INFORMATICS LARGE PRACTICAL COURSEWORK 2 REPORT

Frederik Kelbel
s1653529

OVERVIEW

Terminology and general information

Firebase structures

1. Algorithms and data structures

This section contains information about features of the implementation which make up the core of the application. It clarifies how the required components were realized in the final version.

2. Elements that have not made it into the final application

This section contains information about additional features of the implementation which were described in the report of coursework 1 but did not make it into the final application.

3. New elements that have made it into the final application

This section contains information about additional features of the implementation which were not described in the report of coursework 1.

4. Bonus features and their structure

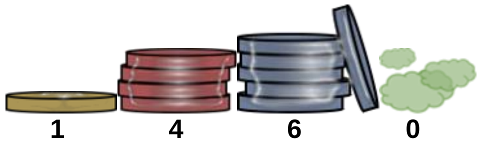
This section contains information about features of the implementation which go beyond the core of the application. It clarifies how the additional components were realized in the final version.

5. Acknowledgements

This section contains information on any features that made use of publicly-available sources and acknowledges those.

Terminology and general information

Coins are generally referred to as coin items here. Coin items are collected and put into a currency bag/sac. The four different currency bags make up the user's wallet. This simply allows me to use the icons on the right to display the locations of the coins on the map. The yellow icon on the right for example is worth [1, 2) QUID. The blue icon on the right is worth [6, 7) DOLR. You can click on the icon on the map to get the exact value. The icons on the right substitute for the marker-symbol. It simply looks more interesting than with normal markers.



Various examples of coin items

SHIL
DOLR
QUID
PENY

Currency sacs in your wallet

Activities and fragments with their purpose:

MainActivity: Manages Log in and retrieval of user data
SignUpActivity: Manages Sign up and user data set up
ProfileActivity: Manages user data while playing, hosts all fragments
FragmentDepot: Hosts subfragments in viewpager, displays user name, gold and team
SubFragmentAccount: Displays which team provides the better advantage for its members
SubFragmentExchange: Displays exchange rates and links to ExchangePopUp
SubFragmentTrading: Links to MakeanOffer, BrowseOffers
SubFragmentEvents: Events can be started/accepted from here (Bonus feature)
FragmentMap: Displays map with coin locations and team zones
FragmentSettings: Provides opportunity to customize profile
TeamZone: User can purchase campus zones here (Bonus feature)
MakeanOffer: Place coin items as an offer and ask for gold in return (coin messaging)
BrowseOffers: Accept offers here (coin messaging)
ExchangePopUp: Exchange coin items for gold here (banking coins)

Clearing up on underspecificaction and additional comments:

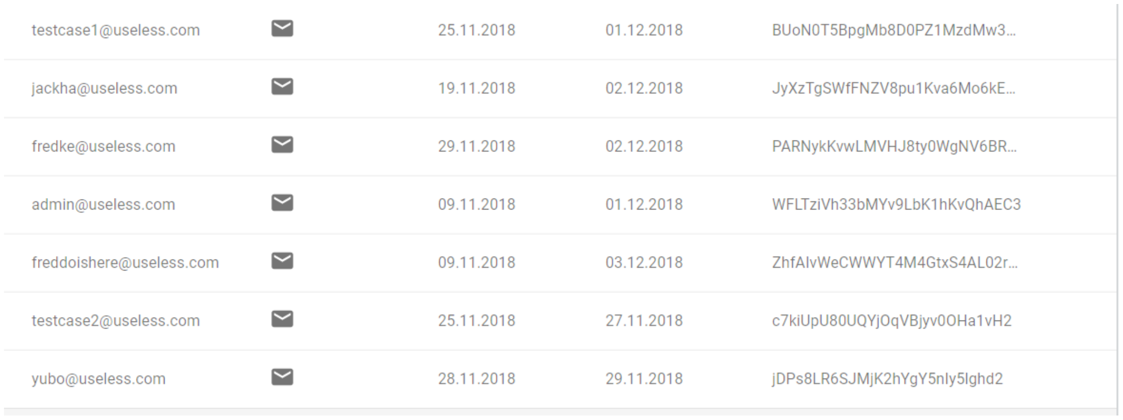
Coin items are converted to gold at the time that they are paid into the bank/exchanged. Coin items do not expire in your wallet and can be held indefinitely. Coins gathered from the event reduce your daily exchange capacity of 25. Traded/Messaged coins do not reduce it. Users can hold duplicates of traded coins. Traded coins are marked with a suffix to their ID.

Available Tests

There are five tests available. It tests the basic UI, the messaging of coins, the receiving of coins from a message, the banking of the coins and the gathering of coins along a route on the map of the 24th of November 2018. Each test contains a step by step description on what it does. The test that tests the collection of coins also sets all team zones to blue for the purpose of guaranteeing the collection along the chosen route for the test.

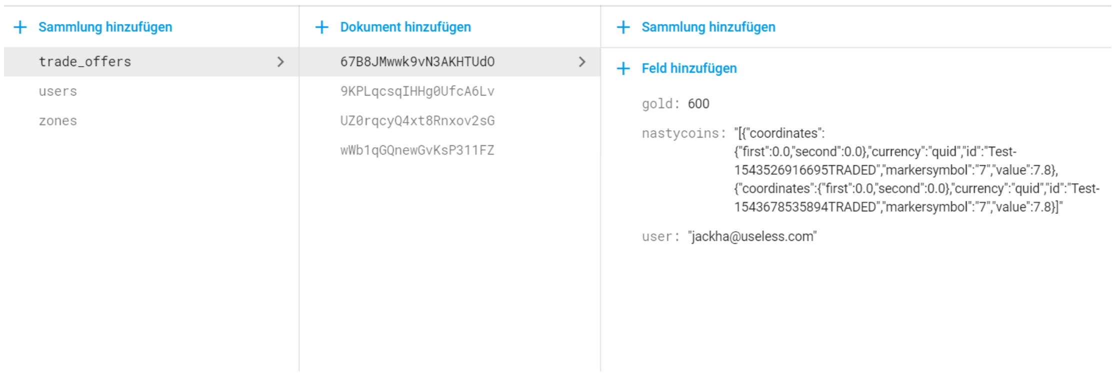
Terminology and general information:

Authentication profile examples:



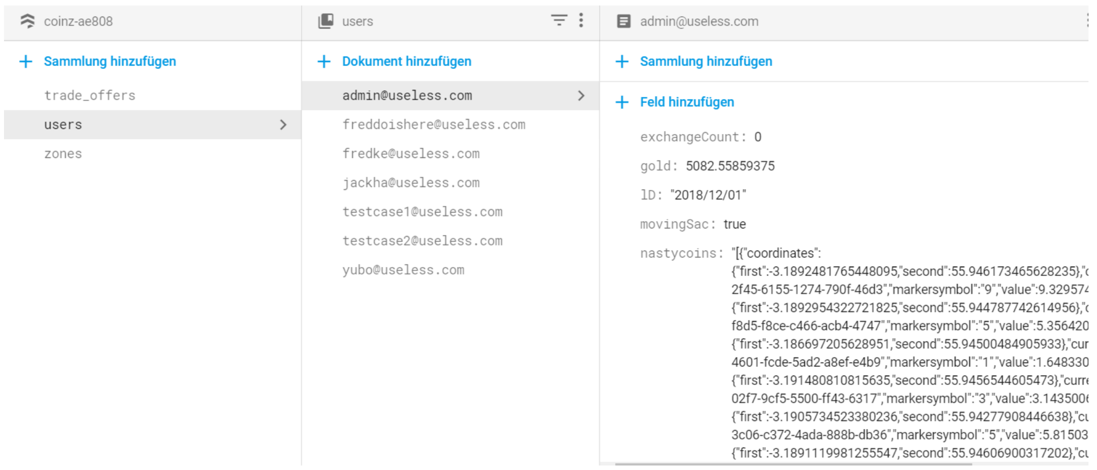
testcase1@useless.com	✉	25.11.2018	01.12.2018	BUoN0T5BpgMb8D0PZ1MzdMw3...
jackha@useless.com	✉	19.11.2018	02.12.2018	JyXzTgSWfFNZV8pu1Kva6Mo6kE...
fredke@useless.com	✉	29.11.2018	02.12.2018	PARNykKvwLMVHJ8ty0WgNV6BR...
admin@useless.com	✉	09.11.2018	01.12.2018	WFLTziVh33bMYv9LbK1hKvQhAEC3
freddoishere@useless.com	✉	09.11.2018	03.12.2018	ZhfAlvWeCWWYT4M4GtxS4AL02r...
testcase2@useless.com	✉	25.11.2018	27.11.2018	c7kiUpU80UQYjOqVBjyv0OHa1vH2
yubo@useless.com	✉	28.11.2018	29.11.2018	jDPs8LR6SJMjK2hYgY5nly5lghd2

Trade offers examples:



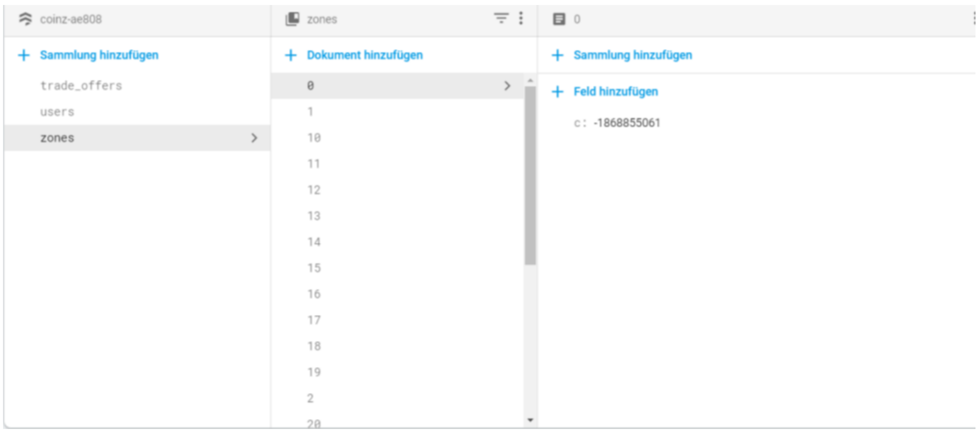
```
gold: 600
nastycoins: "[{"coordinates":
            {"first":0.0,"second":0.0},"currency":"quid","id":"Test-
            1543526916695TRADED","markersymbol":"7","value":7.8},
            {"coordinates":{"first":0.0,"second":0.0},"currency":"quid","id":"Test-
            1543678535894TRADED","markersymbol":"7","value":7.8}]"
user: "jackha@useless.com"
```

User base examples:



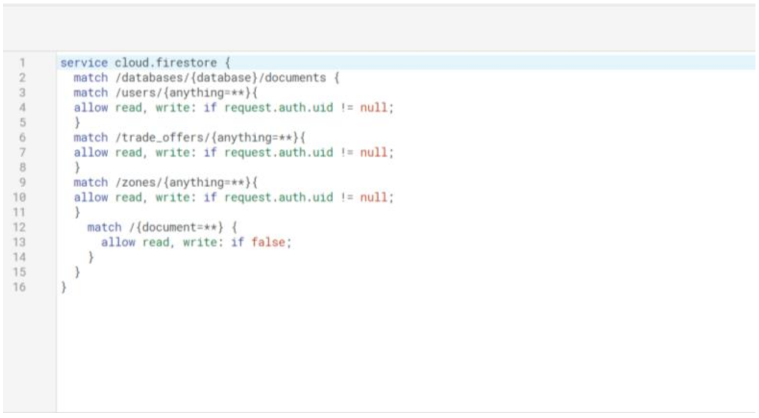
```
exchangeCount: 0
gold: 5082.55859375
lD: "2018/12/01"
movingSac: true
nastycoins: "[{"coordinates":
            {"first":-3.1892481765448095,"second":55.946173465628235},"c
            2f45-6155-1274-790f-46d3","markersymbol":"9","value":9.32957
            {"first":-3.1892954322721825,"second":55.944787742614956},"c
            f8d5-f8ce-c466-acb4-4747","markersymbol":"5","value":5.356420
            {"first":-3.186697205628951,"second":55.94500484905933},"cur
            4601-fcde-5ad2-a8ef-e4b9","markersymbol":"1","value":1.648330
            {"first":-3.191480810815635,"second":55.9456544605473},"curre
            02f7-9cf5-5500-ff43-6317","markersymbol":"3","value":3.143500
            {"first":-3.1905734523380236,"second":55.94277908446638},"cu
            3c06-c372-4ada-888b-db36","markersymbol":"5","value":5.81503
            {"first":-3.1891119981255547,"second":55.94606900317202},"cu
```

Zone example configuration:



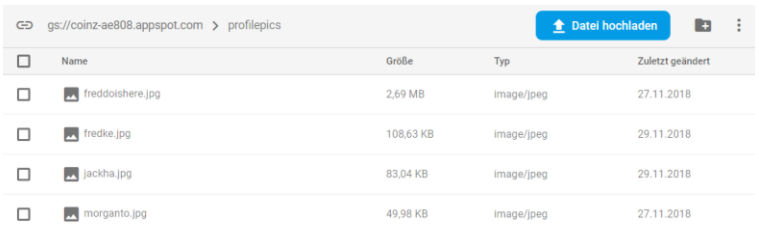
c: -1868855061

Rules:



```
service cloud.firestore {
  match /databases/{database}/documents {
  match /users/{anything=**}{
  allow read, write: if request.auth.uid != null;
  }
  match /trade_offers/{anything=**}{
  allow read, write: if request.auth.uid != null;
  }
  match /zones/{anything=**}{
  allow read, write: if request.auth.uid != null;
  }
    match /{document=**} {
      allow read, write: if false;
    }
  }
}
```

Firebase storage for profile pics:



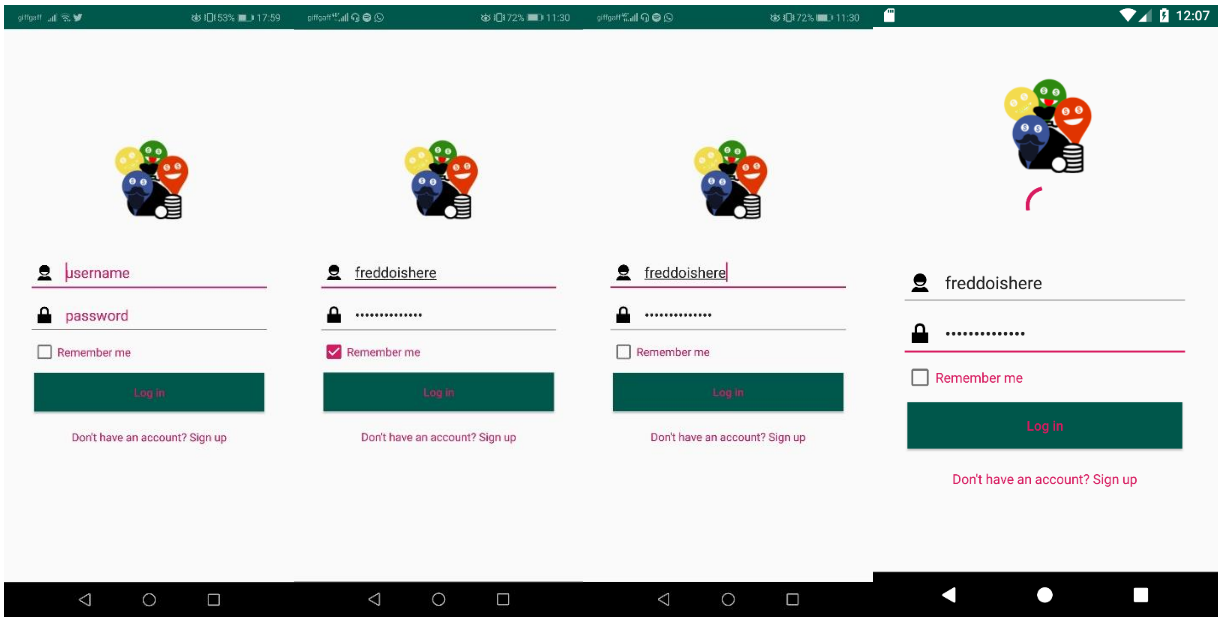
Name	Größe	Typ	Zuletzt geändert
freddoishere.jpg	2,69 MB	image/jpeg	27.11.2018
fredke.jpg	108,63 KB	image/jpeg	29.11.2018
jackha.jpg	83,04 KB	image/jpeg	29.11.2018
morganto.jpg	49,98 KB	image/jpeg	27.11.2018

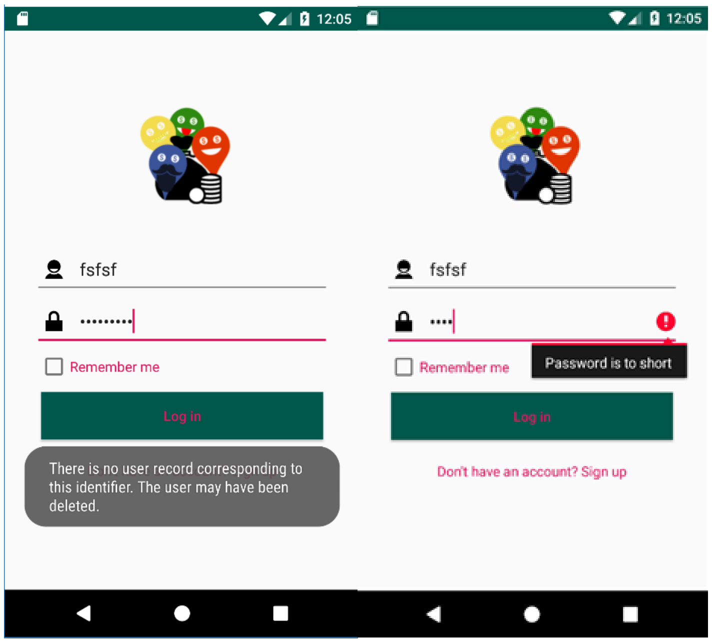
Algorithms and data structures

Log in (MainActivity.kt)

Essential function calls in chronological order:
signInWithEmailAndPassword() ➔ Firebase authentication
getUserAccount() ➔ Retrieving data from Firestore

The user must fill out two EditText-fields. The first field is for the username. The second field is for the password of the respective account. After pressing the Log-in-Button the string “@useless.com” is added to the username. Thus, we can use signInWithEmailAndPassword() without the need of entering an email-address and risking spam from the exposure to other users. If the log-in is successful getUserAccount() will be called. It retrieves the user’s data from Firestore and assigns the respective values to the variables in the companion object of ProfileActivity. The password and username are being saved to shared preferences if the user ticks the provided checkbox. When the user opens the app again the text of the EditText-fields is being filled automatically with the data stored in the shared preferences.


username
password
Remember me
Log in
Don't have an account? Sign up
freddoishere
Remember me
Log in
Don't have an account? Sign up
freddoishere
Remember me
Log in
Don't have an account? Sign up
freddoishere
Remember me
Log in
Don't have an account? Sign up
Log in page, loading circle appears, screenshots taken from various phones


fsfsf
Remember me
Log in
There is no user record corresponding to this identifier. The user may have been deleted.
fsfsf
Remember me
Password is to short
Log in
Don't have an account? Sign up
Some of the error messages that can appear, screenshots taken from various phones

Sign up (SignUpActivity.kt)

Essential function calls in chronological order:

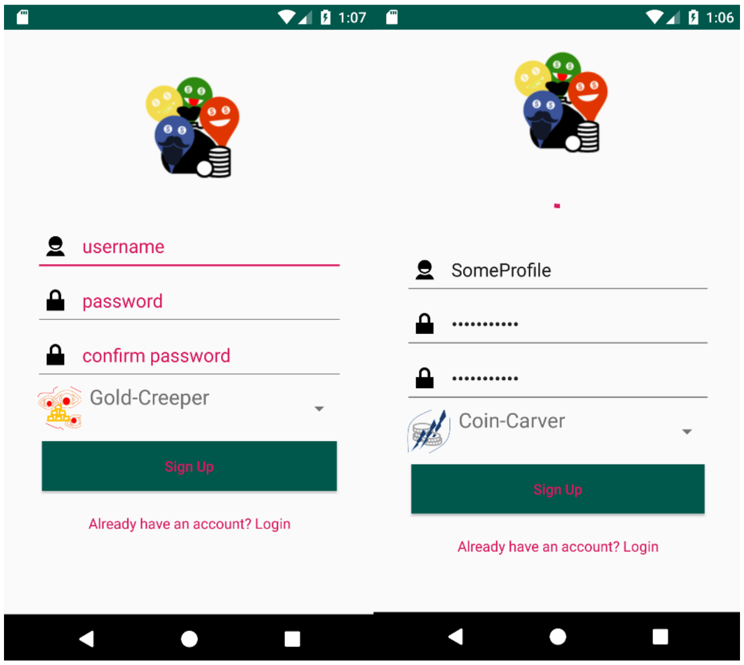
createUserWithEmailAndPassword() ➔ Firebase user creation
setUpUserAccount() ➔ Setting up user data in Firestore and app

The user must fill out three EditText-fields (username, password, password confirmation). The user then selects a team through a spinner-element (see 3). After pressing the Signup-button the string "@useless.com" is added to the username. We use createUserWithEmailAndPassword() to create the account. If the sign-up is successful setUpUserAccount() is called. It puts the essential variables of the companion object of ProfileActivity into a HashMap and stores said in Firestore under a user-specific document in a collection of users. On success it starts ProfileActivity and sets it companion object's variables to the basic starting values. Every user document contains the following variables that are also all in the companion object of ProfileActvity:

team: Int ➔ 0 for team Goldcreeper, 1 for team Coincarver
downloadDate: String ➔ date when the last map was downloaded
exchangedCount: Int ➔ amount of coins exchanged today, limited to 25
gold: Float ➔ amount of gold in the bank
wallet: Wallet ➔ 4 instances of ArrayList<NastyCoin> containing coin items of the re-spective currency in your wallet
nastycoins: ArrayList<NastyCoin> ➔ List of coins that have not been collected today

coinExchangeRates: ArrayList<CoinExchangeRates> ➔ List of the exchange rates of the four different currencies from today to a week ago
SubFragmentEvent.eventAvailability ➔ stored in companion object of SubFragmetEvent to indicate whether the "Catch me if you can"-event is available, set to false after participation in event has taken place until the end of the day

plus ➔ this is not in the companion object and only in the Firestore user document; It increases when the user receives gold from a trade. A SnapshotListener then updates the user's gold based on value in plus and sets plus to zero



Sign up page, loading circle appears after sign up, screenshots taken from various phones

Handling the user's progress and storing data

Essential function calls in chronological order:

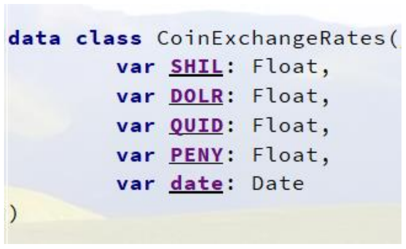
OnPause() ➔ calls saveProgress()
saveProgress() ➔ upload of data to Firestore

Generally, the storing of the game's data is handled through a function named saveProgress(). The function is called in the onPause()-method of ProfileActivity. OnPause() is more likely to be called after quitting the app than onDestroy()/onStop(). Hence, data is always stored when the user quits the app or opens another activity in front of ProfileActivity. The data is always uploaded directly to Firestore to make the user's data accessible from different devices. The exchange rates for the coin items are not uploaded to Firestore. They are stored in the shared preferences so that users who use the same device multiple times a day do not have to download the exchange-rates again every time they open the app. This also means that another user who uses the same device must not download the rates again. It is not necessary to store anything else locally since the device will retry to save to Firestore in case of a bad connection until it succeeds and stores it in the cache in the meantime. While ProfileActivity is active and in use the players data is stored in local variables in a companion object.

Download and parsing of the map

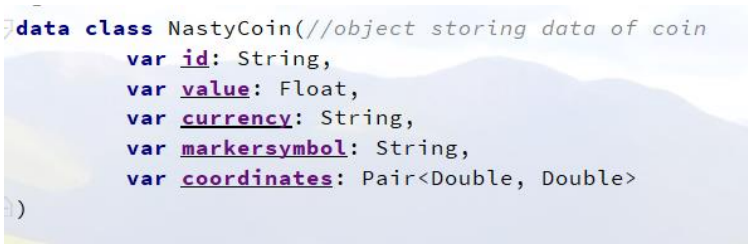
Essential function calls in chronological order:
OnViewCreated() ➔ executes DownloadFileTask()
DownloadFileTask()➔ calls retrieveCoins()
➔ callback to ProfileActivity to start rendering/addMarkers()
retrieveCoins() ➔ parses all coin items of the new map
OnViewCreated() (SubFragmentExchange)➔ executes DownloadExchangeRates()
DownloadExchangeRates() ➔ calls retrieveCoinExchangerates()
➔ callback to SubFragmentExchange ➔ initGraphs()
initGraphs() ➔ initializes graphs

With the OnViewCreated() in ProfileActivity we call an asynchronous task which downloads the daily map. This task is only executed if the last download date that we retrieve from Firestore (user specific) does not match the current date. The task connects to the website and opens an input stream that is read into a string. We then call a function on that string that parses the new coin items. The string is converted into a json-Object on which we call retrieveCoins(). The coin items are parsed with json.getJSONArray("features") which gives us an array of JSONObjects and then with getJSONObject("properties") on each object secondly. Each JSONObject is then stored in an instance of a data class (Fig. 1.2). When the asynchronous task is finished ProfileActivity receives a callback and tries to render coin items on the map (addMarkers()). addMarkers() uses the currency and the marker-symbol and concatenates them to us in resources.getIndentifier() for the retrieval of the fitting marker icon to display on the map. Then the respective icon is put on the map with the functions available through MapBox. If the map is not initialized by the time the task finishes it retries every two seconds until it succeeds. This can happen when the map from the MapBox servers requires a lot of time to load. The user can still make use of all the other features of the app in the meantime. An array of the uncollected coin items is stored in Firestore. Right after the download that array consists of fifty coin items. Coin items are being removed from that array when they are collected. The map must be downloaded and parsed only once a day. The GeoJson file of the map is not stored since we have the uncollected objects stored in Firestore. As an additional design feature the exchange rates of the past week are displayed through graphs in the “Exchange”-tab. These rates together with the current rate are retrieved in a similar manner but instead of putting the whole GeoJson file into an input stream it is managed through an additional asynchronous task which uses buffered input streams to only read the very beginning of the files resulting in less data usage and fatser download. The rates from the current date back to one week before that are stored in an array of exchange rate objects (Fig. 1.1). The second task is called on initialization of the SubFragmentExchange tab which is done directly after ProfileActivity is initialized. retrieveCoinExchangerates() takes the JSONObject “rates” and gets the respective strings for the currencies. SubFragmentExchange also receives a callback and plots the graphs. Both arrays are variables in the companion object of ProfileActivity.



```
data class CoinExchangeRates(
        var SHIL: Float,
        var DOLR: Float,
        var QUID: Float,
        var PENY: Float,
        var date: Date
)
```

Fig. 1.1



```
data class NastyCoin(//object storing data of coin
        var id: String,
        var value: Float,
        var currency: String,
        var markersymbol: String,
        var coordinates: Pair<Double, Double>
)
```

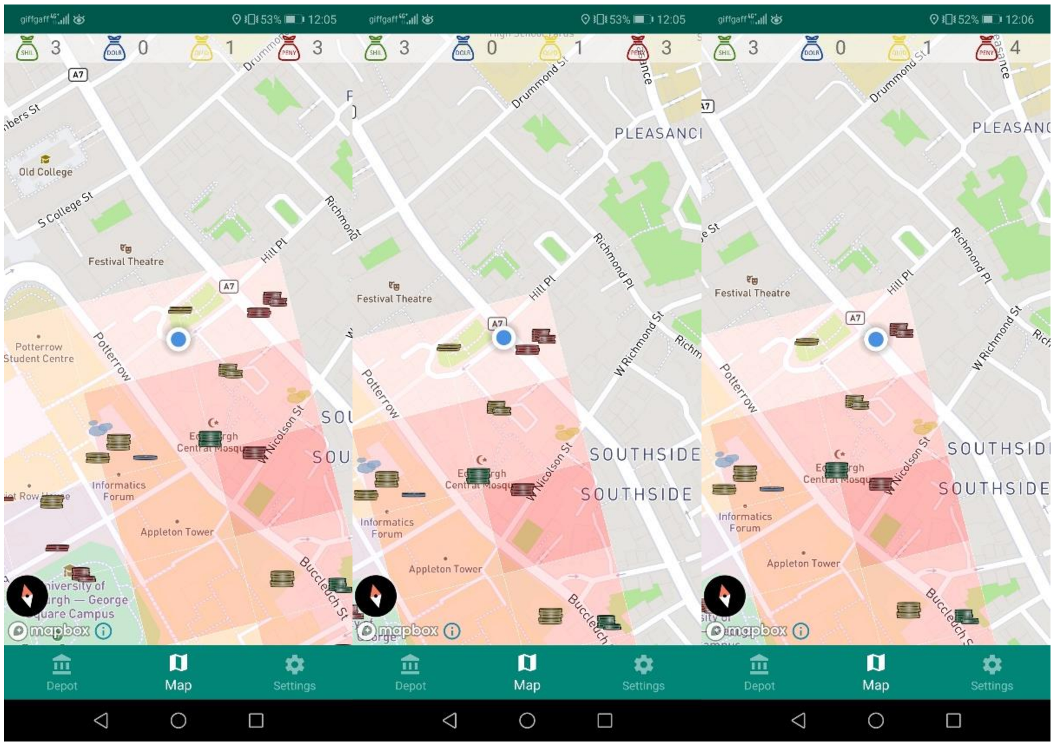
Fig. 1.2

Collection of coin items

Essential function calls in chronological order:

OnLocationChanged() ➔ calls checkForCoin()
checkForCoin() ➔ calls compareCoordinates() on every uncollected coin item
compareCoordinates() ➔ Let’s checkForCoin() finish early if it returns a coin item in range

The location is updated every one to five seconds. When the location has changed OnLocationChanged() is called. In OnLocationChanged() we call a function named checkForCoin() on the new location. The function then compares the coordinates of every uncollected coin item with the given location. It collects the first coin item in the list of uncollected coin items that has a distance of 25m to the player. It makes use of the distanceTo() function of locations.



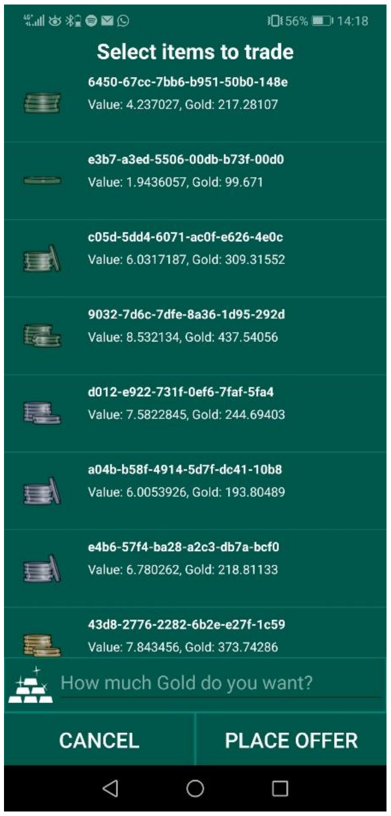
User is collecting PENY item, screenshots taken from various phones, maximum distance for collecting coins was much less 25m in pictures (old app version), submitted version uses range up to 25m

Messaging coin items (MakeanOffer.kt, BrowseOffers.kt)

Sending coin items off (MakeanOffer.kt)

Essential function calls in chronological order:
OnCreate() ➔ sets up adapter for ListView
onItemClick() ➔ calls selectOrUnselect()
selectOrUnselect() ➔ selects or unselects a coin item in the ListView
onClick() ➔ uploads the selected coin items to Firestore or cancels

With the creation of the MakeanOffer-Activity all coin items in the wallet are fed into an ArrayList of HashMaps with their basic information (id, value and corresponding gold value, currency). The HashMaps are then used to set up a SimpleAdapter as the adapter for the ListView. Clicking on a coin item results in a call of selectOrUnselect() which adds the corresponding coin item to an ArrayList representing the currently selected coin items. Another tap on the same item removes it from the ArrayList again. The user confirms his selection of coin items through a click on a button (onClick()). It puts the selected coin items in a HashMap consisting of user, gold (you can ask for any amount of gold in return for the coin items) and the coin items as it's attributes. This is then placed in Firestore. The object is placed as a document in a collection of documents called "tradeoffers". The object is generally going to be referred to as an offer.


56% 14:18

Select items to trade

6450-67cc-7bb6-b951-50b0-148e
Value: 4.237027, Gold: 217.28107

e3b7-a3ed-5506-00db-b73f-00d0
Value: 1.9436057, Gold: 99.671

c05d-5dd4-6071-ac0f-e626-4e0c
Value: 6.0317187, Gold: 309.31552

9032-7d6c-7dfe-8a36-1d95-292d
Value: 8.532134, Gold: 437.54056

d012-e922-731f-0ef6-7faf-5fa4
Value: 7.5822845, Gold: 244.69403

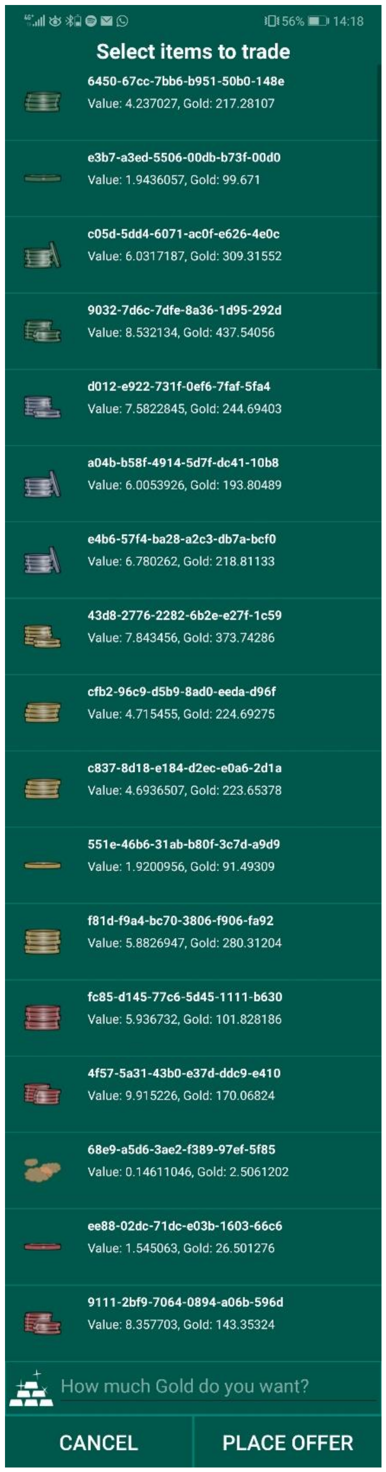
a04b-b58f-4914-5d7f-dc41-10b8
Value: 6.0053926, Gold: 193.80489

e4b6-57f4-ba28-a2c3-db7a-bcf0
Value: 6.780262, Gold: 218.81133

43d8-2776-2282-6b2e-e27f-1c59
Value: 7.843456, Gold: 373.74286

How much Gold do you want?

CANCEL PLACE OFFER


56% 14:18

Select items to trade

6450-67cc-7bb6-b951-50b0-148e
Value: 4.237027, Gold: 217.28107

e3b7-a3ed-5506-00db-b73f-00d0
Value: 1.9436057, Gold: 99.671

c05d-5dd4-6071-ac0f-e626-4e0c
Value: 6.0317187, Gold: 309.31552

9032-7d6c-7dfe-8a36-1d95-292d
Value: 8.532134, Gold: 437.54056

d012-e922-731f-0ef6-7faf-5fa4
Value: 7.5822845, Gold: 244.69403

a04b-b58f-4914-5d7f-dc41-10b8
Value: 6.0053926, Gold: 193.80489

e4b6-57f4-ba28-a2c3-db7a-bcf0
Value: 6.780262, Gold: 218.81133

43d8-2776-2282-6b2e-e27f-1c59
Value: 7.843456, Gold: 373.74286

cfb2-96c9-d5b9-8ad0-eeda-d96f
Value: 4.715455, Gold: 224.69275

c837-8d18-e184-d2ec-e0a6-2d1a
Value: 4.6936507, Gold: 223.65378

551e-46b6-31ab-b80f-3c7d-a9d9
Value: 1.9200956, Gold: 91.49309

f81d-f9a4-bc70-3806-f906-fa92
Value: 5.8826947, Gold: 280.31204

fc85-d145-77c6-5d45-1111-b630
Value: 5.936732, Gold: 101.828186

4f57-5a31-43b0-e37d-ddc9-e410
Value: 9.915226, Gold: 170.06824

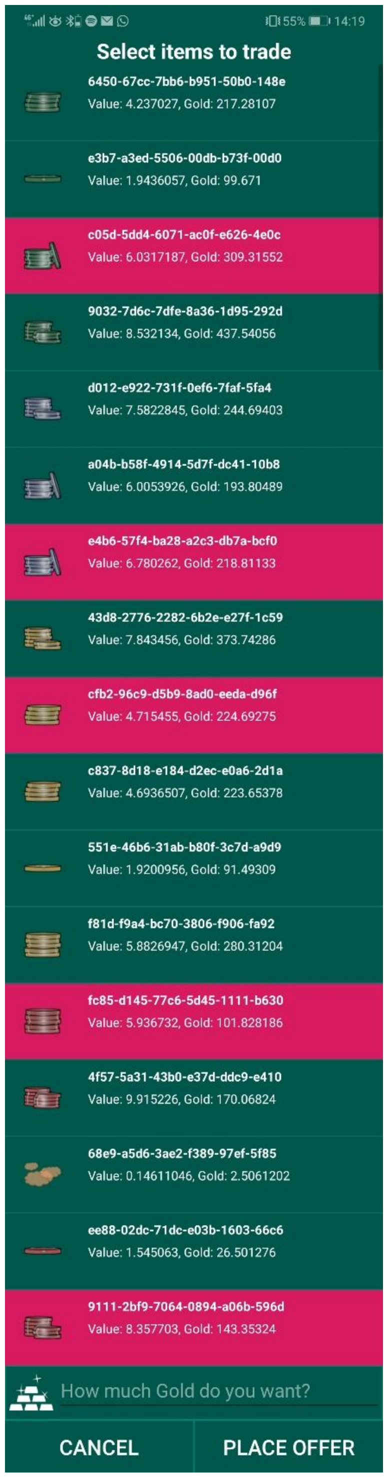
68e9-a5d6-3ae2-f389-97ef-5f85
Value: 0.14611046, Gold: 2.5061202

ee88-02dc-71dc-e03b-1603-66c6
Value: 1.545063, Gold: 26.501276

9111-2bf9-7064-0894-a06b-596d
Value: 8.357703, Gold: 143.35324

How much Gold do you want?

CANCEL PLACE OFFER


55% 14:19

Select items to trade

6450-67cc-7bb6-b951-50b0-148e
Value: 4.237027, Gold: 217.28107

e3b7-a3ed-5506-00db-b73f-00d0
Value: 1.9436057, Gold: 99.671

c05d-5dd4-6071-ac0f-e626-4e0c
Value: 6.0317187, Gold: 309.31552

9032-7d6c-7dfe-8a36-1d95-292d
Value: 8.532134, Gold: 437.54056

d012-e922-731f-0ef6-7faf-5fa4
Value: 7.5822845, Gold: 244.69403

a04b-b58f-4914-5d7f-dc41-10b8
Value: 6.0053926, Gold: 193.80489

e4b6-57f4-ba28-a2c3-db7a-bcf0
Value: 6.780262, Gold: 218.81133

43d8-2776-2282-6b2e-e27f-1c59
Value: 7.843456, Gold: 373.74286

cfb2-96c9-d5b9-8ad0-eeda-d96f
Value: 4.715455, Gold: 224.69275

c837-8d18-e184-d2ec-e0a6-2d1a
Value: 4.6936507, Gold: 223.65378

551e-46b6-31ab-b80f-3c7d-a9d9
Value: 1.9200956, Gold: 91.49309

f81d-f9a4-bc70-3806-f906-fa92
Value: 5.8826947, Gold: 280.31204

fc85-d145-77c6-5d45-1111-b630
Value: 5.936732, Gold: 101.828186

4f57-5a31-43b0-e37d-ddc9-e410
Value: 9.915226, Gold: 170.06824

68e9-a5d6-3ae2-f389-97ef-5f85
Value: 0.14611046, Gold: 2.5061202

ee88-02dc-71dc-e03b-1603-66c6
Value: 1.545063, Gold: 26.501276

9111-2bf9-7064-0894-a06b-596d
Value: 8.357703, Gold: 143.35324

How much Gold do you want?

CANCEL PLACE OFFER

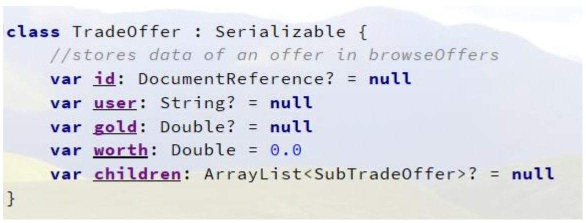
Coin items you want your offer to include, screenshots/scroll-screenshots taken from various phones

Receiving coin items (BrowseOffers.kt, FragmentDepot.kt)

Essential function calls in chronological order:

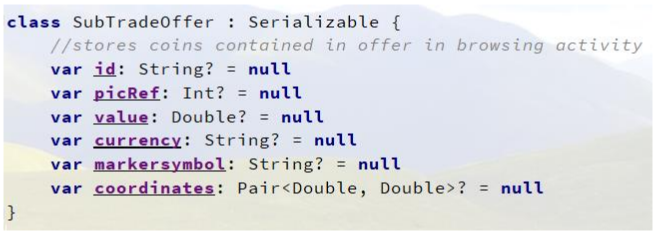
OnCreate() ➔ sets up RecyclerView, calls loadOffers(), adds SnapshotListener
loadOffers() ➔ sets up FirestoreRecyclerAdapter ➔ populates ViewHolders
executeOrder66() ➔ removes offer from Firestore, manages coin items for gold exchange

OnCreate() feeds the RecyclerView in the activity with live-updates on the offers through a SnapshotListener. It also calls loadOffers() which initializes the FirestoreRecyclerAdapter as an object that is set up with the Listener and RecyclerView. It makes use of three classes named TradeOffer (Fig. 1.3) (holding offeror name, gold price, and SubTradeOffers), a List of SubTradeOffers (Fig. 1.4) (holding coin items offered) and a ViewHolder named TradeOfferHolder. When an offer in the RecyclerView is accepted executeOrder66() is called which removes the accepted offer from the Firestore, reduces the acceptors gold and puts the coin items contained in said offer into the user's wallet. In OnBindViewHolder() of the adapter we filter out the offers made by the acceptor or by users that do not match the predefined username filter by setting their visibility to GONE and their layout height to 0. Coin items received from an offer are marked with the syllable "TRADED" in their ID so that they can be exchanged without reducing the daily exchange capacity. FragmentDepot uses a SnapshotListener to listen to a feature called "plus" in the user's document in Firestore. If its value changes it is added to the gold of the user and a toast is printed to inform the user either while he/she is online or on reopening the app.



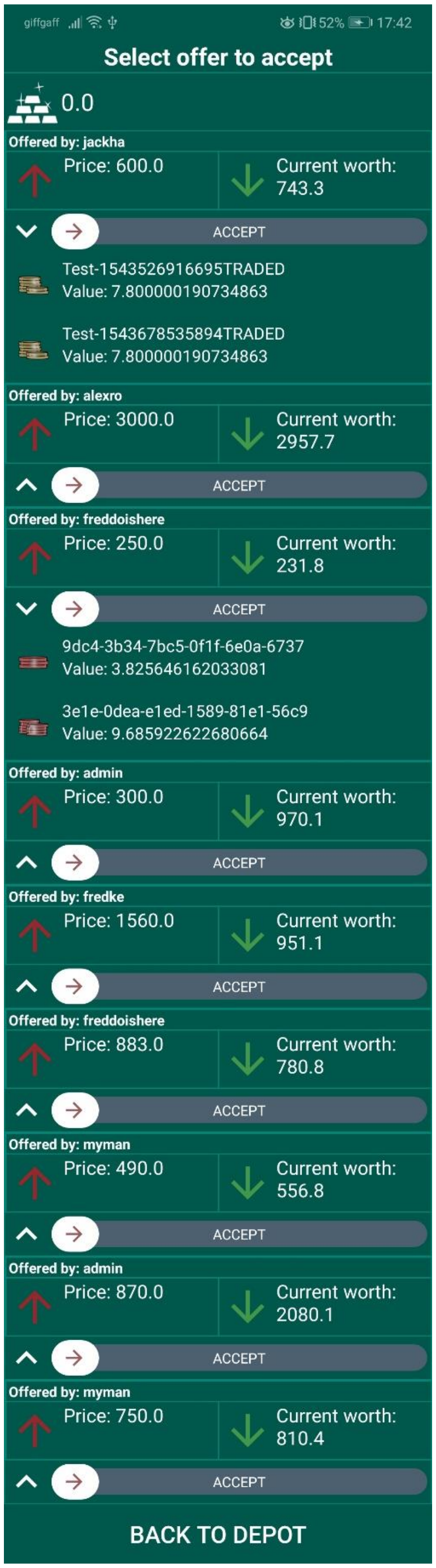
```
class TradeOffer : Serializable {
    //stores data of an offer in browseOffers
    var id: DocumentReference? = null
    var user: String? = null
    var gold: Double? = null
    var worth: Double = 0.0
    var children: ArrayList<SubTradeOffer>? = null
}
```

Fig. 1.3



```
class SubTradeOffer : Serializable {
    //stores coins contained in offer in browsing activity
    var id: String? = null
    var picRef: Int? = null
    var value: Double? = null
    var currency: String? = null
    var markersymbol: String? = null
    var coordinates: Pair<Double, Double>? = null
}
```

Fig. 1.4


giffgaff 52% 17:42

Select offer to accept

0.0

Offered by: jackha
Price: 600.0 Current worth: 743.3
ACCEPT

Test-1543526916695TRADED
Value: 7.800000190734863

Test-1543678535894TRADED
Value: 7.800000190734863

Offered by: alexro
Price: 3000.0 Current worth: 2957.7
ACCEPT

Offered by: freddoishere
Price: 250.0 Current worth: 231.8
ACCEPT

9dc4-3b34-7bc5-0f1f-6e0a-6737
Value: 3.825646162033081

3e1e-0dea-e1ed-1589-81e1-56c9
Value: 9.685922622680664

Offered by: admin
Price: 300.0 Current worth: 970.1
ACCEPT

Offered by: fredke
Price: 1560.0 Current worth: 951.1
ACCEPT

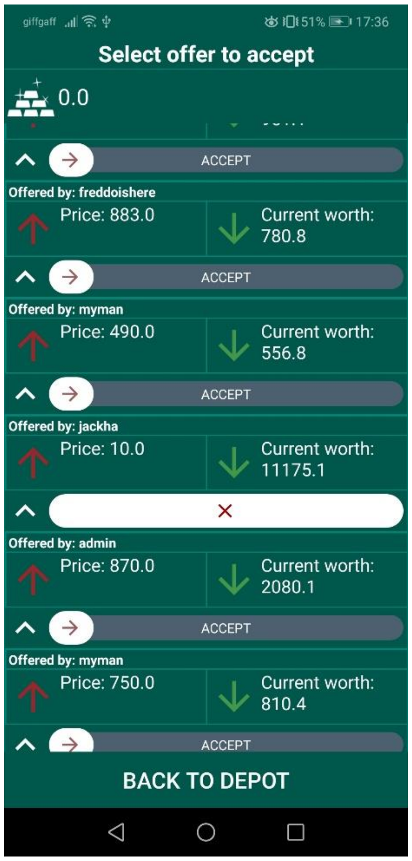
Offered by: freddoishere
Price: 883.0 Current worth: 780.8
ACCEPT

Offered by: myman
Price: 490.0 Current worth: 556.8
ACCEPT

Offered by: admin
Price: 870.0 Current worth: 2080.1
ACCEPT

Offered by: myman
Price: 750.0 Current worth: 810.4
ACCEPT

BACK TO DEPOT


giffgaff 51% 17:36

Select offer to accept

0.0

ACCEPT

Offered by: freddoishere
Price: 883.0 Current worth: 780.8
ACCEPT

Offered by: myman
Price: 490.0 Current worth: 556.8
ACCEPT

Offered by: jackha
Price: 10.0 Current worth: 11175.1

Offered by: admin
Price: 870.0 Current worth: 2080.1
ACCEPT

Offered by: myman
Price: 750.0 Current worth: 810.4
ACCEPT

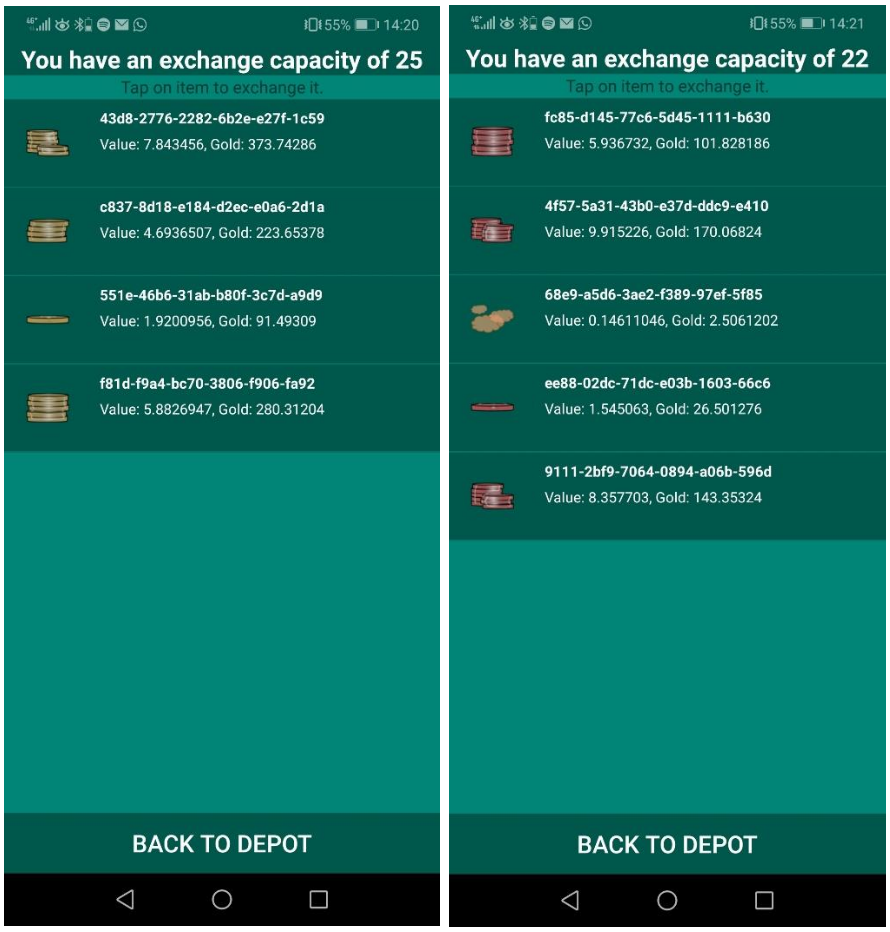
BACK TO DEPOT

If you do not have enough gold your request to accept the offer will be denied, offers can be expanded to see what coin items are included in the offer, screenshots/scroll-screenshots taken from various phones

Exchanging coin items for gold/Paying them into the bank (ExchangePopUp.kt)

OnCreate() ➔ sets up adapter for ListView
onItemClick() ➔ exchanges the coin item that was clicked on

This works similarly to sending coin items off for offer placings. Again OnCreate() sets up the ListView with an adapter and an ArrayList of HashMaps. Here the HashMaps are all from one kind of currency which is being determined based on an intent-extra from the activity that opened this one. When an item is clicked it is transferred to gold and the respective coin item is removed from the wallet. If the coin item was collected from the map the daily exchange capacity is reduced by one (starting at 25).



You have an exchange capacity of 25
Tap on item to exchange it.

43d8-2776-2282-6b2e-e27f-1c59
Value: 7.843456, Gold: 373.74286

c837-8d18-e184-d2ec-e0a6-2d1a
Value: 4.6936507, Gold: 223.65378

551e-46b6-31ab-b80f-3c7d-a9d9
Value: 1.9200956, Gold: 91.49309

f81d-f9a4-bc70-3806-f906-fa92
Value: 5.8826947, Gold: 280.31204

BACK TO DEPOT

You have an exchange capacity of 22
Tap on item to exchange it.

fc85-d145-77c6-5d45-1111-b630
Value: 5.936732, Gold: 101.828186

4f57-5a31-43b0-e37d-ddc9-e410
Value: 9.915226, Gold: 170.06824

68e9-a5d6-3ae2-f389-97ef-5f85
Value: 0.14611046, Gold: 2.5061202

ee88-02dc-71dc-e03b-1603-66c6
Value: 1.545063, Gold: 26.501276

9111-2bf9-7064-0894-a06b-596d
Value: 8.357703, Gold: 143.35324

BACK TO DEPOT

Users tap on an item and will disappear from the list and get exchanged, the ListViews can also be scrolled, screenshots taken from various phones

Elements that have not made it into the final application

The following features were mentioned as potential bonus features in the previous report from coursework 1 and were not implemented in the final version of the application:
Regard the fact that the previous report mentioned about nine potential bonus features. This was criticized and resulted in a mark reduction. For reasons of completeness all of those which were not implemented are listed here.

- Sound effects/haptic feedback/music
- Coin item collection against the clock
- Displaying the location of your friends and friend requests
- Scotland Yard event
- “Cargo”-transportation event
- Relay race event

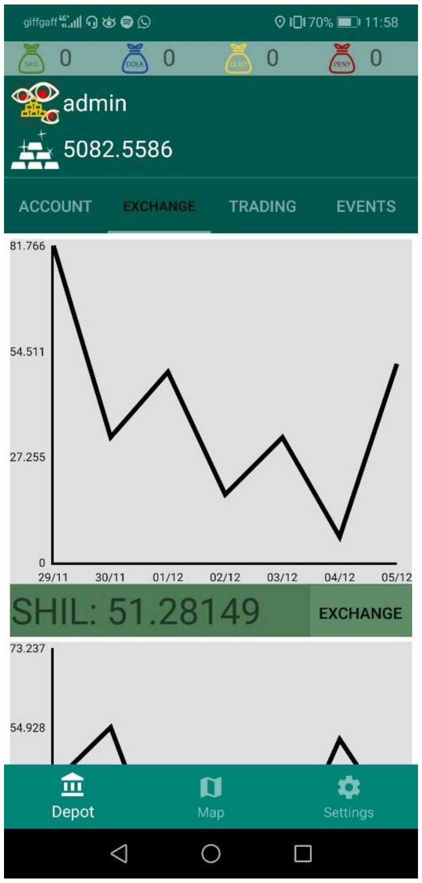
Although it was said that these features were only under consideration, I will explain why they were quickly dropped.
A basic level of sound effects and haptic feedback is already provided by the UI elements of Android Studio. Thus, there was little need for that. Music turned out to be not suitable for the app and rather annoying. Coin item collection against the clock was not directly implemented but found its way into the application through the “Catch me if you can”- event which also features a timer (see **New elements that have made it into the final application**) but is more interesting than coin item collection against the clock. The idea of friend requests and their location display was originally bound to the Scotland Yard event and the relay race event so that people could find a way to find other players to play the event with. Both ideas were dropped together since one did not make much sense without the other. The Scotland Yard event turned out to be too extensive and would have undermined the basic game principles. It would have turned the game into something completely different. The same was true for the “Cargo”-transportation event. These events would have also required multiple people to test it which is impractical.

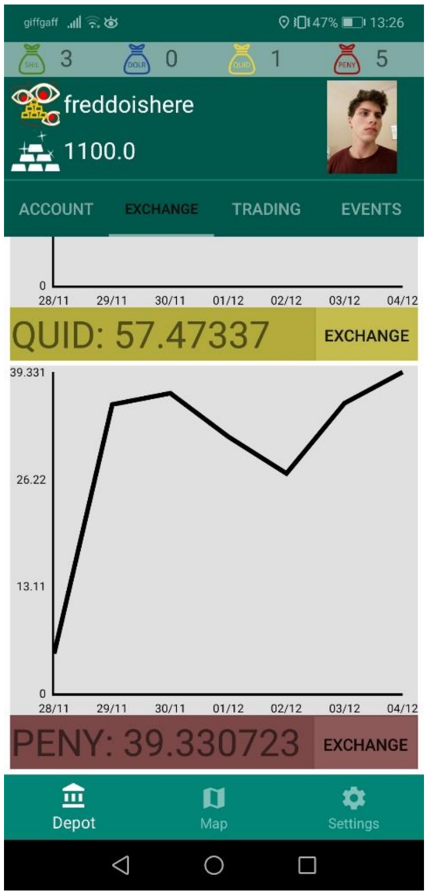
New elements that have made it into the final application

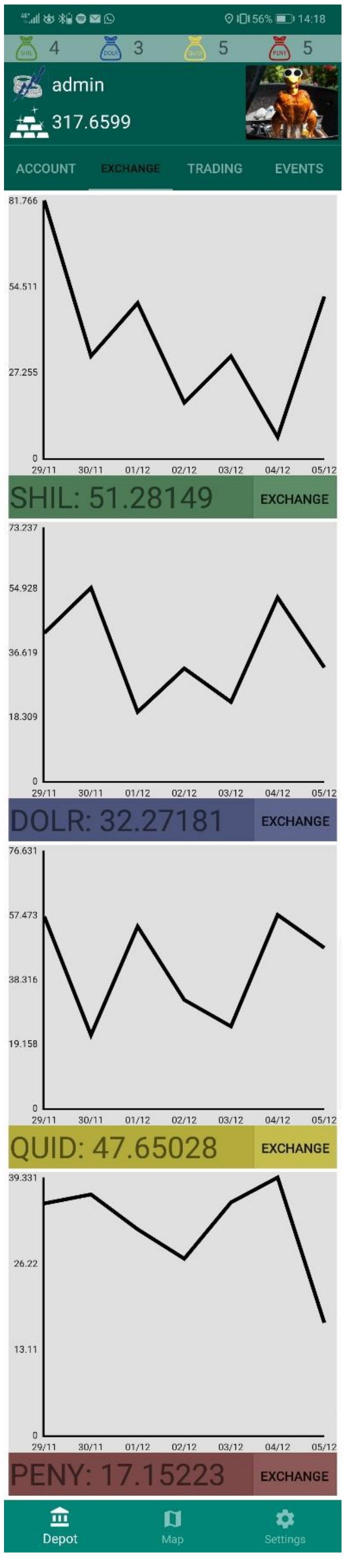
The following features were not mentioned as potential bonus features in the previous report from coursework 1 but were implemented in the final version of the application:

- Graphic representation of the past coin item exchange rates (additional design feature)
- “Catch me if you can”- event (Bonus feature)
- Zone purchases for the user’s team (Bonus feature)

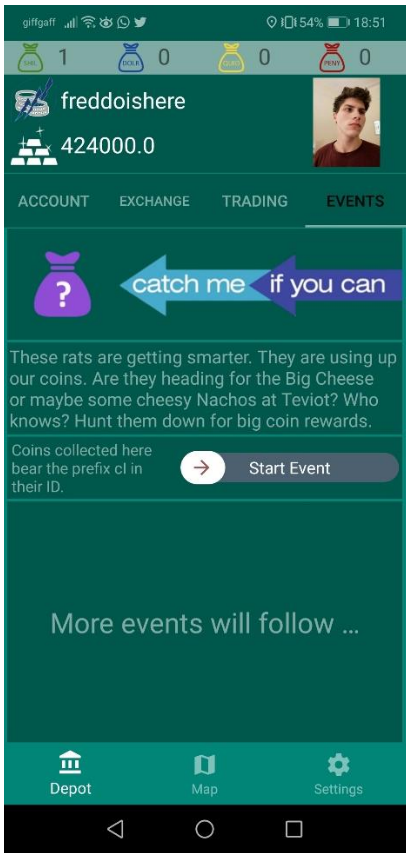
One can see the exchange rates of every currency of the past week in the “Exchange”-tab. The rates are plotted through a line-plots respectively. This helps the users in their decision to either keep coin items in their wallet or exchange them for gold. Coin items can be held in the wallet indefinitely. The “Catch me if you can”- event refers to a famous movie. The user must catch a sac filled with a random amount of coin items. The sac is carried by rats and moves around the campus for a couple of minutes. When the timer hits zero seconds the user has failed to gather any reward. The event can be played once a day. Zone purchases are based on the concept of forming teams which was introduced in the previous report. Players can purchase zones for their respective teams. Players from another team will not be able to collect any coin items in a zone controlled by another team. Players can only gather coin items in one of the zones owned by their team. There are 25 zones distributed over the campus. The zones are being purchased with gold. The more a player invests into a zone the more gold must be invested to purchase it back for another team by other players. Players must be smart about whether it is worth investing a certain amount of gold into zone.





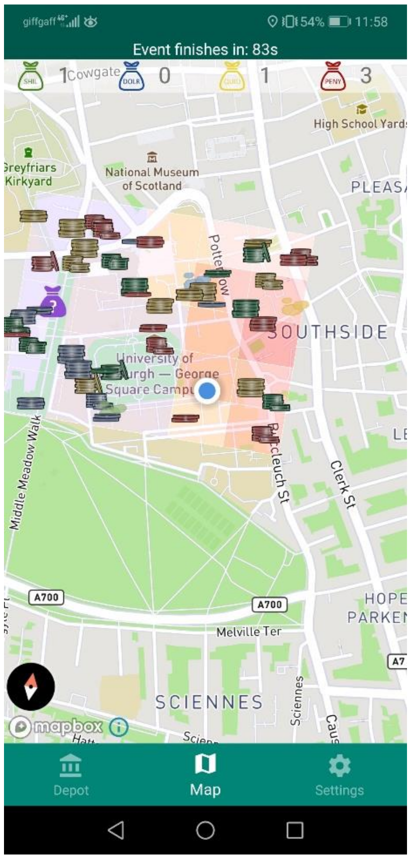


Users can view the past exchange rates and make their decision on whether to exchange today based on that, screenshots/scroll-screenshots taken from various phones

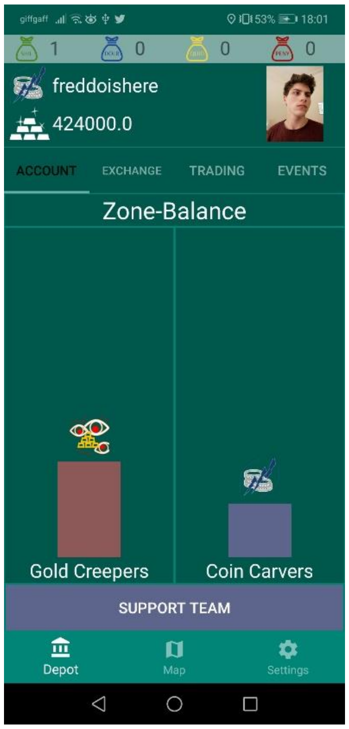


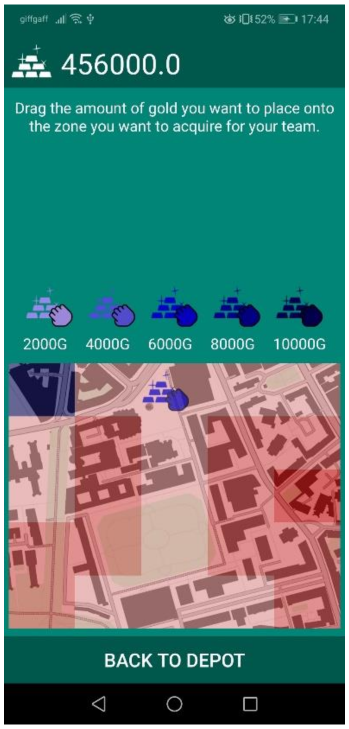
These rats are getting smarter. They are using up our coins. Are they heading for the Big Cheese or maybe some cheesy Nachos at Teviot? Who knows? Hunt them down for big coin rewards.

Coins collected here bear the prefix cl in their ID.

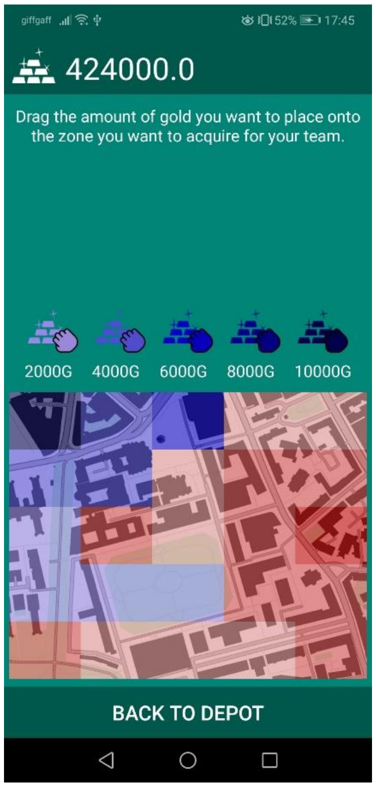


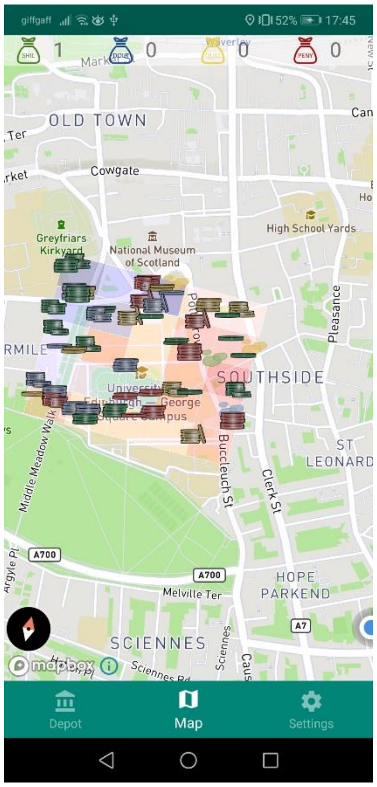
The sac is added by swiping the button, the app goes to the map and the user must catch the moving sac against time, screenshots taken from various phones





Drag the amount of gold you want to place onto the zone you want to acquire for your team.


424000.0
Drag the amount of gold you want to place onto the zone you want to acquire for your team.
2000G
4000G
6000G
8000G
10000G
BACK TO DEPOT



Users click the “support team”- button, in the started activity the user drags the desired amount of gold onto a zone (some UI colors depend on the user’s chosen team), screenshots taken from various phones

Bonus features and their structure

Additional design features: Graph plots of past exchange rates
Profile picture upload

Additional game features/Bonus features: Teams and zone purchases
“Catch me if you can”- event

Graph plots of past exchange rates (SubFragmentExchange.kt)

The download and parsing of the exchange rates is explained under “**Download and parsing of the map**” in **Algorithms and data structures**.
DownloadExchangeRates() ➔ callback SubFragmentExchange ➔ initGraphs()

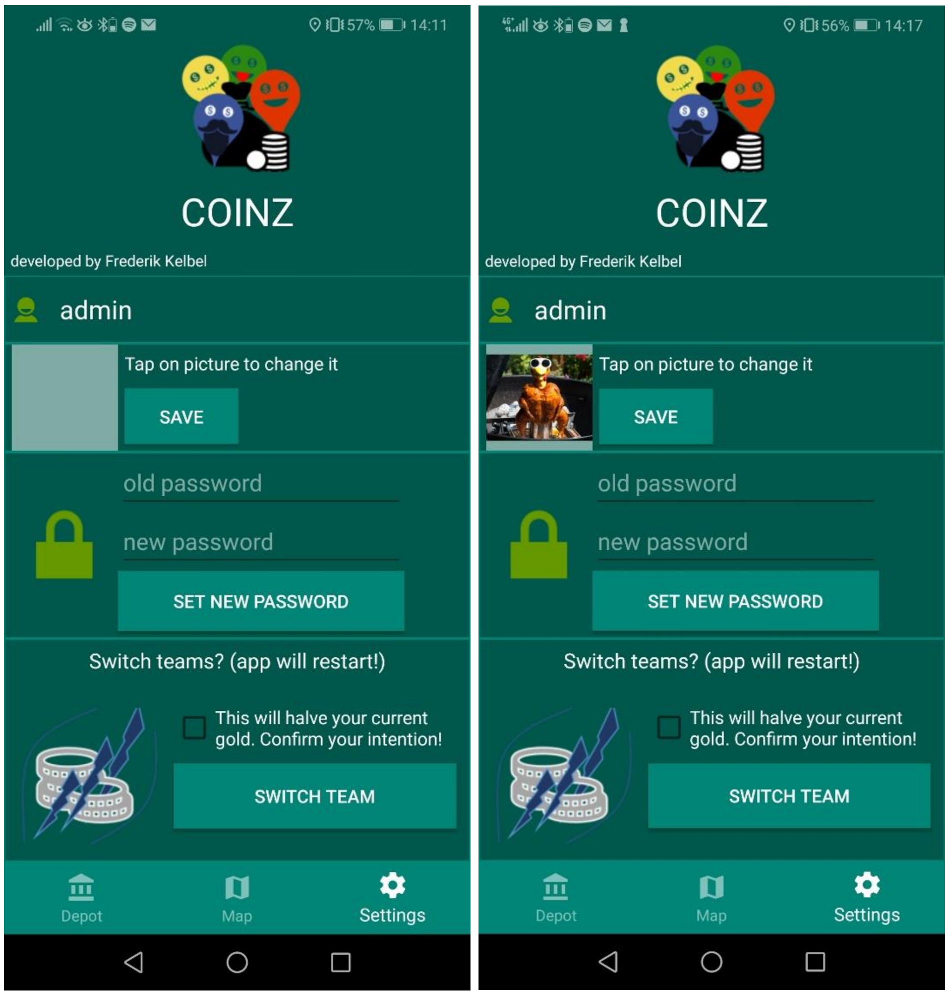
The graphs are plotted with 'com.diogobernardino:williamchart:2.5.0' (see **Acknowledgements**). As soon as SubFragmentExchange is created DownloadExchangeRates() is executed in the background. As soon as it finishes the function initGraphs() is called in SubFragmentExchange. InitGraphs() makes use of the coinExchangeRates-array in the companion object of Profileactivity which also contain the respective dates. The rest follows the basic instructions of 'com.diogobernardino:williamchart:2.5.0' of feeding line plots the necessary data. If DownloadExchangeRates() does not finish the graphs are blank and it will say beneath that the exchange rates are not retrievable.

Profile picture upload (FragmentDepot.kt, FragmentSettings.kt)

Upload
onClick() ➔ calls choosePic()
choosePic() ➔ formulates intent to get image from storage, calls startActivityForResult()
startActivityForResult() ➔ executes intent, calls onActivityResult()
onActivityResult() ➔ sets source for ImageView in settings
onClick() ➔ calls uploadImageToFirebase()
uploadImageToFirebase() ➔ uploads image to Firbase storage, sets photo URI in firebase
Retrieving profile picture
OnViewCreated() ➔ calls loadInformation()
loadInformation() ➔ loads picture

Aside of minor color and logo changes depending on your team of choice uploading a profile picture is another way to customize your profile. Clicking on the picture

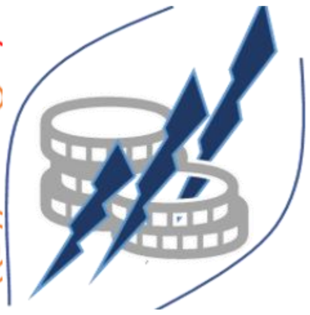
placeholder in FragmentSettings.kt results in choosePic() being called. The function creates an intent with ACTION_GET_CONTENT of type "image/*" and calls startActivityForResult(). Following that the user selects a picture from his/her gallery which leads to onActivityResult() being called. It checks whether resultCode and requestCode are correct and puts the image in place of the placeholder. Clicking on the save button then results in uploadImageToFirebase() being called. It uploads the image under a folder named profilepics in the Firebase storage and sets the user specific phot URI so that it can be retrieved again later.
To load an image on opening of the app we use the bumptech library ('com.github.bumptech.glide:glide:4.3.1', see **Acknowledgements**) and its function Glide.with().load().into(). This is what loadInformation() does.


57% 14:11
COINZ
developed by Frederik Kelbel
admin
Tap on picture to change it
SAVE
old password
new password
SET NEW PASSWORD
Switch teams? (app will restart!)
This will halve your current gold. Confirm your intention!
SWITCH TEAM
Depot
Map
Settings
56% 14:17
COINZ
developed by Frederik Kelbel
admin
Tap on picture to change it
SAVE
old password
new password
SET NEW PASSWORD
Switch teams? (app will restart!)
This will halve your current gold. Confirm your intention!
SWITCH TEAM
Depot
Map
Settings
The user taps on the picture to change it, to save it to Firebase the user clicks the save-button, screenshots taken from various phones

Teams and zone purchases (FragmentDepot.kt, SubFragmentAccount.kt, FragmentSettings.kt, TeamZone.kt, SignUpActivity.kt, FragmentMap.kt)

FragmentDepot.kt displays the team logo next to your user name depending on the variable team in the companion object of ProfileActivity.
FragmentSettings.kt does the same and changes the variable in the companion object of ProfileActivity respectively to your intention to switch teams.

Goldcreeper team


Coincarver team

Signing up for a team (SignUpActivity.kt)

SignActivity makes use of a custom class named CustomDropDownAdapter which implements BaseAdapter. It takes an array of pictures and an array of titles and makes up the adapter for the spinner used at sign up to select your team.
FragmentMap.kt sets up a SnapshotListener on a document collection called zones in Firestore. Each document is named with only a number ranging from 0 to 24 representing each zone on campus. Each document has only one attribute named c for color which captures an integer referencing one of the 10 colors (there are different levels of ownership for each team)that a zone can take on. Its function drawPolygons is called when ProfileActivity receives the callback from DownloadFileTask() or in the startup phase of FragmentMap if the app had been opened before in the day. The polygons are added with map.addPolygon() and the Listener is set up and changes the colors based on the document changes. checkForCoin() then checks for every coin item that is near enough if it is a valid zone you want to collect your coin item in. The zones make up a grid over the campus. Thus, it compares the boundaries of the polygons with the location the potential coin item collection is taking place.
SubFragmentAccount.kt displays the amounts of zones every team owns through a bar plot made up of two simple LinearLAyouts that vary in height depending on the respective amounts. The SnapshotListener in FragmentMap calls a function called updateBars() in SubFragmentAccount to update the height of the LinearLayouts according to the count of blue or red zones. The button color of the button linking to TeamZone also varies depending on the value of the team variable in the companion object in ProfileActivity.

Changing the balance on campus (TeamZone.kt)

onCreate() ➔ sets up drag objects, calls initViews(), loads array of zone colors from Firestore, sets up SnapshotListener
initViews() ➔ sets up fields to drag onto/into
onDrag() ➔ calls reduceGold(), calls getFittingColor()

TeamZone makes use of two custom classes. One is named ChoiceTouchListener, implements View.OnTouchListener and listens for potential drags on the draggable objects. The other one is called ChoiceDragListener, implements View.OnDragListener and listens for drops into the fields.The ChoiceDragListeners are set up for each of the 25 ImageViews representing the fields/zones in initViews(). The ChoiceTouchListeners are set up on the draggable objects in onCreate(). Each draggable object associates to a gold value that can be dropped on the zones to induce a certain color level in the zone. If the event DragEvent.ACTION_DROP is noticed the listener calls reduceGold() which checks whether the user has enough gold and reduces t appropriately afterwards. If the user has enough gold for the drop the function return true and getFittingColor() is called. Draggable objects of one team a tagged with numbers from -1 to -5 and draggable objects of the other team are tagged with numbers from 1 to 5. These tags are added to the index that the previous color had in an array of all the color levels. This yields a new index that can be used to get the new color and update the integer referencing the specific color in Firestore. The SnapshotListener works in the same way the SnapshotListener in FragmentMap works.

"Catch me if you can"- event (SubFragmentEvent.kt, FragmentMap.kt, ProfileActivity.kt)

onViewCreated(), generatek2() ➔ calls addMovingSac(), generates track of sac
addMovingSac() ➔ adds sac as marker, initiates ObjectAnimator, initiates CountDownTimer, calls updateMovingSac()
updateMovingSac() ➔ updates destinations of Animator to fit sac movement to generated track
catchMovingSac() ➔ generates coin items that where in the sac

A SwipeButton from 'com.ebanx:swipe-button:0.8.3' (see **Acknowledgements**) starts the event.

A SwipeButton from 'com.ebanx:swipe-button:0.8.3' (see **Acknowledgements**) starts the event. If the static variable eventAvailability is true, it starts the event. If it is false, the button signalizes that the request to start the event was denied through a red cross on the button. eventAvailabiltiy is true once a day. The sac moves at a constant speed that is comparable to fast walking. generatek2() is called iteratively to generate checkpoints that the sac is moving along until a duration of roughly 150000 ms to complete the track is reached. Checkpoints are generated near the last checkpoint

(Lat/Lng differ by max 0.0029) except if the generated checkpoint would lead to the sac leaving the given map boundaries. In that case the next checkpoint is generated at a random position within the boundaries. A list of the checkpoints and durations to reach the respective next checkpoint is then passed to addMovingSac() and FragmentMap is put to the foreground. addMovingSac() starts an ObjectAnimator with the respective checkpoint and the duration to reach it. It then calls updateMovingSac() which does the same but calls itself recursively via onAnimationEnd() until the sac is either caught or reaches the last checkpoint. Thus, every time an animation ends a new one is started to vary the animation track. To check whether the animator reached the next checkpoint or got caught the ObjectAnimator makes use of a custom class named LatLngEvaluator which is a TypeEvaluator used to interpolate between checkpoints. It also checks whether the last location of the user is near the current location of the sac. If it is it calls the static method catchMovingSac() in ProfileActivity. It generates up to 11 coin items and adds them to the user's wallet. The generated coin item is totally random and marked with the prefix
"cI-" in their ID. When the sac reaches its last checkpoint a CountDownTimer initiated in addMovingSac() hits 0 and the marker is removed from the map.

Acknowledgements

The following features were implemented with help of open-source libraries (not included are necessary libraries):

`org.jetbrains.kotlin:kotlin-stdlib-jdk7`: Use of Kotlin
`com.android.support:appcompat-v7`: Support library
`com.android.support.constraint:constraint-layout`: Support library
`com.android.support:design`: Support library
`junit:junit`: Unit tests
`com.android.support.test:rules`: Instrumented tests
`com.android.support.test:runner`: Instrumented tests
`com.android.support.test.espresso:espresso-core`: Instrumented tests
`com.mapbox.mapboxsdk:mapbox-android-sdk`: Map provider
`com.google.firebase:firebase-core`: Cloud provider
`com.google.firebase:firebase-auth`: User authentication
`com.google.firebase:firebase-storage`: Storing of profile pictures
`com.github.bumptech.glide:glide`: Loading profile pictures
`com.github.bumptech.glide:compiler`: Annotation processing
`com.google.android.gms:play-services-location`: Location provider
`com.mapbox.mapboxsdk:mapbox-android-plugin-locationlayer`: Map provider
`org.apache.directory.studio:org.apache.commons.io`: Inputstreams etc.
`com.google.code.gson:gson`: Converting objects to strings for Firestore
`com.diogobernardino:williamchart`: Exchange rate plotting
`com.google.firebase:firebase-firestore`: User database
`android.arch.core:runtime`: Android architecture components
`android.arch.core:common`: Android architecture components
`com.firebaseui:firebase-ui-firestore`: FirestoreRecyclerAdapter
`com.ebanx:swipe-button`: Swipe Buttons

The following tutorial helped me in setting up the FireStoreRecyclerAdapter for the RecyclerView in BrowseOffers.kt:

`grokonez.com/android/kotlin-firestore-example-crud-operations-with-firebaseui-firestorerecycleradapter-android`