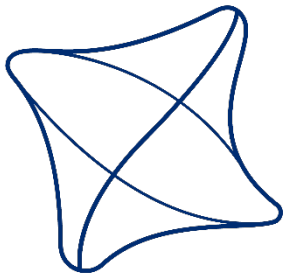
Žilinská univerzita v Žiline
Fakulta riadenia a informatiky



Semestrálna práca
Vývoj aplikácií pre mobilné zariadenia

Cryptolit

Android aplikácia

Dominik Ježík

2023

1 Popis a analýza riešeného problému

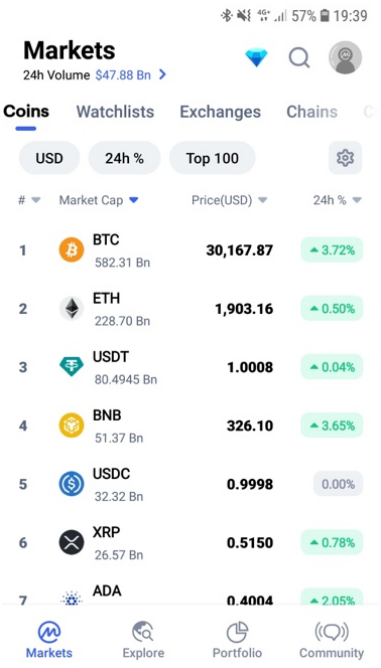
1.1 Špecifikácia zadania

Našou úlohou je vytvoriť Android aplikáciu v prostredí Android Studio použitím jazyka Kotlin, ktorá poskytne prehľad o dostupných kryptomenách, aktuálnu hodnotu každej kryptomeny, vývoj ceny v čase a poskytne kalkulačku na výpočet hodnoty jednej kryptomeny vyjadrenú v jednotkách inej kryptomeny. Aplikácia bude postavená na trojvrstvovej architektúre, vďaka čomu bude jednoducho rozšíriteľná aj do budúcnosti. Závislosti medzi triedami v jednotlivých vrstvách bude spravovať knižnica Hilt, ktorá poskytne Dependency Injection. Dáta o menách získame z verejne dostupného API – Coin Gecko API, ktoré na naše účely poskytuje prístup k dátam zadarmo s istými obmedzeniami (10-30 požiadaviek za minútu na zariadenie). Používateľ bude mať možnosť uložiť si vybrané kryptomeny do zoznamu obľúbených mien alebo do zoznamu sledovaných mien. Preto aplikácia bude využívať aj lokálnu databázu a knižnicu Room, pre jednoduchý prístup k dátam.

1.2 Rozbor podobných aplikácií

Podobných aplikácií v oblasti kryptomien je mnoho, keďže je momentálne rastúci trend v tejto oblasti a väčšia z nich zdieľa podobnú funkcionalitu, preto sú v tejto časti vybraté dve aplikácie, ktoré majú podobnú funkcionalitu ako naša aplikácia.

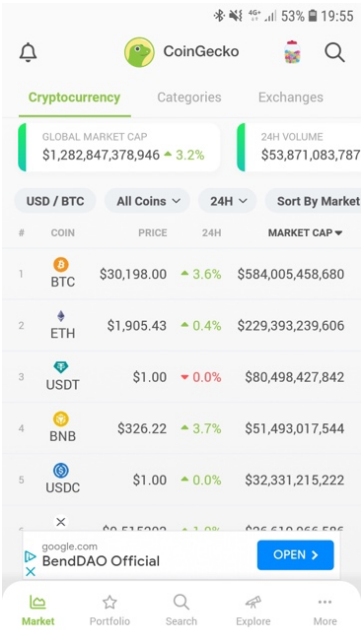
1.2.1 CoinMarketCap



CoinMarketCap je aplikácia, ktorá poskytuje komplexný prehľad o kryptomenách – zoznamy kryptomien, krypto-búrz, novinky o kryptomenách. Umožňuje vytvoriť watchlist kryptomien, o ktoré má používateľ záujem. Ponúka pokročilé možnosti filtrovania kryptomien. Tiež tu nájdeme konvertor podľa aktuálneho kurzu, ktorý slúži na výpočet čiastky v jednej mene na inú menu. Používateľské rozhranie je zamerané pre pokročilých používateľov a nový používateľ sa dokáže v aplikácií stratiť kvôli množstvu tlačidiel a čísel, ktoré zaujmú hlavne náročných používateľov.

Odlišnosti: Naša aplikácia je podstatne jednoduchšia na používanie pre začiatočníkov – jednoduchšie používateľské rozhranie, základné funkcie. Dizajn CoinMarketCap-u je veľmi jednoduchý, naša aplikácia bude mať futuristický dizajn, ktorý napriek tomu ostane minimalistický.

1.2.2 CoinGecko



CoinGecko je veľmi podobná aplikácia ako CoinMarketCap, ale v zozname všetkých kryptomien sa zobrazuje reklama, čo nie je veľmi príjemne z hľadiska UX (User Experience). Ďalšia nepríjemná skutočnosť je, že si používateľ nedokáže pridať kryptomeny do zoznamu obľúbených kryptomien pre rýchly prístup pokiaľ sa do aplikácie neprihlási. CoinGecko ponúka tiež konvertor kryptomien, pričom zadanú hodnotu automaticky prepočítava do zoznamu pevne daných kryptomien.

Odlišnosti: Naša aplikácia využíva priamo API od CoinGecko, ale neotravuje používateľa, neustálymi žiadosťami o prihlásenie a obmedzením základných funkcií pokiaľ sa používateľ neprihlási. Tiež nebude otravovať reklamami na hlavnej obrazovke.

2 Návrh riešenia problému

2.1 Prípady použitia

Používateľ aplikácie dokáže

- zobraziť zoznam všetkých kryptomien spolu s ich cenami
- zobraziť informácií o konkrétnej kryptomene
- zvoliť časové obdobie, v ktorom si chce zobraziť graf vývoja cien
- pridať/odstrániť kryptomeny do/z obľúbených
- pridať / odstrániť kryptomeny do/z sledovaných mien (watchlist)
- vyhľadať kryptomeny podľa zadaného výrazu (časť z názvu alebo používaného symbolu)
- vypočítať hodnotu jednej kryptomeny v jednotkách inej, podľa aktuálneho kurzu

2.2 Návrh aplikácie

Aplikácia bude postavená na trojvrstvovej architektúre. Závislosti medzi triedami v jednotlivých vrstvách bude spravovať knižnica Hilt, ktorá poskytne Dependency Injection.

2.2.1 Používateľská vrstva (UI Layer)

Táto vrstva predstavuje priamo to čo vidí používateľ. Skladá sa z fragmentov a aktivity. Úlohou používateľskej vrstvy je používateľovi zobraziť dáta, ktoré táto vrstva preberá z vyššej vrstvy a na interakciu s používateľom. Aplikácia bude využívať MainActivity, v ktorej sa budú zobrazovať nasledujúce fragmenty: HomeFragment, CoinDetailsFragment, ConverterFragment a SearchFragment.

HomeFragment

Predstavuje hlavnú obrazovku, ktorá sa zobrazí používateľovi po spustení aplikácie ako prvá. Zobrazia sa tu používateľove obľúbené kryptomeny, ktoré má záujem sledovať. Zoznam dostupných kryptomien je možné prepnúť na zoznam kryptomien, ktoré sú používateľom zaradené do sledovaných (watchlist). Po kliknutí na kryptomenu sa používateľovi zobrazí CoinDetailsFragment. Na hlavnej stránke je možné obnovenie stiahnutých mien a opätovné vyžiadanie kryptomien z repozitára.

CoinDetailsFragment

V tomto fragmente sa budú zobrazovať podrobnejšie informácie o kryptomene ako napríklad aktuálna cena, percentuálna zmena za zvolené obdobie, graf historického vývoja cien a podobne. Obdobie vývoja cien je možné zmeniť tlačidlom z dostupných období, pričom vždy si fragment vyžiada nové dáta z repozitára. V grafe je implementovaná funkcia, kde pri podržaní a pohybe prstu používateľa sa vyznačí zvolená hodnota a hodnota sa zobrazí v aktuálnej cene.

SearchFragment

Používateľ dokáže vyhľadávať kryptomeny podľa zadaného výrazu z dostupných kryptomien, napríklad podľa názvu alebo podľa skratky kryptomeny. Po kliknutí na položku sa používateľovi zobrazia detaily o kryptomene. Pri zadávaní vyhľadávaného výrazu nie je potrebné výraz potvrdiť ale po ukončení písania sa dáta automaticky vyžiadajú.

ConvertorFragment

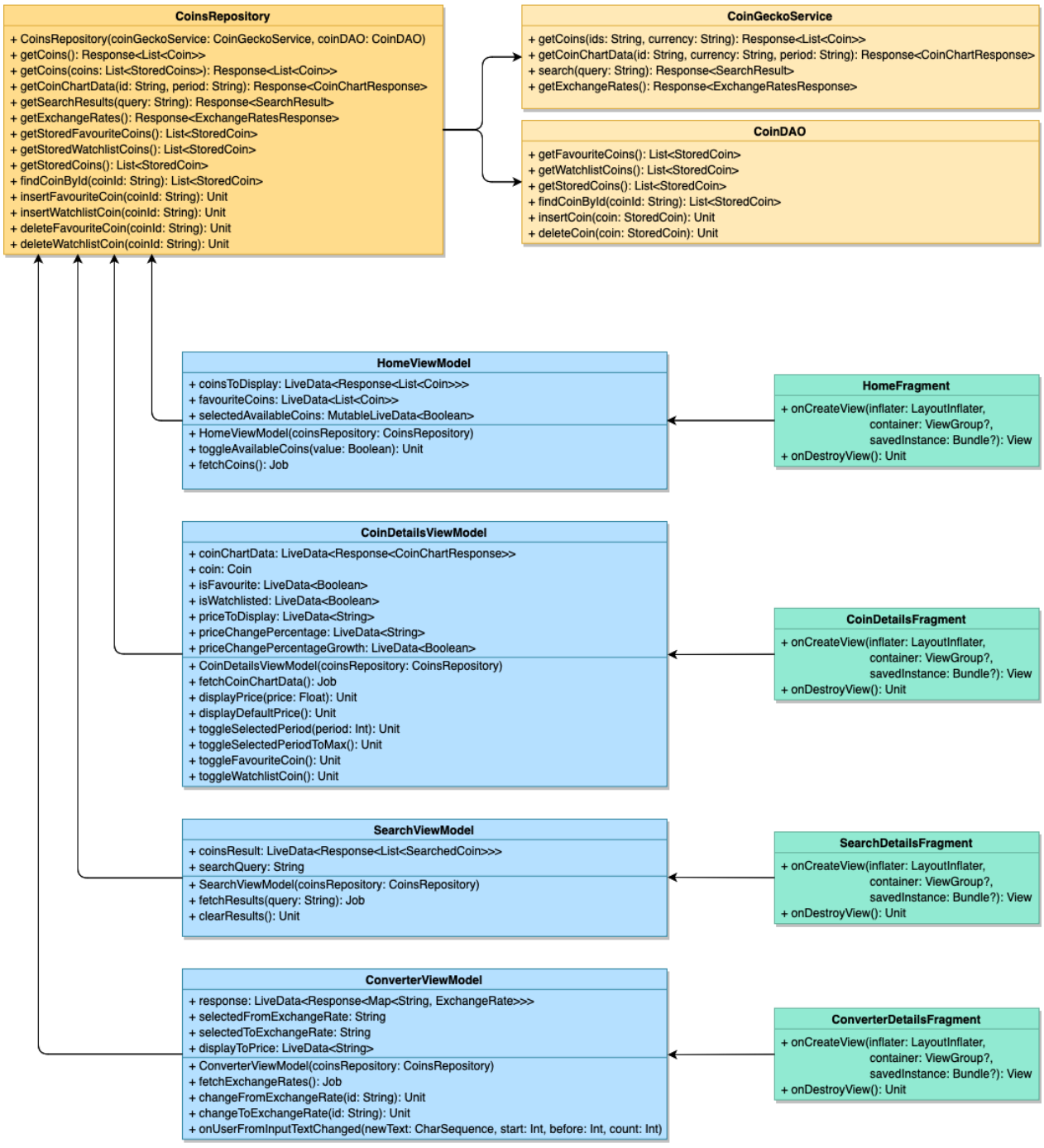
Zobrazuje rozhranie na výpočet prevodu medzi kryptomenami, pričom vždy sa vyžiadajú aktuálne kurzy. Napríklad používateľ zadá hodnotu v jednotkách BTC a zobrazí sa mu vypočítaná hodnota v jednotkách ETC. Prevod je tiež možné vykonať medzi fiat menami a kryptomenami.

2.2.2 Vrstva biznis logiky (Domain Layer)

Slúži na prepojenie používateľskej vrstvy a dátovej vrstvy. Jej realizáciou budú ViewModely a Modely, pričom pre každý fragment bude existovať práve jeden ViewModel. ViewModel bude poskytovať používateľskej vrstve dáta na zobrazenie, prípadne pomocou nej sa budú vyžadovať zmeny v dátovej vrstve. Dáta budú zapuzdrené v samostatných modeloch.

2.2.3 Dátová vrstva (Data Layer)

Dátová vrstva sa skladá z repozitára, lokálnej databázy (knižnica Room) a z API (knižnica Retrofit). Aplikácia bude využívať verejne dostupné API CoinGecko, ktoré pokryje všetky funkcie, ktoré od našej aplikácie požadujeme. Do lokálnej databázy budeme ukladať používateľove obľúbené a sledované kryptomeny. Repozitár je medzivrstva medzi týmito dátovými zdrojmi a biznis logikou.



Obr. UML diagram tried architektúry (vonkajší pohľad)

3 Popis implementácie

V tejto časti sú opísané zaujímavé časti aplikácie, konkrétne ich implementácie.

3.1 Dagger Hilt

3.1.1 Dependency Injection

V našej aplikácií sme použili 3 vrstvovú architektúru, pričom každá vrstva sa skladá z tried. Aby tieto triedy mohli správne fungovať, musia o sebe vedieť, teda sú medzi sebou závislé. Existuje viacero prístupov ako spravovať závislosti medzi triedami. Trieda si môže napríklad vytvoriť inštanciu triedy od ktorej je závislá (kompozícia), v tomto prístupe je ale problém keď aj táto trieda je tiež závislá od ďalšej triedy a pôvodná trieda ju nedokáže inicializovať. Tiež by tu nastal problém, keby túto závislosť potrebuje iná trieda.

V aplikácií je zvolený návrhový vzor Dependency Injection (DI), v ktorom sme oddelili nutnosť vytvárania inštancií tried. Každá trieda si vyžiada svoje závislosti, ktoré automaticky z vonku dostane. Tento proces je zautomatizovaný vyšším mechanizmom. Všetky inštancie, ktoré budeme žiadať sú registrované v servisnom kontajneri.

Knižnica Hilt, ktorá je vybudovaná nad knižnicou Dagger, poskytne DI v Android aplikáciách, tým že nám dáva možnosť použiť servisný kontajner pre triedy, a tiež spravuje životný cyklus objektov. Pre správne fungovanie knižnice je potrebné podľa dokumentácie pridať správnu dependency do gradle a tiež správne pluginy. Každá aplikácia používajúca Hilt musí obsahovať triedu Application s anotáciou *@HiltAndroidApp*.

3.1.2 Závislosti Aktivity a Fragmentov

Všetky aktivity a fragmenty, ktoré sú závislé od iných tried a budú tieto závislosti vyžadovať od Hiltu (v našom prípade hlavne ViewModel objekty), musia obsahovať anotáciu *@AndroidEntryPoint*. Výhoda tohto prístupu je, že nemusíme vytvárať žiadnu ViewModelFactory, stačí iba ViewModel, ktorý si svoje závislosti vyžiada v konštruktore. Hilt inštancie ViewModelov zostrojí a vloží do atribútu aktivity pomocou delegátu *by viewModels()*. Tým sme znížili množstvo tzv. boilerplate kódu, ktorý by sme museli napísať, pretože Hilt pri kompilácií kódu analyzuje triedy a na pozadí generuje kód na vytvorenie ViewModelu. Aby vedel zostrojiť ViewModel, musí obsahovať anotáciu *@HiltViewModel*.

3.1.3 Závislosti ViewModelov a Repozitára

Ostatné triedy, v našom prípade ViewModely a Repozitár, si svoje závislosti vyžiadajú v primárnom konštruktore triedy, pričom treba použiť anotáciu *@Inject* a kľúčové slovo *constructor*. Hilt tieto závislosti vloží do konštruktora pri vytvorení objektov.

3.1.4 NetworkModule

V aplikácií používame externé API, z ktorého získavame dáta – aktuálne informácie o kryptomenách. Na túto sieťovú komunikáciu používame knižnicu Retrofit. Inštanciu hlavnej Retrofit triedy definujeme v objekte NetworkModule, ktorý obsahuje predpisy ako zostrojiť inštancie súvisiace so sieťovou komunikáciou. Účelom týchto modulov, je definovať ako má Hilt vytvoriť inštancie tried, ktoré nevie ako má vytvoriť. V tomto prípade je inštancia Retrofitu vytvorená cez builder, kde uvedieme URL adresu API a nastavíme konvertor odpovedí zo servera na GsonConverter. Tiež tu definujeme metódu na vytvorenie implementácie a inštancie rozhrania *CoinGeckoService*, ktorá obsahuje metódy na vykonanie jednotlivých API requestov a konkrétne endpointy. Obe metódy musia mať anotáciu *@Provides*, ktorou Hiltu povieme, že touto metódou si Hilt vie zostrojiť inštanciu a v prípade potreby ju vie poskytnúť, triedam závislým na danej inštancií. Modul musí obsahovať anotáciu *@Modul*, tým ho zaregistrujeme do Hiltu. Chceme aby sme vždy keď vyžiadame inštancie Retrofit alebo CoinGeckoService z ľubovoľnej triedy aplikácie dostali jednu inštanciu počas celej doby bežania aplikácie, preto pridáme anotáciu *@InstallIn(SingletonComponent::class)* a nad každú metódu pridáme anotáciu *@Singleton* aby sa pri každom vyžiadaní nevytvorila vždy nová inštancia, ale aby sa používala iba jedna.

3.1.5 DatabaseModule

V aplikácií používame aj lokálnu databázu a knižnicu Room, preto závislosti spojené s databázou sú v tomto module. Nastavenie modulu (anotácie) je rovnaké ako v predchádzajúcom module. Definujeme tu inštanciu databázy pomocou Room database buildera a tiež tu definujeme vytvorenie rozhrania *CoinDAO* na komunikáciu s databázou.

3.2 Proces získania a zobrazenia výsledkov

Po spustení aplikácie sa zobrazí ako prvý HomeFragment. Následne sa z fragmentu zavolá vo HomeViewModely metóda na stiahnutie zoznamu kryptomien označená ako suspend, čiže ju musíme spustiť v rámci coroutiny, aby sme nezablokovali používateľské rozhranie. Táto metóda volá metódu z repozitára a tá zase z CoinGeckoService. Odpoveď zo servera nám Retrofit namapuje na príslušný model (zoznam inštancií triedy Coin) a odpoveď odošle do atribútu typu LiveData. Z fragmentu sledujeme zmeny tohto atribútu (observer) a keď príde úspešná odpoveď spolu s dátami tak výsledok zobrazíme cez dynamické rozloženie (v zozname) pomocou RecyclerView.

Dáta z jednotlivých kryptomien zobrazujeme pomocou Data Bindingu. Na hlavnej obrazovke máme dve takéto dynamické rozloženia (zoznamy) jeden na zobrazenie obľúbených kryptomien a druhý na zobrazenie dostupných kryptomien, respektíve kryptomien označených ako watchlisted (podľa zvolenej možnosti). V odpovedi z API má každá kryptomena aj URL adresu na svoje logo, takže v každej položke kryptomeny zobrazujeme aj jej logo, ktoré za nás stiahne a zobrazí knižnica Glide. Po kliknutí na položky RecyclerViewera sa dokážeme navigovať na CoinDetailsFragment s detailmi o zvolenými kryptomenami.

Zoznam kryptomien je možné opätovne stiahnuť z API pomocou SwipeRefreshLayoutu, ktorý po potiahnutí prstu z vrcholu zoznamu nadol opätovne spustí metódu na stiahnutie dát.

Rovnaký princíp zobrazenia dát je implementovaný vo všetkých ostatných fragmentoch a ViewModeloch kde dochádza k sieťovej komunikácií s API.

3.2.1 Koncept zapečatených tried (sealed classes)

Priebeh sieťovej komunikácie vieme rozdeliť na určité stavy. Na začiatku sa odošle požiadavka na API a čakáme na odpoveď, čiže sme v stave čakania (Waiting). Keď príde odpoveď z API, môže byť buď úspešná (Success) alebo neúspešná (Error). Chyba môže tiež nastať ešte pred samotným odoslaním požiadavky, napríklad ak nemáme internetové pripojenie. O tomto procese chceme používateľa informovať, napríklad v stave čakania chceme zobraziť indikátor čakania. V prípade chyby chceme zobraziť chybovú hlášku s chybou aká nastala a v prípade, že sme dostali úspešnú odpoveď s dátami, chceme ich používateľovi zobraziť.

Na informovanie používateľa by sme potrebovali atribúty vo ViewModely, pričom minimálne jeden by musel byť LiveData, kde uložíme v akom stave sa nachádzame. V prípade úspešnej odpovede potrebujeme

atribút na dáta z API a ak nastane chyba potrebujeme atribút na uchovanie typu chyby respektíve kódu alebo chybovej správy. Potom by sme vo fragmente mohli sledovať atribút so stavom a pri zmene spraviť potrebné UI zmeny.

V našej aplikácií sme použili koncept zapečatených tried, kde nám stačí vo ViewModely jediný atribút, typu LiveData<Response<T>>. Response je zapečatená trieda, ktorú nedokážeme priamo inštanciovať pomocou konštruktora. Táto trieda obsahuje pevný počet podtried, dediace z triedy Response, z ktorých už vieme vytvoriť inštancie. Podtriedy vyjadrujú jednotlivé stavy sieťovej komunikácie (Success, Error, Waiting). Stav Waiting nepreberá žiadne parametre. Stav Success obsahuje dáta z úspešnej odpovede a stav Error obsahuje typ chyby. Do atribútu s odpoveďou stačí ukladať príslušnú inštanciu, ktorá obsahuje aj informáciu o stave a aj príslušné dáta. Potom vo fragmente kde sledujeme zmenu LiveData atribútu, použijeme *when* výraz a pre každý stav spravíme príslušné UI zmeny.

3.2.2 Filtrovanie výsledkov na základe dát lokálnej databázy

Na domovskej obrazovke zobrazujeme zoznam dostupných kryptomien. Na pozadí sa z API stiahne prvých 100 kryptomien podľa “market cap rank“ a zobrazí ich do zoznamu. Ak používateľ označil nejaké kryptomeny ako obľúbené alebo ako sledované (watchlist coins), potom potrebujeme z lokálnej databázy pomocou knižnice Room a vytvoreného rozhrania CoinDAO získať zoznam týchto kryptomien a na základe odpovede z API vyfiltrovať kryptomeny na zobrazenie do príslušných zoznamov.

Problém môže nastať ak si používateľ pomocou vyhľadávania nájde nejakú kryptomenu, ktorá nie je zahrnutá v zozname dostupných kryptomien (prvých 100 podľa “market cap rank“) a označí ju ako obľúbenú alebo sledovanú. Tieto kryptomeny musíme dodatočnou požiadavkou stiahnuť z API aby používateľ videl naozaj všetky kryptomeny, ktoré si v minulosti uložil.

3.2.3 Ošetrenie chybových stavov sieťovej komunikácie

Keďže používame sieťovú komunikáciu potrebujeme ošetriť už vyššie spomínané chybové situácie (používateľ nie je pripojený na internet, prečerpané požiadavky na API, iné sieťové anomálie). Chybové situácie sú rovnaké vo všetkých štyroch ViewModeloch, preto tu vzniká duplicita kódu na ošetrenie. Preto je tu dvojica pomocných funkcií *handleNetworkCall* a *handleIfNotSuccessful*, kde tento duplicitný kód premiestnime a následne cez lambda funkciu posielame kód, ktorý je už unikátny pre každý ViewModel.

Rovnaká situácia s duplicitným kódom nastáva aj vo fragmente, preto je vytvorená pomocná funkcia *displayErrorSnackBar*, ktorá zobrazí chybovú hlášku používateľovi s konkrétnou chybou a možnosťou skúsiť zopakovať požiadavku na API.

3.2.4 Automatické spustenie akcie vyhľadávania

Aplikácia umožňuje vyhľadávať kryptomeny podľa zadaného výrazu. Slúži na to SearchFragment, obsahujúci vyhľadávacie pole a RecyclerView na zobrazenie výsledkov. Aby sme zlepšili UX je tu funkcia, ktorá po zadaní výrazu automaticky odošle požiadavku na API s hľadaným výrazom bez nutnosti dodatočného stlačenia tlačidla. Túto funkcionalitu umožňuje knižnica RxBindings, ktorá sleduje zmeny zadaného textového poľa a po zmene vstupu a po následnej nečinnosti v určitý čas (700 ms) sa spustí nami zadaná akcia – odoslanie zadané výrazu na API.

3.3 Dynamické zmeny používateľského rozhrania

3.3.1 Graf vývoja ceny kryptomeny

Po kliknutí na položku kryptomeny na hlavnej obrazovke alebo vo vyhľadávaní užívateľa navigujeme na CoinDetailsFragment. Následne z API stiahneme vývoj cien za posledných 24 hodín a dáta zobrazíme pomocou knižnice MPAndroidChart. Z dát vypočítame ešte percentuálnu zmenu za 24 hodín, zobrazíme ju používateľovi a podľa toho či je zmena kladná alebo záporná nastavíme pozadie rámčeka. Zmenu zobrazujeme pomocou Data Bindingu.

Používateľ má možnosť vybrať si časové obdobie, ktorého graf a percentuálnu zmenu chce vidieť. Predvolene je to 24 hodín ale na výber má ešte 7, 14, 30 dní, rok a maximálny interval aký poskytuje API. Po každom výbere sa z API stiahnu dáta pre príslušné obdobie a aktualizujeme dáta v grafe a tiež percentuálnu zmenu.

Používateľ má k dispozícií možnosť zobraziť si presnú cenu v danom časovom okamihu. Keď podrží prst na grafe a prípadne s ním bude po grafe hýbať, aktuálna cena kryptomeny sa zmení na cenu, ktorá prislúcha danému miestu na grafe, kde má používateľ prst. Ak prst z grafu uvoľní, cena sa nastaví na pôvodnú cenu, čiže cena na konci zvoleného časového obdobia. Toto nám umožňuje použitie Data Bindingu a LiveData, z ktorého sa zobrazuje cena používateľovi.

3.3.2 Výpočet ceny meny v jednotkách inej meny

V menovej kalkulačke má používateľ možnosť vypočítať si hodnotu ľubovoľnej fiat alebo kryptomeny v inej fiat alebo kryptomene. Na začiatku sa z API stiahnu dostupné menové kurzy, ktoré sú vyjadrené v jednotkách BTC (Bitcoin). Dostupné meny vložíme do dvoch vyberateľných vstupných políčok. Používateľov vstup prepojíme s LiveData atribútom a pomocou Data Bindingu zobrazíme vypočítanú hodnotu. Novú cenu vypočítame tak, že zoberieme používateľov vstup, vydelíme ho hodnotou, ktorá prislúcha mene, z ktorej počítame cenu. Tým sme získali hodnotu v BTC a následne túto hodnotu vynásobíme hodnotou, ktorá prislúcha mene, na ktorú cenu prevádzame. Kvôli pohodlnosti používania je prepočet vykonaný automaticky po zadaní cifry čísla bez nutnosti dodatočného stlačenia tlačidla. Prepočet musí automaticky nastať aj pri zmene meny, inak by kalkulačka ukazovala starý výsledok.

4 Zoznam použitých zdrojov

Android developer dokumentácia

https://developer.android.com

Prevzatý koncept zapečatených tried na spracovanie sieťovej komunikácie

https://proandroiddev.com/modeling-retrofit-responses-with-sealed-classes-and-coroutines-9d6302077dfe

Dependency Injection v Androide

https://developer.android.com/training/dependency-injection

Dokumentácia knižnice Dagger Hilt

https://dagger.dev

Dokumentácia knižnice Retrofit

https://square.github.io/retrofit/

Dokumentácia knižnice MPAndroidChart

https://weeklycoding.com/mpandroidchart-documentation/

Zhrnutie ako použiť knižnicu RxBindings

https://guides.codepath.com/android/RxJava-and-RxBinding

Ako zmeniť farbu komponentu “chip“

https://stackoverflow.com/questions/51089150/set-com-google-android-material-chip-chip-selected-color

Ako zmeniť vzhľad okraja vstupné poľa s výberom možnosti

https://stackoverflow.com/questions/24823983/autocomplete-textview-with-oval-shape-corners