Rates Android test application – Aleksei Kalachev

Tasks

9th February 2020
According to the requirements here are the milestones that are needed to accomplish.

RVLT100. Design:
RVLT110. Basic layout that includes a toolbar and a list with a title, a subtitle and a currency rate. The currency rate can have maximum two digits after the comma.
RVLT120. Make the list item clickable and place it on top. It should remain there while the list is scrolled.
RVLT130. Make the currency rate field clickable. Only the top field is clickable. Only numbers and a separator are allowed.
RVLT140. When the item is edited and contains nothing, show a placeholder containing zero and coloured with gray.
RVLT150. (optional) Hide the toolbar when scrolling the list down and show it when scrolling up.
RVLT160. Find a proper source for the flags. The best is to get it from some API so that adding a new currency wouldn't require a new version of the app to compile.

RVLT200. Business logic.
RVLT211. Call the API to show rates
RVLT212. Call the API once per second.
RVLT220. Multiply the currency rates with the amount that is typed in the edited item.
RVLT230. Customise the API call. The base parameter should be taken from the selected item. The selected item should be persisted in the app and shown when the app is opened again.
RVLT240. (optional) Handle the offline mode.
RVLT241. (optional) If the internet is unavailable – show the outdated results with a snackbar informing the user that they are now working with old data.
RVLT242. (optional) If the data is unavailable – keep the selected item and show a snackbar informing the user that the data is unavailable.
RVLT243. (optional) It the database is completely empty – show an empty list with a snackbar informing the user that they need to establish a network connection first in order to work with the app.
RVLT250. (optional) Handle the online mode. When the device is back online, hide the snackbar and update the data.
RVLT260. (optional) Copy the value to the buffer when the item is long pressed.

RVLT300. Maintenance.
RVLT310. Verify that the necessary comments are written and unnecessary ones are deleted. All TODOs are implemented, no IDE placeholders are left unhandled.
RVLT320: Verify that the necessary tests are written and passed.
RVLT330: Verify that the UI handles large numbers properly. Introduce limits based on the results and common sense.
RVLT340. Verify that unnecessary files are removed, imports are cleaned, unused methods deleted.
RVLT350. Verify that the project description is ready.

10th February 2020
RVLT270. Find an external service that provides a proper name of the currency. Call it with the rates API.

11th February 2020
RVLT280. Fix crash when force clicking the list

13th February 2020
RVLT170. In the offline mode indicate that the item is disabled.

14th February 2020
RVLT270. Stop updating when going to background. Resume updating when going to foreground.

Project description.

Libraries

The app uses Realm database as a database engine to operate with the currencies and currency rates. Realm was chosen as a reliable engine without a need to write SQL commands and keep the development in the scope of Kotlin programming.

For the inspection of the content of the app we use Stetho library. When the app is launched in the debug configuration one can inspect the app via Google Chrome or the latest Microsoft Edge by typing chrome://inspect or edge:// inspect.

The app uses Dagger2 as a dependency injector.
It helps to build and provide
- the Realm database (AppModule),
- the NetworkService (NetworkServicesModule),
- the Repositories for currencies and rates (RepositoryModule)
- the presenter for the list of currency rates (PresenterModule)
- the adapter for the RecyclerView in the RatesFragment.

The app uses Picasso as an image processing engine. For transforming the images to be of a circle shape a CircleTransform class is used.

For handling the http request the Retrofit library is used.
For background work the RxJava2 is used.

Reused utilities

The following utilities were written outside of the challenge and were reused here to save time.
L - a utility for handling the Log methods.
InternetConnectivityWatcher - a utility to track if the app has gone offline/online.
CircleTransform - to reshape the images into the form of a circle.

Architecture and structure

The app is based on an MVP architecture. MVP was chosen because it's used in the commercial project so it made it easier and quicker to switch between them two.

The app is a single activity app (MainActivity) with a fragment (RatesFragment) that represents a list of currency rates. The business logic is handled by the presenter (RatesPresenterImpl) and the list handling is done in the adapter (RatesAdapter).

The base classes and interfaces (BaseActivity, BaseFragment, BasePresenter) are made for simplifying further extension of the app if needed.

Business logic: online

In order to make the app flexible it was decided not to hardcode the flags and currency names in the app and use external sources. In the real world app this can make it easier to localise and add/change/remove the results without having a need to upload a new APK/Appbundle and wait for the users to update their devices.

The app uses three external sources to provide the info to a user:
- the currency rates endpoint (https://revolut.duckdns.org/)
- the currency endpoint (https://openexchangerates.org/api/currencies.json)
- the flags endpoint (https://raw.githubusercontent.com/hjnilsson/country-flags/master/png100px/)

The flags are handled by Picasso library and are shown when the app is online. If the flags are unavailable the app will continue to work normally.
The currency endpoint provides a list of titles for each existing currency, so it is mandatory to get that list first. Once the list of currencies is got, it is stored in the database and no other requests to that endpoint are made.

Then the rates are got once per second and stored in the database by pair <base-currency, target currency>. Once the same pair is fetched from the api, the rate and the date got updated. This is done in a purpose of offline mode handling which will be covered later.

When a user clicks on any item, it moves to the top of the list and the amount that was in the input field becomes the base amount to multiply the other rates with.
Upon this swap procedure the list is scrolled to the top. To speed this procedure up, an instant scroll to the top can be performed depending on how close to the bottom of the list the user was standing.

When a user long clicks on any item – the amount of that item is copied to their clipboard and a snackbar is shown.

When a user changes the input on the top (base) currency item, the other items handle this change accordingly. The format of the input is decimal with a maximum of 10 symbols including the separator with a maximum of 2 digits after the separator. This is done in terms of the UI to look nice on smaller screens.

Business logic: offline

If the currency rate changes once per second then the offline mode would be useless because a user might not want to see outdated result but it was decided to add this mode since all the data that is needed is already in the database.

If a user is offline, they see a constant snackbar indicating that the app is offline. Once the app gets connection, the snackbar disappears and the app will try to get the necessary data again (either currencies or rates or both).

Also it may be the case that a user has previously switched to some other currencies so their rates are stored locally. In this case when the app goes offline – we get the list of all the currencies which can be used offline and update the list accordingly. So the user is free to switch between “offline” currencies and is informed which currencies are unavailable to switch to.

Also when a user long clicks the item - the “clipboard” message is shown instead of the “offline” message. However afterwards the “offline” message is back so that the user is still informed they are offline.

Once the app is back to online the list is coloured back to normal and the “offline” message is gone.

Usability issues
During the development a few usability issues were found but not fixed due to the lack of time.
1. When scrolling to top and swapping the times some frames are dropped on slow devices.
2. In offline more it’s unable to long click the disabled devices.