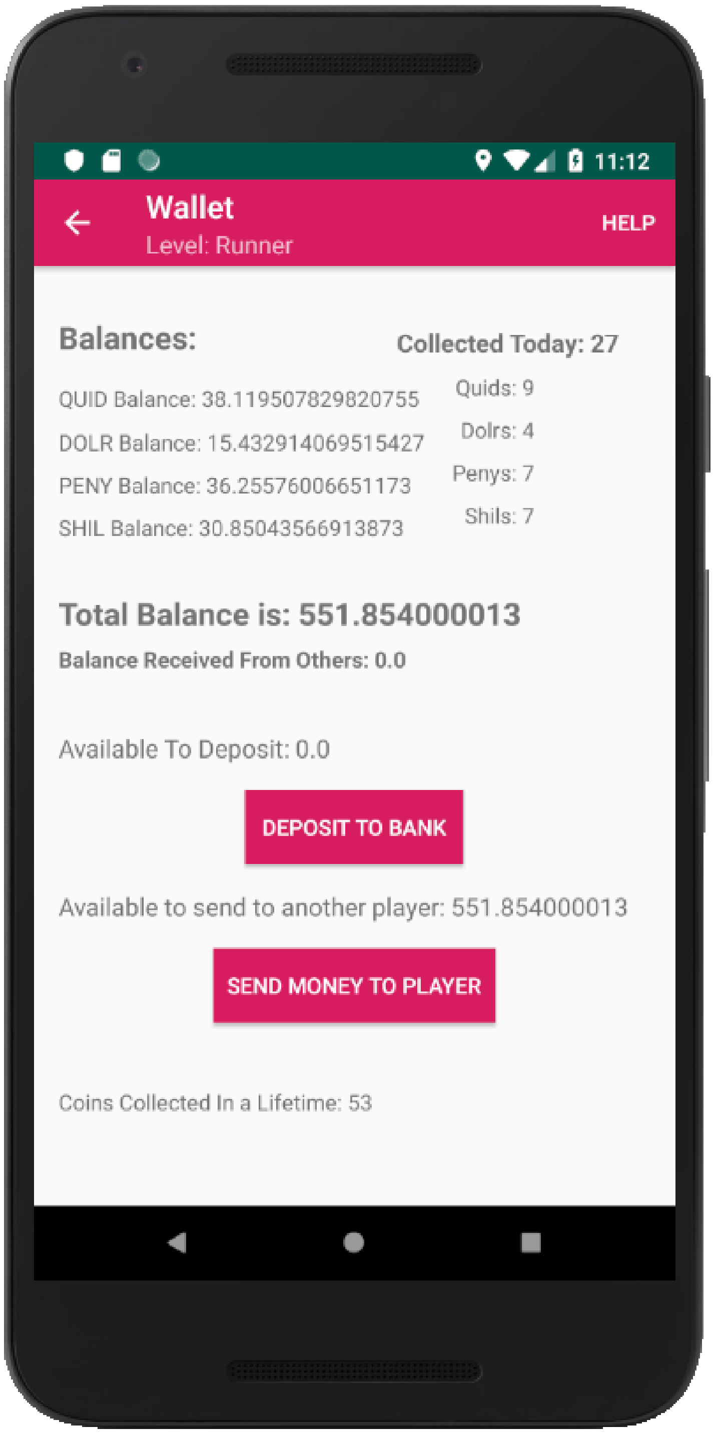
Total Balance is: 505.3858866776785
Balance Received From Others: 0.0

Available To Deposit: 505.3858866776785
DEPOSIT TO BANK
Available to send to another player: 0.0
SEND MONEY TO PLAYER

Coins Collected In a Lifetime: 23

Once the wallet is clicked and no coins have been collected yet, top right screen is presented. The user collects some coins and if he wants to deposit them to the bank, he can do so by clicking “Deposit to bank” button. If the deposit is possible, toast with success message is presented, as can be seen at bottom left screenshot. Balances in the wallet are subtracted,




Wallet

Level: Runner

HELP

Balances:

Collected Today: 27

QUID Balance: 38.119507829820755 Quids: 9

DOLR Balance: 15.432914069515427 Dolrs: 4

PENY Balance: 36.25576006651173 Penys: 7

SHIL Balance: 30.85043566913873 Shils: 7

Total Balance is: 551.854000013

Balance Received From Others: 0.0

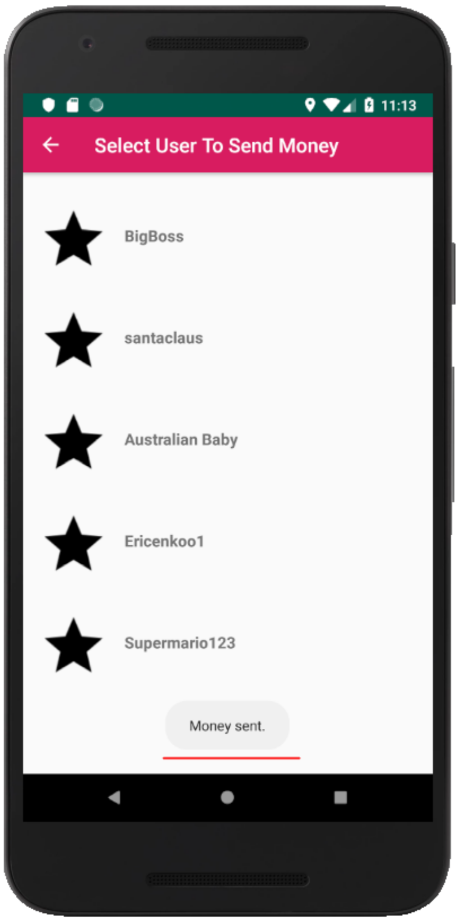
Available To Deposit: 0.0

DEPOSIT TO BANK

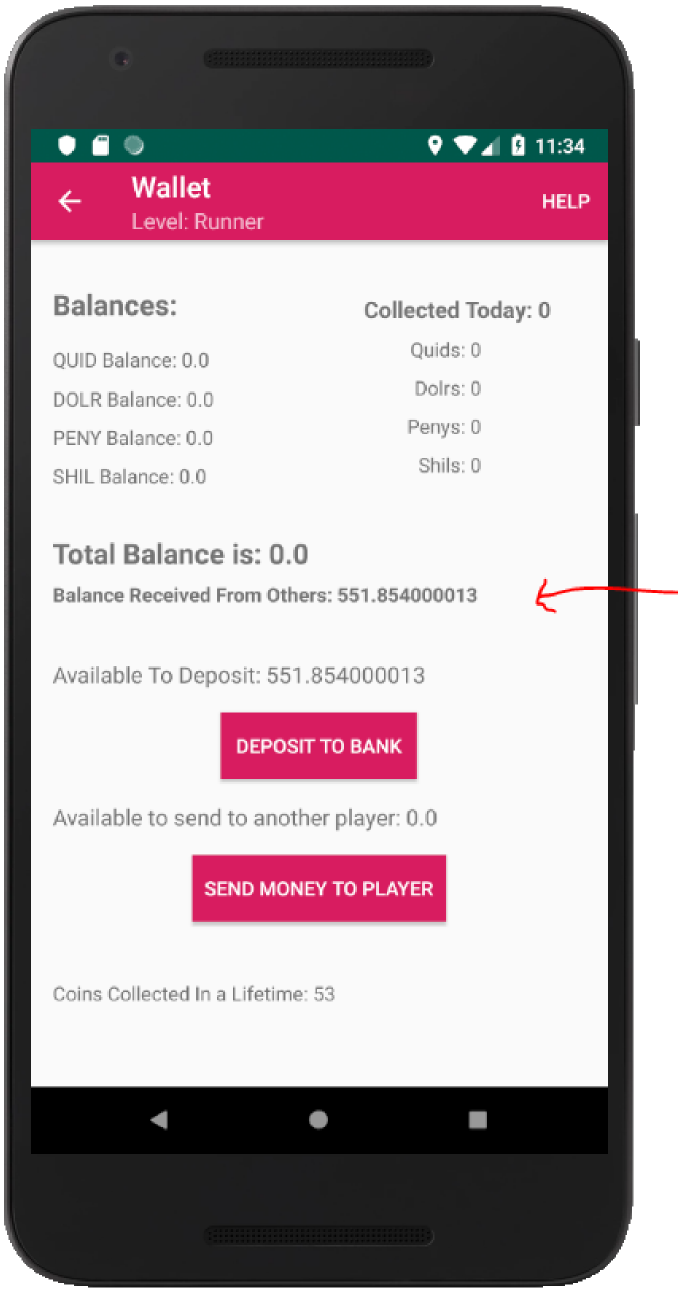
Available to send to another player: 551.854000013

SEND MONEY TO PLAYER

Coins Collected In a Lifetime: 53



The user can send money to other users only from the coins he collected after the 25th coin collection of that day. Then he would be able to click “Send money to player” button, he is redirected to another window as seen in top left screenshot. Once the user clicks the name of other player, the money is sent, balance is subtracted from his account and added to the balance of the other user.



11:34

Wallet

Level: Runner

HELP

Balances:

QUID Balance: 0.0

DOLR Balance: 0.0

PENY Balance: 0.0

SHIL Balance: 0.0

Collected Today: 0

Quids: 0

Dolrs: 0

Penys: 0

Shils: 0

Total Balance is: 0.0

Balance Received From Others: 551.854000013

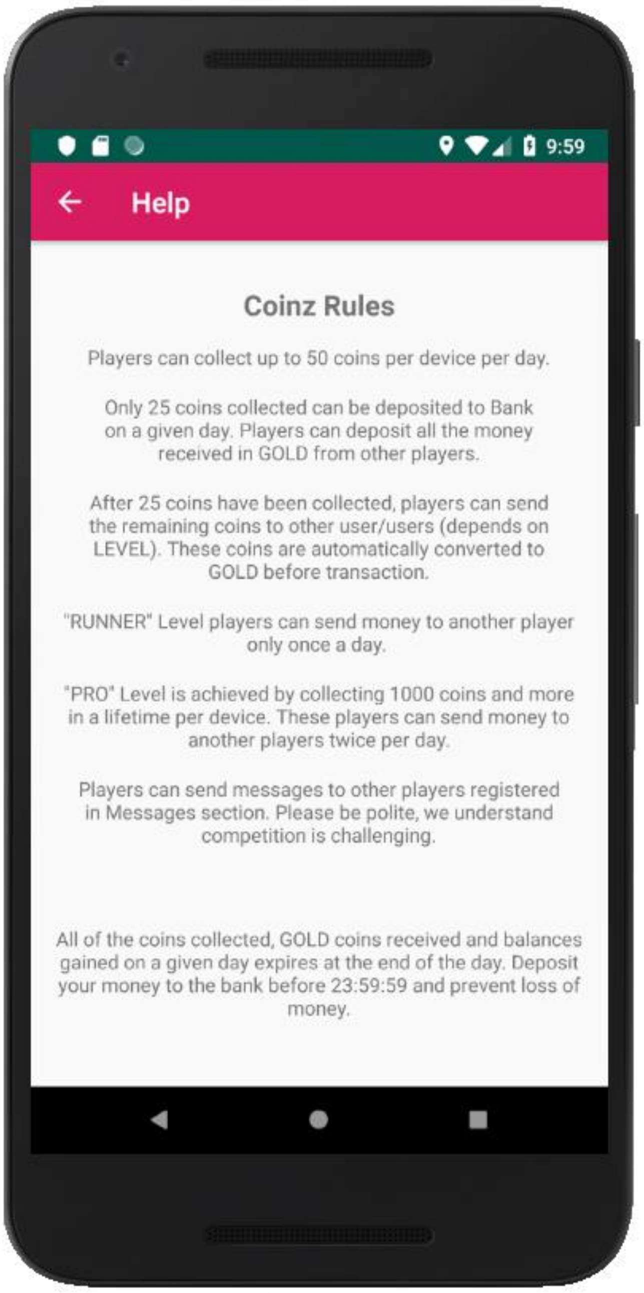
Available To Deposit: 551.854000013

DEPOSIT TO BANK

Available to send to another player: 0.0

SEND MONEY TO PLAYER

Coins Collected In a Lifetime: 53



9:59

Help

Coinz Rules

Players can collect up to 50 coins per device per day.

Only 25 coins collected can be deposited to Bank on a given day. Players can deposit all the money received in GOLD from other players.

After 25 coins have been collected, players can send the remaining coins to other user/users (depends on LEVEL). These coins are automatically converted to GOLD before transaction.

"RUNNER" Level players can send money to another player only once a day.

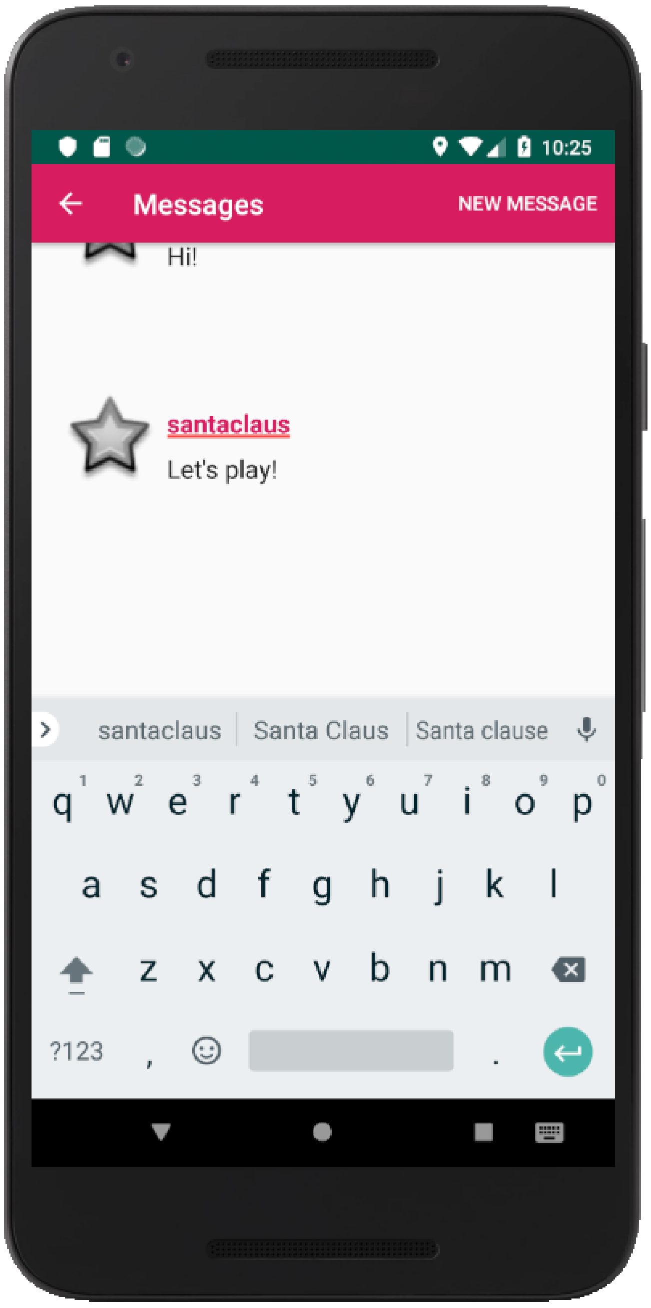
"PRO" Level is achieved by collecting 1000 coins and more in a lifetime per device. These players can send money to another players twice per day.

Players can send messages to other players registered in Messages section. Please be polite, we understand competition is challenging.

All of the coins collected, GOLD coins received and balances gained on a given day expires at the end of the day. Deposit your money to the bank before 23:59:59 and prevent loss of money.

Once the money is sent, the recipient receives the money as “Balance received from others”. This can be seen in top left screenshot. Recipient can deposit this money to his bank account. The number of players that the user can send the money to is dependent on the level of the user.

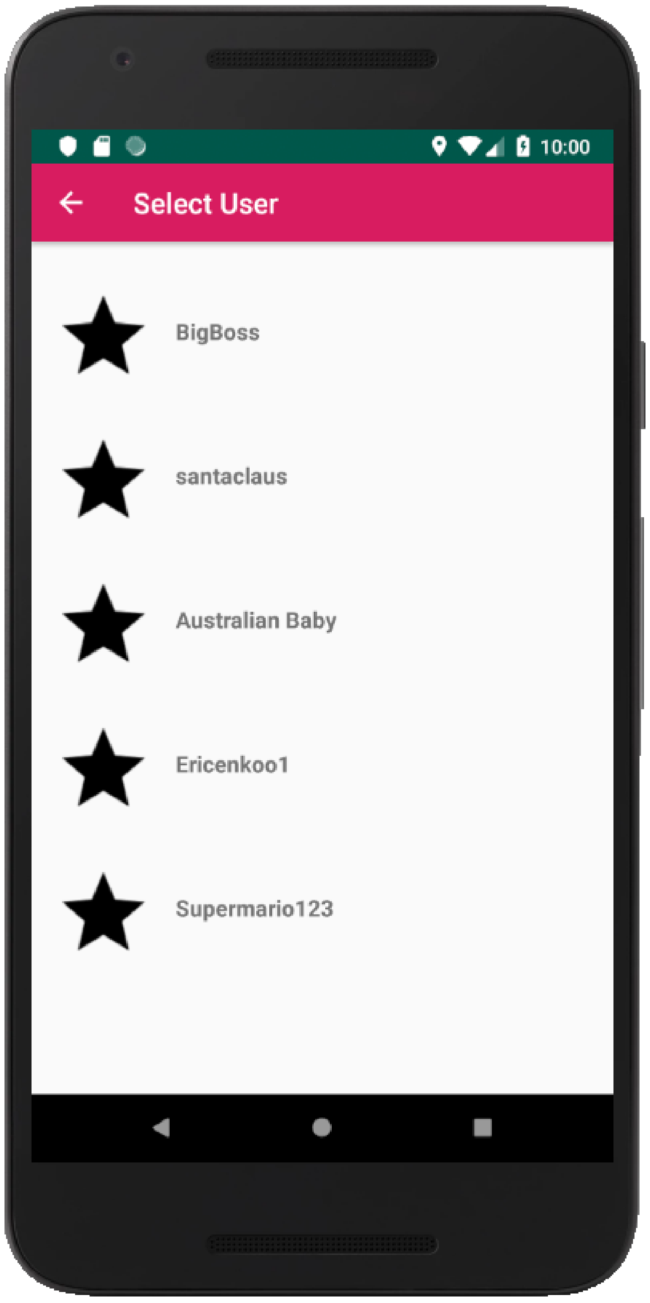
Top right screenshot shows the Help activity. This screen can be accessed from Main Activity once the user clicks the “?” floating button or “Help” from the Wallet menu.



Hi!

santaclaus

Let's play!



BigBoss

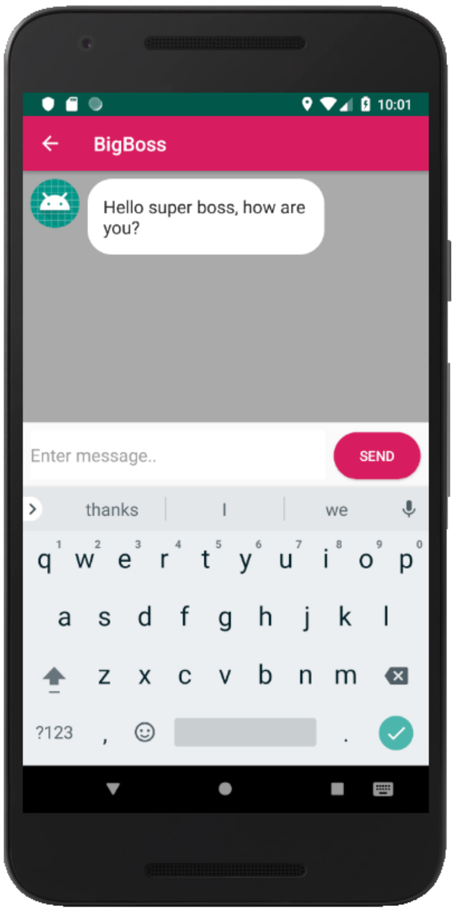
santaclaus

Australian Baby

Ericenkoo1

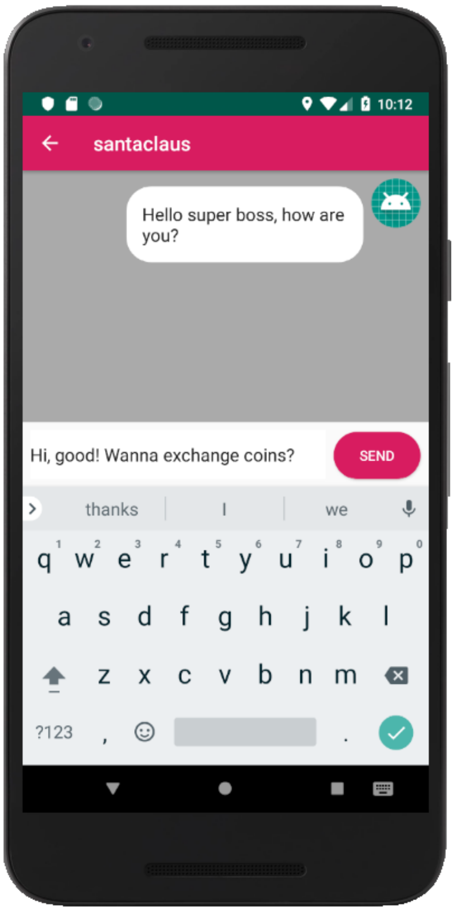
Supermario123

Messages can be accessed from Main Activity. Once the user clicks “Messages”, he is redirected to Messages screen, as seen in top left screenshot. This screen is showing the latest messages received from other players. By clicking “New Message” in the Messages menu, top right screen is presented. User can choose the player he wants to send the message to.



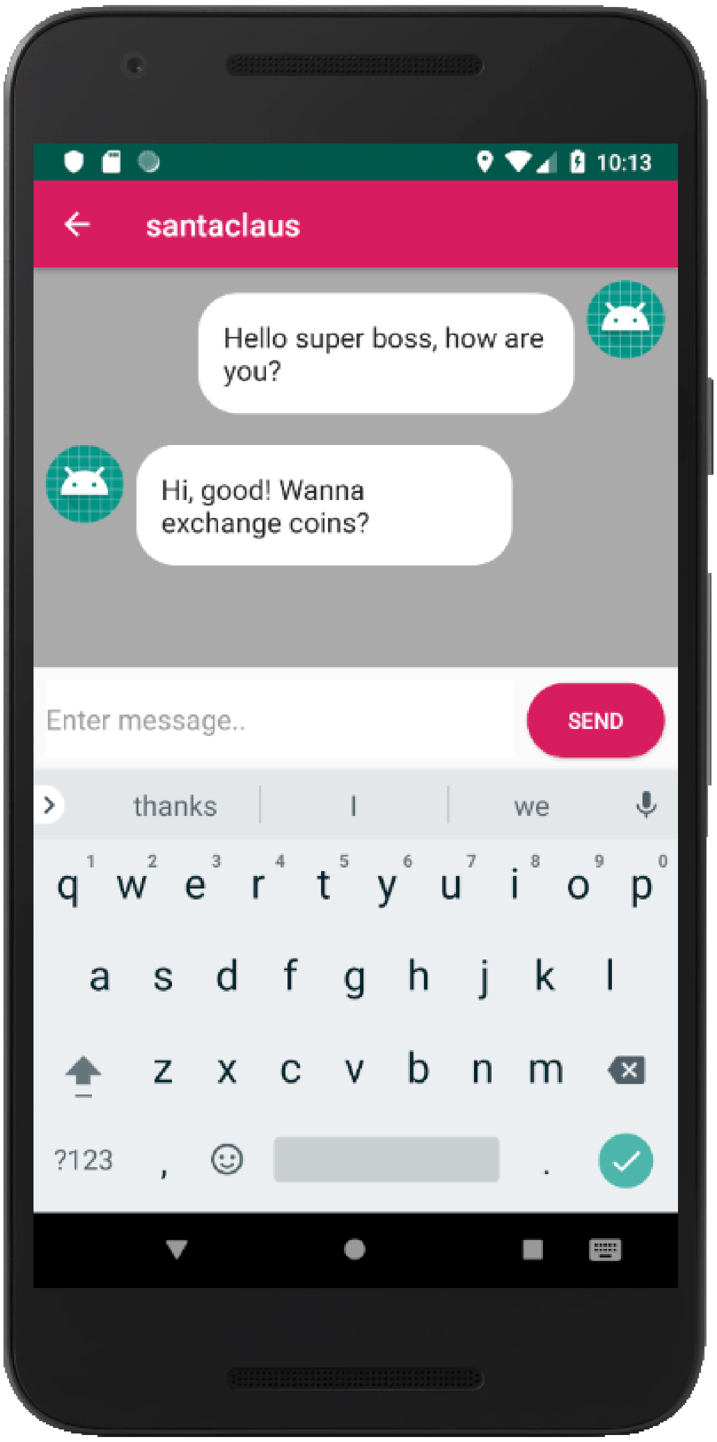
Hello super boss, how are you?

Enter message..



Hello super boss, how are you?

Hi, good! Wanna exchange coins?



Hello super boss, how are you?

Hi, good! Wanna exchange coins?

Enter message..

Then, the private chat window can be seen. It can be seen only by these two players. Here, the players can discuss exchange of the coins, meetups, challenges or just talk, but they can't send the money. Sending money needs to be done from the wallet as was presented before.

## 5 Acknowledgements

1. Getting started with Kotlin Messenger. Available at: https://www.youtube.com/watch?v=SuRiwVF5bzs&index=4&list=PL0dzCUj1L5JE-jiBHjxlmXEkQkum_M3R-

2. OkHttp3 client for android and java applications. Available at: https://square.github.io/okhttp/3.x/okhttp/

3. Groupie Adapter. Available at: https://github.com/lisawray/groupie

4. Fetch and Parse JSON with OkHttp and Gson. Available at: https://www.youtube.com/watch?v=53BsyxwSBJk&t=1272s